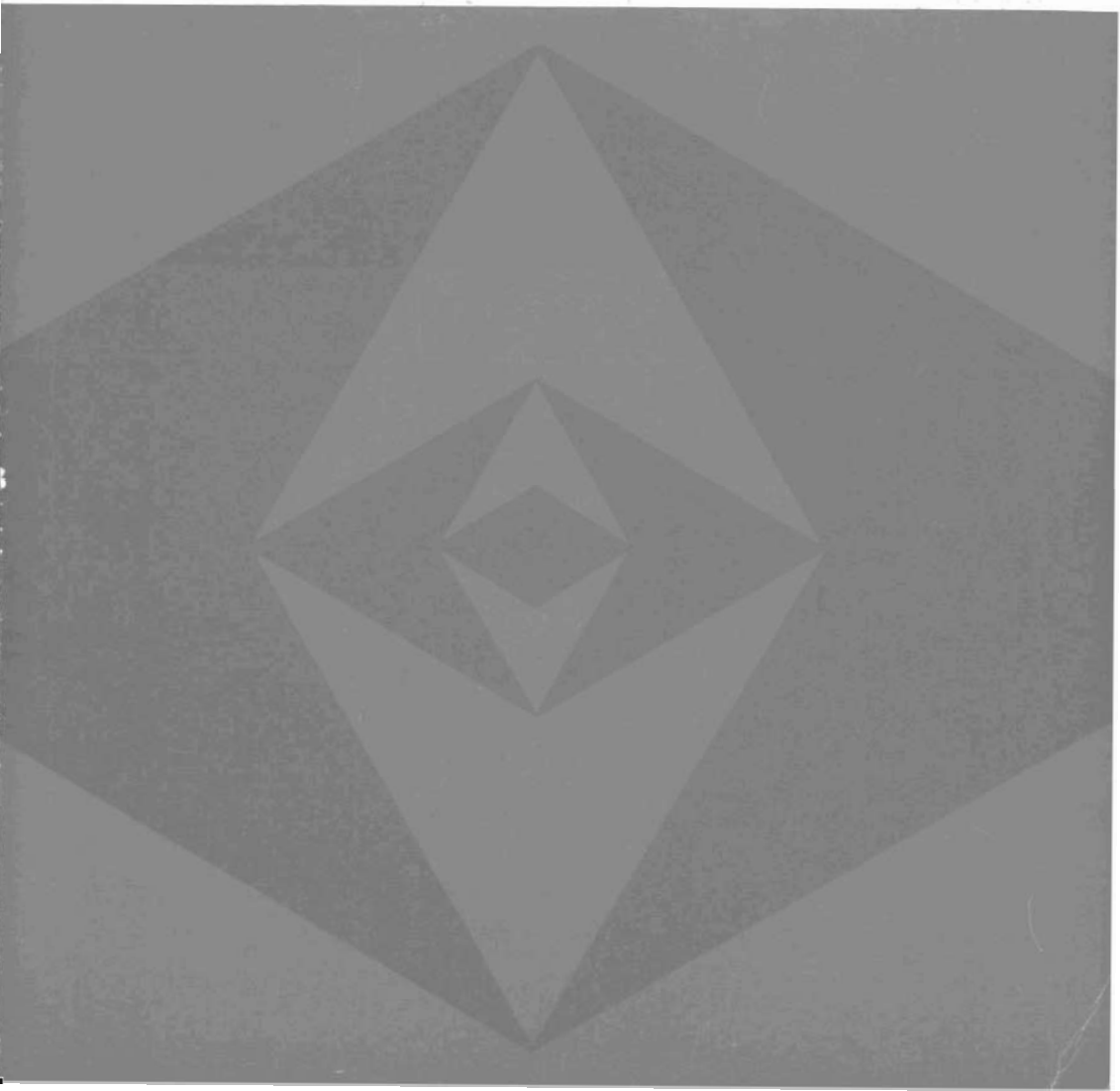
Janis Bubenko jr
Tomas Ohlin

Introduktion till operativsystem

STUDENTLITTERATUR

DEL 1



Janis Bubenko
Tomas Ohlin

Introduktion till operativsystem

Del 1

Studentlitteratur Lund
Akademisk Forlag København

Sjätte tryckningen
Printed in Sweden
Studentlitteratur
Lund 1975
ISBN 91-44-05221-9

Det skal et par dumheder
med i en bog
for at ogsaa de dumme
skal syns, den er klog

GRUK
(Piet Hein)

INNEHÅLL

FÖRORD

Tillgänglig svensk såväl som utländsk litteratur avseeende operativsystem är knapphändig - om man bortser från en mängd handböcker, som beskriver speciella tillverkares produkter. Det finns många artiklar i fackpressen, som behandlar isolerade problemställningar som kan sägas inrymmas under begreppet "operativsystem". Däremot kan vi idag (1970) ej peka på något verk, där ämnesområdet behandlas på ett samlande sätt.

Inom undervisningen i informationsbehandling och även bland praktiskt ADB-verksamma har behovet av allmän, introducerande litteratur i detta område länge existerat. Med viss tvekan gav vi oss på uppgiften att försöka åstadkomma en bok om operativsystem. När vi nu framlägger resultatet av våra vedermödor, gör vi det med en viss känsla av otillfredsställelse. Visst var vi medvetna om att området var svårt att behandla på ett generellt sätt, men dock anade vi vid starttidpunkten föga de perioder av modlöshet, som vi under arbetets gång skulle uppleva. Det har för oss kommit att framstå allt klarare varför bristen på samlande litteratur om operativsystem är så påtaglig. Anledningarna är bland annat

- ämnesområdet är vagt definierat, dynamiskt och under snabb utveckling. Nya system, för att inte tala om versioner av existerande system, frisläpps tätt.
- någon enhetlig och allmänt tillämpad underliggande begreppsapparat och teoribildning kan ej anses existera.
- terminologiproblemen är dominerande - standardiseringssträvanden kan ej skönjas.
- operativsystems arbetsprinciper är, åtminstone på en mindre grov nivå, beroende av olika tillverkares datorprodukters särdrag m m.

Att någotsånär behärska ett fåtal tillverkares operativsystem är därför knappast en tillräcklig förutsättning för att skriva en uttömmande bok om operativsystem. Vi vill därför starkt understryka att vi, trots studier av olika tillverkares op-system, inte gör anspråk på att ha behandlat området generellt och uttömmande. Läsaren blir sålunda inte "os-specialist" genom att inhämta innehållet i denna bok.

Vår förhoppning är dock att boken skall anses beskriva vissa fundamentala os-principer och att den skall kunna ge den mindre erfarne databehandlaren en bakgrund, som dels underlättar specialstudier av olika speciella

Medförfattarens T Ohlin arbete på denna skrift är till sin merpart finansierat av Statens Naturvetenskapliga Forskningsråd.

op-system och dels gör att dessa studier kan bedrivas med en viss kritisk blick.

Av ovanstående skäl har vi valt att ge boken titeln "Introduktion till operativsystem" och ej, som vi från början avsåg, "Operativsystem". Vi vill också påpeka, att vi har strävat att belysa operativsystem ej enbart som en samling av styr-, övervaknings- och servicefunktioner utan även som en betydelsefull komponent vid konstruktion och drift av informationsbehandlingssystem.

Bokens innehåll sönderfaller vid betraktande i två delar:
1. Utvecklingen av operativsystemområdet.
2. Operativsystemprinciper och mål,

Del 1 är av översiktlig natur och belyser i någorlunda kronologisk ordning den utveckling, som har ägt rum dels på maskinvarusidan, dels på programvarusidan, och som har lett fram till operativsystem som idag anses som en fundamental komponent hos varje datorinstallation[1]. Kapitlet om maskinvaruutvecklingen kan kanske anses väl omfattande. Vi har dock valt detta omfång dels för att svensk litteratur som belyser området är knapphändig, och dels för att operativsystemen kan och bör ses som en utvidgning av de maskinella datorfunktionerna. En dos datorhistoria torde f ö kunna ses som ett värdefullt "allmänbildande" komplement till övrig existerande specialundervisning inom ADB-området.

Vi vill framhålla, att kapitlet om maskinvarans utveckling i huvudsak författats under år 1969, och därför speglas situationen fram till detta år. Utvecklingen inom detta område är mycket snabb, och publicerat material blir på kort tid inaktuellt. Vi vill därför be läsaren ha överseende med det faktum att flera intressanta datormodeller, som presenterats fr o m år 1970, inte har belysts i kapitlet.

Del 2 går mer i detalj in på olika operativsystemprinciper och mekanismer. Vi behandlar där kapitelvis
- diverse grundläggande begrepp
- olika driftsformer (ur användarens synvinkel)
- jobbövervakningsfunktionen
- jobbprofiler och inplanering av jobb
- övervakning och styrning av processer (styrprogramfunktioner)
- datatransport- och filhanteringssystem

---

1) Vissa avsnitt härav är med tillstånd från ACM hämtade från S. Rosens och R.F. Rosins artiklar i ACM Computing Surveys, vol. 1, nr. 1.

- programkonstruktion och serviceprogram
- mål för operativsystem

Vi har ansett det lämpligt att fysiskt separera de två delarna, och publicerar därför innehållet i två volymer. Viss innehållsmässig överlappning av innehållet kan säges existera. Bl a därför kan de läsas oberoende, dvs för att studera del 2 förutsättes ej innehållet i del 1, även om det kan vara en fördel ur överskådlighetssynvikel. För att med förståelse studera boken krävs ej några speciella "akademiska" förkunskaper. Läsaren bör dock vara bekant med elementa inom programmering, databehandling och datasystemkonstruktion. Del 1 kan exempelvis läsas efter en introducerande kurs i informationsbehandling kompletterat med en laborativ programmeringskurs. Del 2 kan beroende på studieinriktning läsas direkt efter del 1 eller sparas till kursmoment som följer efter några laborativa moment i databehandlingsteknik och systemering. I universitetsmiljö skulle detta motsvaras av att del 1 ingick på 20-poängsnivå och del 2 på 40-poängsnivå i informationsbehandling med inriktning administrativ databehandling eller numerisk analys. Den datalogiinriktade skulle kunna läsa både del 1 och del 2 på 20-poängsnivå.

Vi är medvetna om att delar av bokens innehåll ej kommer att vara okänt stoff för i synnerhet de läsare som redan besitter någon tids erfarenhet från praktisk verksamhet inom ADB-området. Vi är också medvetna om att boken ej har mycket att ge till erfarna operativsystemprogrammerare. Vi tror dock att det finns en icke ringa mängd i dag yrkesverksamma programmerare och systemerare som kan finna det intressant att se området belyst från en annan synvinkel än den som impliceras av den lokala miljön och de lokala problemställningarna.

Eftersom detta är ett första försök till en bok på svenska om det mystikomgärdade området "operativsystem" hoppas vi att läsaren har överseende med diverse inkonsekvenser vad gäller terminologi och försvenskning av engelska facktermer. För det aktuella terminologiområdet finns oss veterligt ej någon svensk standard etablerad. Vi ber också om överseende med den frekventa förekomsten av orden "ofta", "normalt" och "vanligtvis". Härigenom har vi försökt antyda att det förekommer andra lösningar, metoder och betrakelsesätt än de vi beskriver.

Med tillfredställelse skulle vi notera om denna bok skulle inspirera läsaren till specialstudier och publicering av mer detaljerade arbeten inom det omfattande område som vi berör. Det återstår åtskilligt att bearbeta inom området operativsystem, för vår del får det dock räcka för tillfället!

Janis Bubenko Tomas Ohlin

1. MASKINVARUUTVECKLINGEN

Inledning

Detta avsnitt gör på intet sätt anspråk på fullständighet. Härför skulle krävas omfattande specialstudier, ledande till ett digert resultat, och då maskinvaruutvecklingen ur viss synpunkt endast perifert tangerar denna boks egentliga syfte har vi valt ett till omfånget måttligt beskrivningssätt. Avsikten med kapitlet är att ge en icke alltför detaljerad överblick över datorutvecklingen. Vi har ansett det motiverat att inte helt utesluta ett avsnitt om maskinvarans utveckling, av flera skäl:

1. En överblick på svenska språket, om än kortfattad, är av intresse i undervisningssammanhang.

2. Programvaruutvecklingen i stort, som utgör en naturlig inledning till operativsystem, är klart anknuten till maskinvaruutvecklingen.

3. Operativsystem kan till funktion och arbetssätt ses som utvidgningen av maskinvaran, ett förhållande som sannolikt kommer att accentueras i framtiden.

Den version av utvecklingen, som i detta kapitel kommer att ges, baserar sig på ursprungsmaterial av skiftande karaktär. Även om vi sökt hålla en så strikt och källtrogen linje som möjligt, vill vi inte utesluta möjligheten att felaktigheter kan ha kommit på pränt. Detta kan härröra dels från att datorhistorien ännu inte beskrivits fullständigt och dels från det faktum att ett flertal datorleverantörer synes väl så tystlåtna vad gäller information om t ex maskinmodeller som inte i alla stycken visat sig uppfylla förväntningar och utfästelser. Vi har dock velat nämna något om så många intressanta objekt som möjligt trots att våra informationskällor i vissa fall varit begränsade.

Vid beskrivning av datorhistoria ställs man inför ett avvägningsproblem. Skall en strikt tidsmässig behandlingsprincip väljas, eller skall den funktionsmässiga utvecklingen beskrivas? Den strikta tidsmässigheten medför svårigheter för läsaren att få grepp om de funktionsmässiga sambanden, och vice versa. Vi har sökt finna en kompromiss, dock med tonvikt på tidsmässigheten. Vi är medvetna om den ryckighet som denna kompromiss må ge intryck av, och ber läsaren om överseende härmed.

En gängse indelning av datorer innebär klassning som "första, andra eller tredje generationens-maskiner". Vi har undvikit denna i vårt tycke mindre lämpliga indelning, dels emedan vi knappast anser användning av radiorör, transistorer eller integrerade kretselement tongivande för ett datorsystems funktionssätt och dels på grund av att kommande "generationer" svårt låter sig definieras utgående från denna bas.

1.1. Den tidiga utvecklingen

Sporadiska idéer

Bakgrunden till vår tids datorer lades tidigt, i samband med utvecklandet av mekaniska apparater för beräkningsändamål. Redan 1641 konstruerade den då 17-årige Blaise Pascal, den sedermera framstående franske filosofen och matematikern, en Machine Arithmétique som kunde addera och subtrahera med hjälp av tandade sifferhjul. Pascal skänkte ett exemplar av sin maskin till drottning Kristina av Sverige. Några årtionden senare, 1671, framställdes en förbättrad maskin av Gottfrid Wilhelm Leibniz, den banbrytande tyska matematikern. Leibniz´ maskin kunde hantera alla fyra räknesätten.

Information från 1700-talet om uppfinningar av intresse i detta sammanhang är begränsad. Troligt är att flera förbättrande versioner av Pascals och Leibniz´ idéer såg dagens ljus. Det dröjde emellertid till 1820 innan väsentliga steg kunde tas. Engelsmannen Charles Babbage, en dedikerad matematiker-professor och uppfinnare, lade då fram ritningar till en s k differensmaskin, en räknemaskin som skulle vara särskilt lämpad för tabellräkning, med syfte på användning bl a i astronomi och navigation. Babbage lyckades erhålla substansiellt ekonomiskt stöd från den engelska regeringen för sin maskin, närmare bestämt 17000 pund planerat över 20 år, men innan maskinen hunnit helt färdigställas lanserade han, sporrad av myndigheternas intresse, nya planer på en "analytisk maskin" av mera kapabel utformning. Den analytiska maskinen skulle uppvisa stora likheter med dagens datorer. Den skulle vara uppdelad i fyra delar: räkneenheten, minnet (för lagring av mellanresultat), kontrollenheten (för styrning av processerna) samt skrivaren (för utmatning av resultat). Dessutom skulle maskinen vara hålkortsstyrd. De ekonomiskt ansvariga började tvivla på dessa i deras tycke alltför högtflygande planer, och sedan det visat sig att differensmaskinens färdigställande skulle bli mer kostnadskrävande än beräknat, drogs stödet in. Babbages maskiner blev aldrig fullständigt

realiserade, samtiden ansåg honom uppenbarligen som något av en fantast, med idéer huvudsakligen utan praktisk användbarhet. Hans ritningar och även några delkonstruktioner finns emellertid bevarade, och det kan numera fastslås att han kan anses ha varit hundra år före sin tid och bör betraktas som en banbrytare inom området kring beräkningsautomater.

Endera som en anvisning om vår källinformations.ofullkomlighet eller som bevis för utvecklingens långsamma och diskontinuerliga förlopp kan noteras att hålkorts/hålremstekniken, född ca 1725 då den franske vävaren Bouchon lanserade idén att använda en pappersrulle försedd med hål för att styra vävtrådarna, inte hörde av sig förrän 1808. Då konstruerade landsmannen Jacquard en sedermera omtalad vävstol, som mera fullständigt styrdes av hålförsedda pappkort.

Det är intressant att notera årtiondet 1870-talet som tidpunkt för tillkomsten för den svenska ingenjören Odhners uppfinning av en pålitlig räknemaskin, som kom att utgöra grunden för bl a den svenska kontorsmaskinindustrin.

Trots Babbages insyn om hålkortsteknikens möjligheter var det först tysk-amerikanen Hollerith som på 1880-talet i större skala började använda sig av dessa databärare. Hans hålkortsmaskiner, baserade på elektrisk kortavläsning, användes bl a vid bearbetningen av resultatmatetialet från den amerikanska folkräkningen 1890.

Vi kan fastslå att först 1970-talet kommer att bli tiden för ett visst avlägsnande från hålkorten som huvudsakligt databärande medium. I sanning ett hållfast medium, med hänsyn till datateknikens övriga utveckling, framför allt de två senaste decennierna. Det är enkelt att göra ett hål i ett papper, men svårare att radera ut det.

Elektromagneternas intåg

En relativt kort men betydelsefull era inträdde kring mitten av 1930-talet i och med att den unge tyske ingenjören K. Zuse startade konstruktionen av en rad komplicerade automatiska räknemaskiner. De första, kallade Z1 och Z2, var rent mekaniska och framställdes medan Zuse var student tillsammans med ett litet antal kamrater, men från och med 40-talets början fick elektromagnetiska reläer överta det mekaniska arbetet. Modellen Zuse Z3 från 1941 blev en helt funktionsduglig maskin, uppbyggnadsmässigt med anor från Babbages idéer över ett århundrade tidigare. Denna Zuses maskin adderade på 0,07 sek och multiplicerade på 5 sek. Z3 använ-

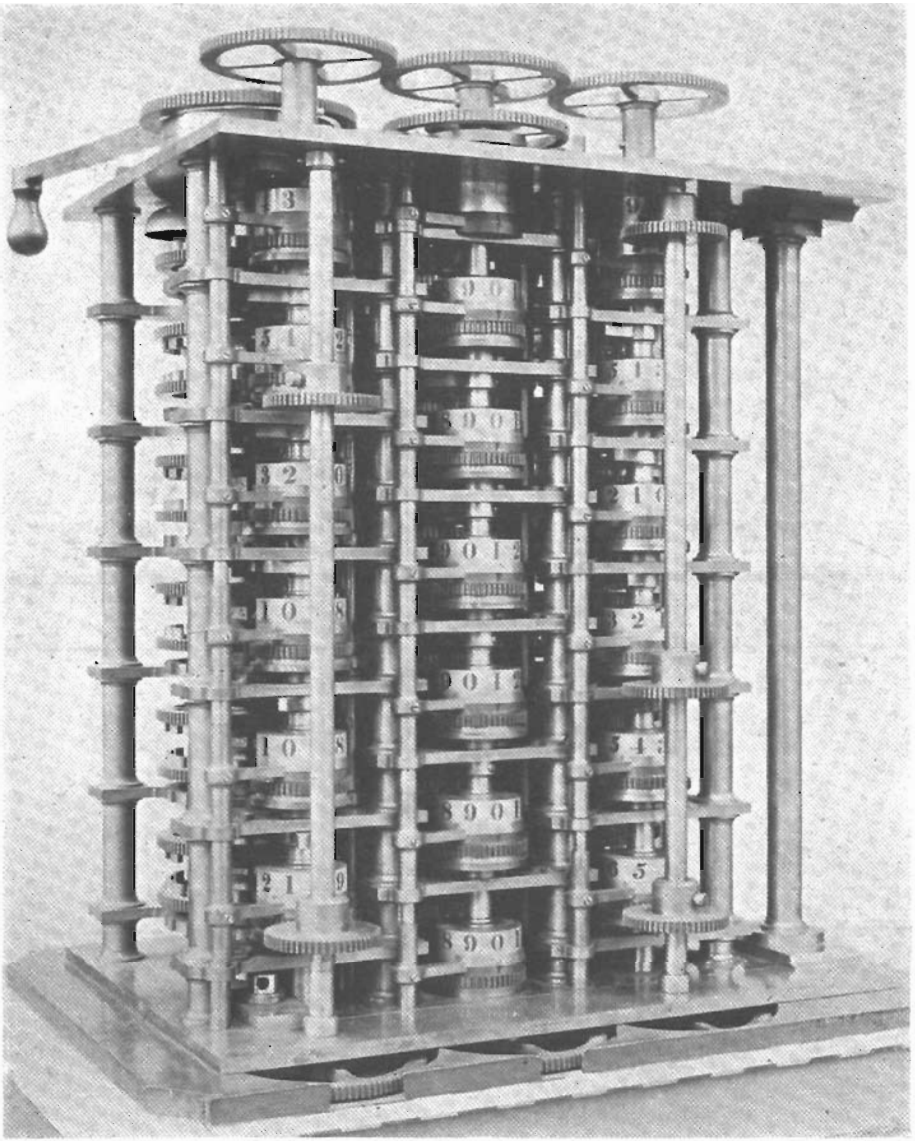


Fig 1.1:1. En del av Babbage´s differensmaskin.

(Ur B.V. Bowden, ed.: Faster than Thought. Med tillstånd av: Sir Isaac Pitman and Sons Ltd, London.)

de över två tusen reläer, dess minne var dock rent mekaniskt. Den förstördes emellertid fullständigt under de allierades bombningar av Tyskland. En kopia kan dock beskådas på Deutsches Museum i München.

Snart därefter startade utvecklingen i USA. Howard Aiken vid Harvard University påbörjade ca 1937 ett samarbete med IBM angående framställning av en kapabel räkneautomat, sedermera kallad Mark I. Dess färdigställande 1944 fick flera direkta efterföljare, fram till ca 1948. Mark I, vars tidigare benämning var ASCC (Automatic Sequence Controlled Calculator), hade additionstiden 0,3 sek och multiplikationstiden 6,0 sek samt var försedd med ett minne på 72 ord (tal). Maskinen behövde ca 1 minut för att framräkna en logaritm. Nämnas kan också de sex olika maskiner som utvecklades 1939-46 av George Stibitz och medhjälpare vid Bell Laboratories.

Den kugghjuls-, vipparms- och reläteknik, som var den gemensamma nämnaren till de i detta avsnitt nämnda maskinerna, utgjorde ett intressant studieobjekt för svensken Stig Ekelöf, som 1939 företog en studieresa till USA för att informera sig, och Sverige, om utvecklingen. Strax efter kriget genomfördes flera dylika resor med svenska experter i syfte att inhämta erfarenheter om den amerikanska positionen. Man hade för avsikt att, med ekonomiskt stöd från den svenska regeringen, söka bana väg för konstruktion av en svensk maskin, emedan USA ställt sig negativ till maskinförsäljning till utlandet.

Det svenska organ som bildades 1948 för att samordna ansträngningarna, Matematikmaskinnämnden, arbetade till att börja med i statlig medvind. Redan 1949 kunde man annonsera planer på en relämaskin som start för den inhemska utvecklingen. I mars 1950 invigdes maskinen, kallad BARK (Binär Automatisk Relä-Kalkylator), framställd under Conny Palm. BARK och dess efterföljare gav Sverige en flygande start i utvecklingen, och maskinen utförde fram till 1955 värdefullt beräkningsarbete bl a åt Marinförvaltningen.

Elektroniken tar över

Från och med andra världskrigets slut övertog USA alltmer initiativet. Den första stora elektroniska maskinen, ENIAC (Electronic Numerical Integrator and Calculator), framställdes 1945-46 av J. P. Eckert och J. Mauchly vid University of Pennsylvania för USAs krigsministeriums räkning. Denna maskin hade add/mult-tiderna 0.2 ms resp 2.6 ms. I och med att de elektronmagnetiska reläerna ersattes av radiorör uppnåddes en kraftig snabbhetsökning. Problem saknades dock inte. Maskinen inne-

höll ca 18 000 rör, och prognoserna för pålitligheten hos en elektronisk konstruktion av denna omfattning var före maskinens tillkomst pessimistiska. Det berättas om hur belysningen i västra Philadelphia dämpades när ENIAC startades upp, och hur startströmmen varje gång brände ut tre eller flera rör. ENIAC upptog en golvyta på 9 x 15 m och vägde 30 ton. Maskinen var dock i framgångsrik drift ända till slutet av 1955. ENIAC var en sladdprogrammerad maskin, dvs programbyte skedde genom byte av sladdanslutningar i maskinens inre.

Ett namn som måste nämnas i principutvecklingssammanhang är A. M. Turing. Denne engelska logiker publicerade redan 1936 en artikel där huvuddragen av en matematisk teori för datorer angavs. Däri angivna tankar ligger till grund för den moderna automatteorin.

Strax efter färdigställandet av ENIAC, 1948, presenterade den berömde matematikern John von Neumann en rapport, som beskrev en ny princip för logisk datoruppbyggnad. Denna princip kom att få fundamental betydelse för hela utvecklingen därefter. Principen byggde på idén att i samma minne lagra både det program som utgjorde aktuell instruktionsföljd samt de data som programmet skulle bearbeta. Tidigare hade program och data lagrats i helt skilda delar av maskinen. von Neumans princip lade grunden för en expansion för datatekniken av enastående art. Med fog kan samtliga dagens maskiner sägas arbeta i enlighet med hans tankar.

Den första kompletta maskin i vilken von Neumanns idéer förverkligades, EDSAC (Electronic Delay Storage Automatic Calculator) kördes igång 1949 vid Cambridge University i England av M. Wilkes och W. Renwick, vilka tidigare tillbringat tid hos Eckert och Mauchly vid Pennsylvania-universitetet i USA. EDSAC blev därmed en banbrytare, den första arbetande maskin i världen med program och data i samma minne. Minnet bestod av kvicksilverfyllda rör, genom vilka ljudimpulser cirkulerade. En viss typ av katodstrålerör kom senare till användning (innan ferritkärnminnena tog över) under namnet Williams-rör, uppkallade efter sin konstruktör, F.C.Williams.

Den ungefär samtidigt som EDSAC påbörjade Pennsylvania-maskinen EDVAC (Electronic Discrete Variable Computer) fördröjdes på grund av att Eckert och Mauchly lämnade universitetet i Pennsylvania för att bilda ett eget företag, the Eckert-Mauchly Computer Corporation. EDVAC kunde dock köras igång strax efter EDSAC, mot slutet av 1949.

Utvecklingen accelererar

Från andra världskrigets slut och till 50-talets början startades härförutom i USA en mångfald maskinkonstruktionsprojekt, huvudsakligen uni-

versitetsanknutna. Vid the Institute for Advanced Study i Princeton, von Neumanns arbetsplats, färdigställdes 1952 "the IAS Computer", en snabb maskin med direktadresserbart (inre) minne och parallell binär aritmetik. Denna fick ett flertal efterföljare, ILLIAC vid University of Illinois, JOHNIAC vid Rand Corporation, MANIAC vid Los Alamos, WEIZAC vid Weizman-institutet i Israel, och andra.

Vid Massachusetts Institute of Technology producerades ca 1950 en konstruktion som gavs namnet MIT Whirlwind I. Denna intressanta maskin kunde multiplicera två 16-bits tal så snabbt som på 16 mikrosekunder. Två aspekter på Whirlwinds betydelsefullhet må nämnas. Den första innebar det utomordentligt värdefulla i färdigställandet av en dokumentsamling, vari maskinens logiska uppbyggnad komplett beskrevs. Dessa dokument nådde en tämligen vid spridning, och hjälpte till att sprida information om datorkonstruktionsteknik på ett mycket värdefullt sätt.

Viktigt i samband med MIT-projektet kan dessutom sägas vara konstruktion av ett magnetiskt kärnminne av den typ som än i början av 70-talet dominerar bland primärminnen. Vid MIT byggdes dessutom en speciell Memory Test Computer för att prova ut det nya minnet innan det installerades i Whirlwind.

Positionen i England

I England framställdes av firman Metropolitan Vickers Ltd så tidigt som 1943 en beräkningsanläggning, som arbetade med elektromagnetiska reläer. Denna maskin togs omedelbart i bruk av den engelska krigsindustrin - världskriget pågick då ännu som hetast.

Såväl på universiteten i Cambridge och London som i firman Elliot Brothers (senare Elliot Automation) arbetades det med datorproblem från 1946. Redan 1947 byggde en liten forskningsgrupp vid Manchesteruniversitetet i samarbete med Ferranti Ltd sin första dator. En på samma ställe framställd efterföljare till denna, i drift från 1949, arbetade med primärminne av Williamsrörtyp som kompletterades även med ett hjälpminne av magnetisk trumtyp. På denna trumma organiserades data i block av fast längd, kallade "pages". Här lades grunden till ett löpande intresse för effektivt minnesutnyttjande vid Manchesteruniversitetet. Detta kom senare att leda till konstruktionen av Atlas, en av föregångsdatorerna både vad gäller multiprogrammering och time-sharing.

1950 - året efter EDVAC och EDSAC - framställdes ytterligare två engelska datorer: The ACE Computer (samarbete mellan National Physical

Laboratory och English Electric Ltd) samt MADM (the Manchester Automatic Digital Machine, vidareutvecklad av Ferranti Ltd), som ytterligare befäste Englands dåvarande framträdande position inom fältet.

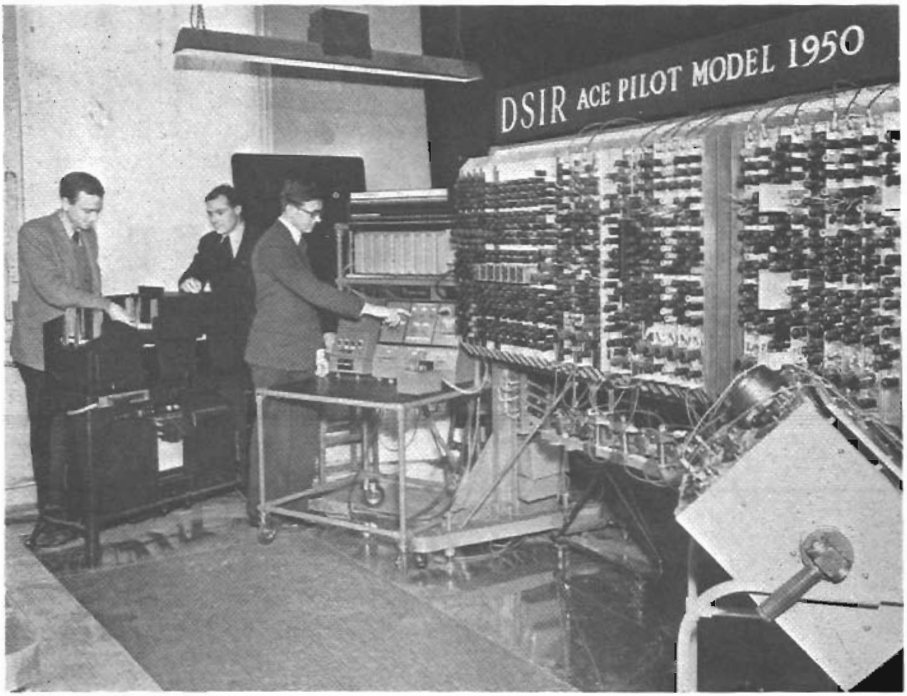


Fig 1.1:2. Den engelskframställda ACE-datorn.
(Ur B.V.Bowden, ed: Faster than Thought. Med tillstånd av: Sir Isaac Pitman and Sons Ltd, London.)

1.2. Datorer utbjuds kommersiellt

Univacstarten

Den första tiden efter det Eckert och Mauchly lämnat Pennsylvania-universitetet och bildat sitt eget bolag ägnade de åt konstruktion av en binär, snabbare ENIAC-version, BINAC. Denna dator var den som först introducerade automatisk självkontroll av alla interna dataflyttningar. 1949-50 påbörjades så arbetet med framställning av en kapabel och för skilda tillämpningar användbar maskin, som gavs namnet UNIVAC (Universal Automatic Computer). Denna maskin, som levererades i sitt första

exemplar 1951, kan betraktas som banbrytande både genom sina kraftfulla interna kontrollmöjligheter och sitt sekundärminnessystem. Maskinen arbetade med ett buffrat magnetbandsystem, varvid band kunde läsas både fram- och baklänges med en hastighet jämförbar med den hos det tidiga 60-talets maskiners. Univac blev troligen den mest flexibla och generellt användbara maskinen under 50-talets första hälft.

Den finansiella uppbackaren av the Eckert-Mauchly Computer Corporation omkom tragiskt vid en flygolycka ca 1951, och företaget hamnade i ekonomiska svårigheter. Det visade sig att man hade underskattat både erforderlig penning- och tidsmängd för produktion av det ambitiösa Univac-systemet. Ett flertal order hade tecknats för blott ca 250 000 dollar per system, och en sådan summa kunde inte ens täcka framställningskostnaderna. Det blev därför förmånligt att acceptera inbjudan att forma Eckert-Mauchly Division inom Remington Rand Corporation. Försäljningspriserna höjdes därvid omgående för nya Univac-system.

Tillsammans med det av Remington Rand inköpta företaget Engineering Research Associates, vilket saluförde de egna maskinerna ERA 1101, 1102 och 1103, bildades småningom Univac Division of Remington Rand, som saluförde både Eckert-Mauchlys maskin, nu kallad UNIVAC 1 och ERA-modellerna, vilka omdöptes till Univac Computers.

Man kan fråga sig varför Remington Rand, vid 50-talets början med hårdvaruprodukter flera år före sina konkurrenter, så pass snabbt därefter föll tillbaka från sin ledande position. Administrationsproblem kan vara en orsak. Man hade inkorporerat två lovande företag från den tidiga utvecklingen, men föreföll aldrig kunna få dem att samarbeta som delar i en större enhet. Det är troligt att bristande försäljningskompetens även spelade in. Den vetenskapligt orienterade personalen från Eckert-Mauchly tenderade att dämpa kundentusiasmen i stället för att backa upp den, genom att sanningsenligt påvisa sin maskins begränsningar när alltför höga förhoppningar från kunderna restes.

Dessutom rådde en inkonsistens på maskinsidan mellan Eckert-Mauchlys periferiutrustning, som arbetade med 80-kolumns hålkort, och Rands 90-kolumnsutrustning. Fram till 1955, då Univacs radskrivare med 600 rader/min lanserades, använde Univac I huvudsakligen den s k Uniprinter, som skrev med skrivmaskinshastighet direkt från magnetband, ett oekonomiskt utnyttjande.

På inmatningssidan introducerade Univac I direkt skrivning på magnetband från skrivmaskin. En tangentbordsopererande bandverifierare blev aldrig fullt utvecklad, varför verifiering och rättning tvingades utföras

Fig 1.2:1. Univac 1.
(Ur T.E.Ivall, "Electronic Computers" med tillstånd av författaren.)

datorprogramstyrt, vilket då innebar låga körtider. Det är troligt att denna teknik kan komma i nytt ljus för 70-talets databehandlingsverksamhet, i samband med kraftfulla processorer och direktanslutna skrivmaskiner till mycket stora direktaccessminnen. Med datorer som Univac I nåddes knappast någon väsentlig framgång i detta avseende.

Den i flera avseenden jämfört med Univac I förbättrade modellen Univac II, försedd bl a med magnetiskt kärnminne, samt programkompatibel med sin föregångare, försenades av organisationsproblem vid sin framställningsprocedur i över två år. Så långt hade utvecklingen redan hunnit vid mitten av 50-talet att denna försening var tillräcklig för att IBM med sin modell 705 och sin erkända försäljningsskicklighet skulle få ett försprång inom datorfältet, som sedan dess ingen konkurrent kunnat hämta in.

IBM träder in

International Business Machines Corporations inmarsch på området kring beräkningsautomater inleddes strax före andra världskriget genom lanserandet av elektromagnetisk utrustning. Detta var som komplement till redan tidigare i företagets produktionsled existerande mekanisk hålkortsutrustning. Som ovan nämnts realiserades samarbete med Aiken vid Harvard University mellan 1939 och 1944 i samband med tillkomsten av MARK I, den enligt uppgift mest voluminösa elektromagnetiska dator som någonsin framställts. Aiken fortsatte under 40-talets senare hälft konstruktionsarbete med färdigställandet av MARK II, en likaledes stor relämaskin, samt MARK III och MARK IV, en-exemplars trumorienterade maskiner. Samtidigt byggde IBM i New York den kraftfulla SSEC (Selective Sequence Electronic Calculator), som emellertid endast delvis var elektronisk, förutom 13 000 rör var den försedd med 23 000 elektromagnetiska reläer. SSEC färdigställdes 1948; den producerades dock aldrig i större serier.

IBM var emellertid redan vid denna tid ett etablerat företag. För hålkortsmaskiner hade man sedan länge en dominerande marknadsposition och den 1948 lanserade modellen 604 Electronic Calculating Punch, som gjorde ökad snabbhet möjlig för hålkortshanteringen, utgjorde en ytterligare förstärkande faktor. Trots att 604-an innehöll över 1 400 radiorör, och därför var tämligen servicekänslig, kunde man sälja tusentals exemplar i den uppvaknande kontorsrationaliseringens tecken utan allvarliga besvär från konkurrenter.

Den från en IBM-kund härstammande idén att sammankoppla en 604 med en redan existerande ren kalkylator, IBM 400, visade sig som ett lycko-

samt steg framåt. IBM lanserade denna tvilling under namnet CPC (Card Programmed Calculator), en maskin som redan 1950 möjliggjorde exekvering av program, bestående av hålkortsinstruktioner, i godtycklig mängd. CPC:s arbetssätt innebar sekvensiell inläsning och direkt utförande av instruktioner, tillhandahållna en per hålkort. Maskinen arbetade sålunda inte i enlighet med minneslagrade användarprogram, den måttliga maximala hastigheten 150 instruktionskort per minut ger dock ett för långsamt intryck, då varje instruktion innebar utförande av en mer eller mindre komplicerad datakorthantering. Förutom kalkyleringar kunde vid varje steg antingen utmatas en tryckt rad eller stansas ett kort.

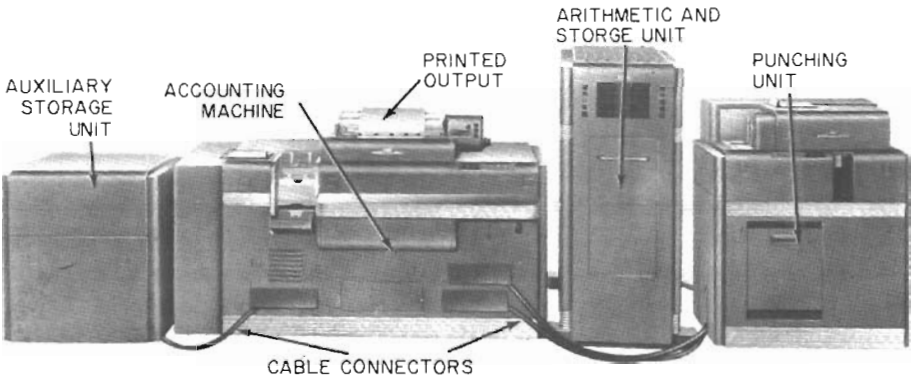


Fig 1.2:2. IBM CPC (card Programmed Calculator).
(Ur F.R.Crawford, "Introduction to Data Processing", Prentice Hall. Med tillstånd av IBM, New York.)

CPC innebar knappast i sig själv något revolutionerande steg framåt i utvecklingen. Dess betydelsefullhet kan snarast sägas ha legat i den databehandlingsmognad som med dess hjälp spriddes ut på fältet. Detta skedde i avvaktan på att maskinerna med minneslagrade program skulle nå upp till givna förhoppningar om leveranser och ökad bearbetningskapacitet.

Trots att IBM sålunda redan hade en ledande ställning vad gäller elektroniskt styrd beräkning i hålkortsmaskiner var man relativt långsam vid inträdet på den kommersiella marknaden kring den uppvaknande icke-hålkortsbundna tekniken. Företaget betraktade Univacs framfart med viss skepsis, anseende att magnetbandstekniken var obeprövad och riskabel. Man rekommenderade sig själv och sina kunder att kvarstå vid hålkortsmetoderna. Magnetbanden bör för IBM vid denna tid ha synts närmast

som ett hot mot hålkorten, för vilka man hade upparbetat en stabil och nära monopolsliknande marknadsposition.

Konsekvenser av krigsaktiviteten

Koreakriget i början på 50-talet kom att innebära ett uppvaknande för hela industrin kring automatiskt beräkningsarbete. Vid denna tid, närmare bestämt från 1953, lanserade IBM sin Defence Calculator, senare kallad IBM 701, en stor vetenskapligt orienterad maskin med 2048 ords inre minne, uppbackat av trumma och (t o m) magnetband. Två år senare dök dess efterföljare 702 upp, avsedd att komplettera sin föregångare för administrativa tillämpningar. Redan före den första 702-leveransen blev det emellertid uppenbart att denna senaste maskin knappast skulle kunna leva upp till förhoppningarna. Det elektrostatiska inre minnet, på 10 000 karaktärer, var inte tillräckligt pålitligt och bearbetningshastigheten var för låg, med 115 mikrosekunders tidsåtgång för exekvering av en standard fem-karaktärers instruktion. Magnetbanden kunde bara läsas framlänges, banden hanterades fullständigt obuffrat och kortläsare och radskrivare arbetade on-line. Det föreföll klart att Univac i många avseenden var ett överlägset system.

En av de viktigaste bidragande faktorerna till IBM:s obestridliga framgång har varit företagets förmåga att reagera snabbt och effektivt i samband med möjligt hotande kriser. Modell 702 utgjorde en sådan krisrisk, och redan före färdigställandet av denna maskin startades intensivt utvecklingsarbete på en efterföljare, modell 705. Så snabbt som ett år efter leverans av den första 702-an kunde modell 705 släppas ut, och samtidigt drogs 702 bort från marknaden. Denna snabba reaktion krävde en väsentlig resursinsats av företaget, men satsningen visade sig riktig. Det kunde dock höras röster som menade att 705, åtminstone i sin tidiga version, ändå för flera tillämpningar var underlägsen Univac I, en maskin som levererats över fyra år tidigare.

Ett snart uppkommande kundtryck för buffrad bandteknik tvingade IBM att utveckla ett speciellt externt sådant tilläggssystem till 705-an. TRC (Tape Record Coordinator) blev namnet på en kontrollenhet (med bl a 1024 karaktärers separat magnetiskt kärnminne), varav flera kunde anslutas till en central modell 705. Härvid blev maskinen visserligen jämförelsevis dyr, men kom därmed att utgöra ett kapabelt system.

I 705 modell II utökades det inre minnet, och modell III innebar dessutom förbättrad inre snabbhet. Inte ens med denna senare modell, lanserad 1958, kunde emellertid band läsas baklänges, varför IBM knappast kunde

tillhandahålla lika effektiv sortering som vissa konkurrerande utrustningar.

Det bör dock konstateras att IBM med sin intensiva försäljningsdrive för 705-an, speciellt med hjälp av Univac II:s i tidigare nämnda förseningar, vid 50-talets slut hade upparbetat en stark marknadsposition. Tiden var nu mogen för inträde av transistortekniken, som kom att öka bearbetningskapaciteterna och -säkerheten väsentligt.

Den svenska stjärnans uppgång

Den redan 1950 färdigställda svenska relämaskinen BARK utgjorde egentligen inget annat än en språngbräda till mera avancerad utrustning. Elektronikens fördelar stod tidigt klara för BARK-konstruktörerna och maskinens elektroniska broder, kallad BESK (Binär Elektronisk Sekvens Kalkylator), kunde startas upp 1953. Denna rent rörorienterade maskin, framställd under ledning av Erik Stemme, kom att bli en föregångare till flera senare nordiska konstruktioner. BESK var under något år den snabbaste maskinen i Europa, och bidrog väsentligt till att skapa anseende för svensk databehandling. Inom kort tid kördes maskinen i tre skift, utförande värdefullt beräkningsarbete för både industri och forskning i vårt land.

Fram till 1956 förbättrades BESK successivt i tekniskt avseende, tillförlitligheten ökades och kringutrustningen förbättrades. Bl a utbyttes Williamsrören i primärminnet mot ferritkärnor. Maskinens kapacitet bevisas inte minst av det faktum att den var i drift ända till 1967, då den med fanfarer förflyttades till Tekniska Muséet i Stockholm, där den numera kan beskådas som ett monument över svensk datorkonstruktions storhetstid.

De vid mitten på 50-talet upprepade påstötningarna från Matematikmaskinnämnden på statsmakterna att få disponera de icke obetydliga inkomsterna från de löpande BESK-körningarna till projekt av utvecklingskaraktär gav inga positiva resultat. Myndigheterna föreföll helt ha tappat intresset för arbetets vidareförande. Det är därför begripligt att Matematikmaskinnämndens kvalificerade personal ställde sig intresserad när AB Åtvidabergs Industrier i början av 1956 startade förhandlingar om personalförflyttningar. Företaget planerade inträde på marknaden som maskinleverantör, och för BESK-konstruktörerna innebar detta en möjlighet att få vidareutveckla sitt arbete från decenniets början.

I september 1956 skedde övergången. Åtvidaberg anställde merparten av Matematikmaskinnämndens hjärntrust, varvid nämnden decimerades på ett avgörande sätt. Statsmakterna bör med sitt dåvarande, och alltjämt

under 60-talet, svaga intresse för en nationell datautveckling få påtaga sig ansvar härför.

Till att börja med blev arbetet hos Åtvidaberg lyckosamt för utvecklingen. En tidig beställning från ASEA om en ny maskin ledde till att en utveckling av BESK, med förbättringar på en rad punkter, såg dagens ljus redan 1957. Den gavs namnet Facit EDB (man använde på den tiden företrädesvis termen Elektronisk databehandling snarare än Automatisk). Denna helt rörbestyckade dator, med ordlängd 40 bitar, additionstid 45 mikrosekunder samt kärn/trumminnesstorleken 2048/8192 ord, kom att tillverkas i nio exemplar, varav något ännu (1969) är i drift. I anknytning till dessa maskiner framtogs successivt av Åtvidaberg avancerad kringutrustning. Här kan vi bl a lägga märke till

a) mycket snabb hålremsutrustning

b) ett intressant sekundärminne av magnetbandstyp, det s k karusellminnet. Varje bandstation arbetar med utbytbara "bandpackar", s k skivor, var och en innehållande 64 utbytbara små bandspolar (diameter 3 cm) med ca 9 m band på varje, arbetande med fast blocklängd. Detta lagringskapabla minne kan ses som ett mellanting mellan sekvens- och direktminne.

c) fr o m ca 1964 snabba yttre kärnminnen, som i moduler om 4096 ord kunde anslutas till huvudmaskinen.

De vetenskapligt orienterade maskinerna

Tillbakablickande på den amerikanska utvecklingen kan vi konstatera att under den senare hälften av 50-talet ett flertal nya beräkningsmaskiner såg dagens ljus. IBM 704, med första leverans 1956, utgjorde en kraftfull prestation, som gav företaget vad som nära kan liknas vid monopol beträffande vetenskapliga maskiner. Maskinen arbetade med inbyggd flytande räkning, tre indexregister, en-adressinstruktion och ett minimum om 4096 st 36-bits ord med 12 mikrosekunders cykeltid. Den enda allvarliga konkurrenten var Remington Rands Univac 1103, som fanns tillgänglig i flera versioner. 1103 var den maskin som lanserade egenskapen programavbrott (el. brytsignal), något som på de senaste 10 åren applicerats på så gott som samtliga datorer. Intressant är även att i 1103 kärnminnet programmässigt kunde direktadresseras vidare ut på en magnetisk trumma. IBM:s massiva försäljningssatsning på sin 700-serie plus sena 1103-leveranser och Univacs mindre goda uppbackning av installerade 1103-or medförde emellertid att storebrodern marknadsmässigt inte kunde hotas.

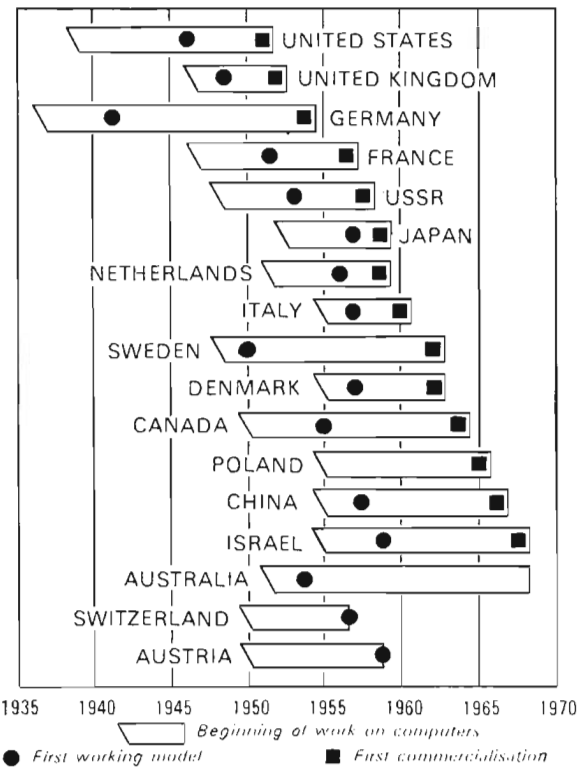


Fig 1.2:3. Tidsplan för den historiska starten för datorindustrin. (Ur OECD Observer nr 40, 1969. Med tillstånd av OECD, Paris.)

De tidiga vetenskapliga maskinerna var konstruerade med tanke på att för beräkningsarbete endast ringa in- och utmatning behövde utföras. IBM 704 var i sin tidiga version försedd med både on-line kortläsare (150 kort per minut) och on-line radskrivare (150 korta eller 75 fulla linjer per minut). Försök gjordes senare att på 704-an installera off-line periferikapabiliteter, som konstruerats för modell 705. Viss inkompatibilitet medförde emellertid problem. Karaktärkoderna var olika, och 704 hade udda medan 705 hade jämn paritetskontroll. Trots detta kom off-linehanteringen till frekvent användning på 704-ans senare versioner. Mera lämplig off-line periferikonvertering dök emellertid hos IBM-produkter inte upp förrän en bit in på 60-talet, med den försäljningsmässigt framgångsrika modellen 1401.

Då in- och utmatning via kort, radskrivare, trumma eller band ej var buffrad, tvingades IBM att vidareutveckla modellerna 701 och 704. Modell 709, levererad från 1958, var försedd med samma 12-mikrosekunders kärnminne som 704, medan instruktionsrepertoaren utökats med bl a indirekt adressering. En nyhet var också det in/ut-system som tillät beräkning att pågå samtidigt som både kort- eller bandläsning och utmatning på band- eller radskrivare. Detta möjliggjordes med hjälp av s k "tidsdelning" av kärnminnet mellan centralenheten och så många som sex datakanaler. Vi kan här se ett av de första stegen tas mot multiprogrammeringsteknik, en driftsfilosofi som av vissa datorsystem tillämpades redan från början av 60-talet, och som 1969 anses nära nog självklar för medelstora datorer och uppåt.

Den rörbestyckade IBM 709 fick kort livslängd enär transistortekniken gjorde sig hörd strax före 1960. 709 efterträddes snart av den transistoriserade 7090, en marknadsmässigt dominerande maskin, som vi skall återkomma till.

Parallellt med IBM:s 709 introducerade Remington Rand modellen Univac 1105, även den med buffrad in/utmatning. Också denna modell kom att komma till korta i samband med transistorernas intåg.

Honeywell och RCA

Ett företag som visade stor tidig aktivitet inom området var Raytheon Corporation. Redan 1948 hade man under utvecklande en kraftfull beräkningsmaskin, först kallad Hurricane och sedermera RAYDAC, med vilken man hoppades få försvarsbeställningar från amerikanska staten. Det försenade och kostnadskrävande framställningsarbetet tillsammans med mindre goda försäljningsprognoser ledde emellertid snart till att man

kom att inse fördelar med ett erbjudet samgående med, och senare uppgående i, det för datorproduktion uppvaknande Minneapolis Honeywell Corporation. Den första gemensamma produkten, DATAMATIC 1000, lanserade bl a ett intressant bandsystem. Maskinen arbetade med tretums band där data registrerades i block av fix längd, arrangerade på så sätt att blockgapet var lika med blocklängden. Vid läsning i den ena riktningen utgjorde blockgapet den databärande arean från läsning i den andra riktningen. Detta kompakta utnyttjande av bandet möjliggjorde lagring av mycket stora datamängder på relativt få bandrullar.

Vid slutet av 1957, då den första DATAMATIC 1000 levererades, hade emellertid IBM två års försprång på marknaden med sin 705, och modell 1000 var jämfört därmed för långsam i förhållande till sitt pris. Försäljningen var så pass dålig under 1958 att maskinen drogs in, och rykten gick att Honeywell skulle lämna datorfältet. I stället beslöt detta företag sig för en ny satsning, i den uppdykande transistoreran, vilket 1960 ledde till den mera framgångsrika modellen Honeywell 800, lanserad med viss multiprogrammeringsmöjlighet.

Även Radio Corporation of America hade varit aktivt inom datorfältet nära nog från början. Sedan andra världskrigets slut hade avancerad forskning bedrivits bl a rörande minnesutveckling, inledningsvis med elektrostatiska minnen, småningom med minnen av magnetisk kärntyp.

RCA:s första större maskin, BIZMAC, kan tillsammans med RAYDAC och DATAMATIC 1000 betraktas som speciellt intressanta bland 50-talets rörmaskiner. BIZMAC arbetade med ett mycket litet kärnminne, uppbackat av en stor magnetisk trumma. Program lagrades på trumman och "bläddrades" in till kärnminnet i fasta block om 32 instruktioner för exekvering. Maskinen annonserades ut som den första sanna variabelordlängds-datorn. På magnetbanden lagrades endast signifikant information. Systemet kunde förses med upp till 200 prisbilliga bandstationer, sålunda mer eller mindre eliminerande tidskrävande bandbyten. Härförutom kunde maskinen förses med speciella sorterare av decentraliserad typ: speciella små dataminipuleringsenheter med inbyggd programmering. Ett telefonliknande system, styrt av två operatörer, skötte kontrollväxling och sammankoppling mellan centralmaskin, bandstationer och sorterarna.

RCA-ingenjörerna ansåg sin produkt så överlägsen att man låste in verksamheten komplett från projektstarten 1952 till 1956, då den första BIZMAC kunde levereras. Denna relativt långa framställningsperiod kom att visa sig ödesdiger, då konkurrenterna under tiden lyckades pressa priserna på större kärnminnen, varför BIZMAC med sin back-up-filosofi blev konkurrenskraftig endast med andra rent trumorienterade maskiner.

De trumorienterade maskinerna

Magnetiska trummor och skivor var bland de allra första enheter som övervägdes för lagring av data- och programmängder. Eckert angav redan 1944 i ett memorandum vid University of Pennsylvania en rekommendation att använda trummor eller skivor för "the general storage of all data required by a computer - not only the numbers being processed, but also instructions ..."

Kring 1949-50 konstruerades, som ovan nämnts, både MARK III och ERA 1101. Båda dessa utgjorde trumorienterade, synnerligen kapabla räkneutrustningar, men på grund av att trummorna var för långsamma för att fungera effektivt som huvudminnen kom ingendera att nå större framgångar.

Magnetiska trummor möjliggjorde förhållandevis snabb åtkomst (5-25 millisek) till relativt stora datamängder, till lågt pris per lagrad bit, i förhållande till elektrostatiska minnen eller kärnminnen. Trots att i början på 50-talet en mångfald dyrare, och väsentligt snabbare, datorer fanns tillgängliga på marknaden, kom maskiner med trumma som huvudminne att nå en viss position, framför allt bland köpare med begränsad tillgänglig budget. Det föreföll förhållandevis enkelt att konstruera en trumorienterad prototyp, att döma av antalet nya modeller som vid denna tid dök upp, men mindre enkelt att bygga upp en adekvat produktionsfacilitet och hålla maskinerna löpande understödda.

En leverantör, Computer Research Corporation, presenterade med en av sina modeller CRC 102D, ett nytt magnetbandsförfarande, som kan vara intressant ur svensk synpunkt. Bandet rullades inte upp på rulle, utan fick fritt falla till bandstationens botten efter läsning/skrivning. Systemet nådde aldrig större framgång, och kan betraktas som en parentes i utvecklingen. Det är dock inte uteslutet att AB Åtvidabergs Industrier lånat en av idéerna till bandhanteringen i sitt från början av 60-talet (till Facit EDB) lanserade karusellminne härifrån.

För en tid hade ett antal mindre firmor, t ex Electronic Computer Corporation, Consolidated Engineering Corporation och Electro-Data Corporation, marknaden för medelstora trumdatorer för sig själva, men 1953 lanserade IBM sin första konkurrent till dessa, modell 650. Denna maskin kunde erbjuda fördelar framför de mindre firmornas utrustningar. 650-trumman (enligt utsago av Univac-konstruktion) roterade med ca 4 ggr så hög hastighet, och dess adresseringssystem, där varje instruktion även innehöll länkadressen till nästa instruktion, var väl lämpat för smartkodning. Hålkortshanteringen var buffrad, men utgjorde enda in/utmöjlighet, vilket begränsade systemets användbarhet jämfört med t ex Electro-Datas flexiblare Datatron.

IBM:s starka position från hålkortstiden kom att visa sig av stort värde för modellen 650. För hundratals datorkonsumenter visade sig denna prisbilliga maskin som det naturliga steget upp från den rena korthanteringen. I stället för ca 50 maskiner som planerat, kunde man utan större besvär få avsättning för över 1000 st.

Under 50-talet utvidgades modellen 650:s kapacitet successivt, med magnetbandshantering, on-line radskrivare, skivminne och mot årtiondets slut även en kraftfullare, 4000-ords, trumma samt 60 ords kärnminne. IBM 650 kom att ge företaget värdefulla erfarenheter och blev en direkt föregångare till modellerna 1620 (för mindre vetenskapliga beräkningar), 1400-serien (för mindre administrativa bearbetningar) samt 7070-serien (för större administrativa rutiner).

Den näst IBM 650 mest sålda trumorienterade datorn var Univac Solid State. Denna var en av de första marknadsmässigt tillhandahållna heltransistoriserade datorerna. Med denna dator bevarade Univac sin marknadsposition under slutet av 50-talet.

Även Burroughs var aktiv på det trumorienterade fältet. Med tradition från 1948 lanserade man under 50-talet flera trumdatorer, från den mindre E 101 till den kraftfulla BEAM IV. BEAM fick emellertid ge vika för den medelstora modellen 220, som hade övertagits från det med företaget inkorporerade Electro-Data. 220 kan sägas ha blivit den sista röroriente-rade maskinen på den amerikanska marknaden.

1.3 Transistorteknikens inmarsch

Uppsnabbade datorversioner

Ända sedan den uppfunnits, 1948, betraktades transistorn som nyckeln till en stark expansion inom datorteknologin. Det krävdes emellertid mera tid än som från början förmodats för att få den att bli en tillräckligt prisbillig och pålitlig produkt. Problemen rörde såväl framställning i större kvantiteter med likformig och enhetlig karaktäristik som finnande av lämpliga utgångsämnen för framställningen.

Halvledarteknikens genombrott uppvisar egentligen en utvecklingslinje som knappast kan sägas vara ny. För nära nog varje teknologiskt lovande uppfinning inträder till att börja med en period av vad som må synas som

stagnation. Detta må vara en tidsrymd för ingenjörsutveckling, eller en erforderlig tid för att konstruera det ekonomiska system som måste till för produktion i större skala. Många lovande idéer och komponenter överlever inte denna tid, utan går under som en följd av praktiska konsekvenser. För en tid såg det hotande ut även för transistorn, men från ca 1955 lättade molnen.

Först på området var Lincoln Laboratories med sin heltransistoriserade maskin TX-O, vilken använde sig av halvledare som utvecklats 1954 av Philco Corporation. Trots vissa problem var det därmed inte längre något tvivel om i vilken riktning maskinvarans elementutveckling skulle gå. Radiorörens dagar var därmed räknade.

Ett flertal företag sökte lyckan genom ett snabbt anammande av den nya tekniken. National Cash Register lade ut riktlinjerna för, och General Electric framställde åt dem, modellen 304, en medelstor administrativt orienterad maskin, som dock på grund av sin enligt uppgift begränsade flexibilitet inte nådde större spridning.

RCA 501 var även den relativt långsam, och dess marknadsframgång berodde snarast på dess tillgång till en av de allra tidigaste Cobol-kompilatorerna. Kompilatorn var långsam, men för många användare var en långsam Cobol bättre än ingen Cobol alls.

IBM:s transistoriserade efterföljare till 650 och 705 var modellen 7070, som släpptes ut något senare än de ovan nämnda, och var klart kraftfullare. Serien 705 avsågs avslutas med 705 modell 3, och kunderna förmodades konvertera alla sina program till den ordorienterade 7070. Som så många gånger senare i utvecklingen visade sig emellertid kompatibiliteten avgörande, IBM mer eller mindre tvingades att lansera 7080, en transistoriserad utveckling av 705, en mindre flexibel, relativt dyr maskin, som garanterade en viss marknad enbart på grund av att den direkt kunde köra 705-program.

Uppmärksamhet kom att riktas mot Honeywell efter annonseringen av modellen 800. Prisklassen låg kring medium, men maskinen lovade, som det föreföll, synnerligen goda prestanda. Konkurrenterna var skeptiska till detta förhållande pris/prestanda, men Honeywell stod fast, och lyckades utan besvär få avsättning för ett icke ringa antal 800-system. Maskinerna var försedda med ett intressant hårdvaruorienterat multiprogrammeringsliknande system, med åtta uppsättningar sekvenserings- och kontrollregister för fördelning av beräknings- och styrkapaciteten.

Burroughs och Philco

Något senare än jämförbara konkurrentutrustningar släppte Burroughs ut sin B 5000. Denna maskin är till sitt arbetssätt influerad av programmeringsspråket Algol. Dess aritmetiska register har bringats att uppträda som om de befann sig på toppen av en s k "pushdown stack", samt övrig hårdvara är väl avpassad för implementering av rekursiva procedurer och dynamisk minnesfördelning (under exekvering). Dessvärre för Burroughs lyckades man mindre väl med att hålla leveransplanerna, och när det dessutom visade sig att maskinen arbetade långsammare än avsett dämpades kundintresset för modellen efter det att den 1963 levererades i sina första exemplar.

En på flera sätt förbättrad 5000, modell 5500, är ännu för närvarande tillgänglig på marknaden, och har nått viss framgång.

Utvecklingen av både transistorn och relativt billiga kärnminnen möjliggjorde för många leverantörer byggandet av mindre maskiner som var förvånansvärt bearbetningskapabla jämfört även med stora tidigare rördatorer. IBM 1400 och 1600-serierna, som kom ut omkring 1960, visade att det gick att få avsättning för modellika datorer i tusental. Även andra leverantörer började vid 60-talets början få klart för sig att en kraftfull marknad existerade för mindre maskiner. Hundratals Univac Solid State, RCA 301 och CDC 160 såldes. Dessutom fanns Burroughs 200, Honeywell 400, General Electric 200, NCR 300, och andra.

På liknande sätt som 60-talets början lanserade de mindre datorerna visar samma decenniums slut en intressant och uppvaknande försäljningspotential för vad som skulle kallas minidatorer, framför allt uppbackade av time-sharing-tekniken.

Philco insåg snabbt transistorernas möjligheter som baselement vid maskinvaruframställning. Företagets kraftfulla TRANSAC S-2000, senare känd som Philco 2000, kunde levereras till årsskiftet 1959-60, uppvisade intressanta egenskaper. Bland annat hade bandsystemet möjlighet att automatiskt sammankoppla alla bandstationer med alla överföringskanaler, vilket inte var möjligt hos jämförbara konkurrentsystem. Philco 2000 avsågs kunna konkurrera med IBM 704 och 709, som då släppts ut. Trots att prestandamässigt så var fallet, var inte uppbackningen tillräckligt tung för försäljningsframgångar i större skala. En avgörande faktor blev även IBM:s snara uppdykande med sin 7090, med sitt 2.18 mikrosekunders minne, jämfört med Philco 2000:s 10 mikrosekunder. Philco tvingades snabbt förbättra konstruktionen, och från och med 1963 kunde man leverera en avsevärt uppsnabbad 2000, modell 212, som möj-

ligen kan anses som mitten av 60-talets maskinvarumässigt kraftfullaste maskin, framgångsrikt konkurrerande med CDC 3600 och IBM 7094 modell II. Minnescykeltiden 2 mikrosekunder kompletterades av mycket snabb aritmetik och en look-ahead-funktion. In- och utmatningssystemet hade även förbättrats sedan de första 2000-leveranserna.

Kring 1963 förändrades Philcos ställning radikalt i och med att företaget köptes upp av Ford Motor Company. Ford bestämde sig för en enbart mycket måttlig satsning inom datorområdet, och trots att den nya Philco 2000 modell 213 utannonserades 1964-65 har Philco i fortsättningen under 60-talet låtit tala om sig enbart sporadiskt.

Control Datas framfart

Historien om CDC:s uppdykande och märkbara framgångar är en av Askunge-sagorna i dataindustrin. En grupp Univac-anställda, innefattande några ursprungligen från ERA, bröt sig ur och bildade 1957 företaget Control Data Corporation. Initiativtagarna hade arbetat med utformning av transistoriserade militära datorer hos Univac, och lyckades på märkligt kort tid få fram sin egen första maskin. Modellen CD 1604 kunde levereras vid årsskiftet 1959-60. Datorn var en binär 48-bits maskin, knappast lika kapabel som Philco 2000 eller IBM 7090, men också väsentligt billigare. Till att börja med tillhandahöll CD inga programvaror alls, utan sålde ren maskinvara till mycket lågt pris till universitetet, som förmodades kunna lösa programvaruproblemen själva.

Företaget noterade framgångar. Modellen 3600, som efterträdde 1604 ca 1963, har varit en konkurrenskraftig maskin. Den kom att inleda CD:s framfart mot en stark position inom fältet större maskiner.

Redan tidigt under framtagningsarbetet med 3600 hade man börjat skissera på en supermaskin, med en helt okonventionell uppbyggnadsfilosofi. Denna dator, modell 6600, blev en banbrytare för Control Data. Med leverans så tidigt som 1964 kunde man demonstrera en för sin tid, och i viss mån fortfarande, osedvanligt stor beräkningskapacitet. Delvis har denna kapacitet samband med maskinens logiska uppbyggnad, den centrala beräkningsenheten är kompletterad med tio perifera med minne programmerbara processorenheter, som decentraliserat utför aritmetiska, logiska och styrande instruktioner. På detta sätt har man bl a lyckats hålla spilltider härrörande från operativsystemets arbete vid en låg nivå. En av de tio perifera processorerna är f ö dedikerad till endast styrfunktioner inom systemet. Enligt CD kan centralenheten utföra i genomsnitt över tre millioner operationer per sekund.

Med 6600 kan CD sägas ha tagit det första steget i utvecklingen mot samarbete mellan små och stora programmerbara datorenheter. Vi kan vänta en intressant utveckling på detta område under 70-talet.

Trots att Control Data visade sig överinvestera något i 6000-projektet, och som följd härav kom i tillfälliga likviditetssvårigheter kring 1966, har man kunnat visa en imponerande expansion under företagets hittills 12-åriga existens. Kring 1967 hade ett stort antal av västvärldens ledande forskningsorienterade större beräkningscentraler antingen en 6600 installerad eller en på order. Denna framgång har medfört att man från CD:s sida avsiktligt hållit inne med frisläppandet av sin nästkommande maskin, CD 6800, sedermera efter inre förändringar kallad CD 7600.

Koncentrationssträvanden i England och Frankrike

Firman ICT (International Computers and Tabulators Ltd) tillkom 1959 genom samgående mellan British Tabulating Machines och Powers-Samas. ICT visade stor villighet att expandera. 1961 övertogs väsentliga delar av General Electrics brittiska elektronikintresse, 1962 införlivades Electric & Musical Industries Ltd's dotterföretag EMI Electronics Ltd, och 1963 vidare Ferranti's datoravdelning.

Samma år, 1963, realiserades en annan fusion i den engelska datorbranschen, i och med att Leo Computers Ltd gick samman med English Electric's dataavdelning, under namnet English Electric - Leo Computers Ltd. 1964 anslöts Marconi-sällskapet, och 1967 även Elliot Automation, varvid namnet ändrades till English Electric Computers Ltd, EEC.

Dessa integrationssträvanden nådde sin kulmen 1968 då EEC och ICT tillsammans med expertis från både The Plessey Company och brittiska teknologministeriet formade International Computers Ltd, ICL. Brittiska staten har 10 % intresse i ICL. Med ca 35 000 anställda och en årsomsättning kring 1.5 miljarder kronor är ICL idag världens största icke-amerikanskt kontrollerade datorkoncern. Den mot slutet av 60-talet saluförda ICL 1900-datorserien, varav ett flertal modeller i olika storleksklasser existerar, har sålts i stora upplagor. Serien omfattar såväl små, modell 1901, som medelstora, 1904, och mycket stora datorer. ICL synes i Sverige lägga sig vinn om låg prissättning och har med hjälp av bl a goda standardmässiga programvaror skapat sig en kraftfull marknadsposition i Europa och inte minst i Skandinavien. ICL's frammarsch har ägt rum under inflytande av substansiella nationella stödåtgärder från engelska regeringen.

I Frankrike togs datorutvecklingens tråd upp av Compagnie des Machines Bull. Denna firma hade startats redan 1931, huvudsakligen förlitande sig på ett antal utifrån inköpta patent. Dessa patent härrörde från den norske ingenjören Fredrik Bull's idéer rörande hålkortsutrustning. Under 50-talet samt början av 60-talet producerades såväl hålkortsmaskiner som datorer. Bull GAMMA-serien innefattade både mindre (modellerna 3 samt 10) och större modeller (modell 60). Trots datorernas goda beräkningskapacitet lyckades firman inte upparbeta en tillräckligt stabil marknadsposition, bl a sannolikt på grund av otillräcklig satsning på marknadsföring, och accepterade 1964 ett halvt uppgående i den amerikanska General Electric-koncernen. I och med att GE snart därefter även övertog betydande intressen i det italienska Olivetti, skapades ytterligare en inkörsport för amerikansk datoraktivitet i Europa.

Dessa förhållanden tilltalade knappast general de Gaulle, som konsekvent önskat hävda Frankrikes egen marknadsställning bl a på detta område. De företag, som tillverkar datorutrustning, och som 1967 ännu var i franska händer, sammanslogs därför detta år till ett gemensamt företag, Compagnie Internationale d'Informatique (CII), med uppgift att bringa till stånd och samordna forskning, utveckling och produktion av franska datorer. Detta skedde under inflytande av arbetet med den s k Plan Calcul, en koordinerad plan för expansion bl a för den franska industrin samt rationalisering av förvaltningen. Fram till 1969 har två datormodeller lanserats av CII. Den tidigaste, IRIS 50, kan betraktas som en maskin av medelstorlek (ungefärligt motsvarande IBM 360/50), huvudsakligen inriktad mot satsvis bearbetning. Modellen IRIS 80, som avses levereras i sitt första exemplar 1971, anges i hög grad avsedd för multipel access-bearbetningar. Intressant att notera är att CII som "introduktionsåtgärd" påtagit sig att licenstillverka det amerikanska Scientific Data Systems' dator Sigma 7. Det pågår sålunda 1969 licensproduktion av amerikanska datorer i företaget CII, vilket som huvudsakligt mål har att främja utveckling av franska datorer.

Såväl England och Frankrike som Tyskland tillämpar f n kraftfulla statliga stödåtgärder till den inhemska datorindustrin. Bland ett flertal åtgärder i denna anda kan nämnas att brittiska statliga upphandlingsinstanser vid datorsystemanskaffningar har tillstånd bortse från prisskillnader upp till 25 % till förmån för inhemska tillverkare.

Strypningen av det svenska statliga datorstödet: Industrin tar över

Ca 1959 fattade Åtvidabergs Industriers ledning beslutet att lägga ned projektet kring vidareutveckling av Facit-maskinen. Detta beslut, ur

vissa synpunkter beklagligt, kan ha haft flera orsaker. Den uppdykande transistortekniken föreföll tvinga till stora investeringar, amerikanska jättesatsningar inom området kunde dämpa entusiasmen för svenska konstruktioner, och Facit-maskinens kapacitet och utbyggnadsmöjlighet föreföll för låg för att kunna mäta sig med de uppdykande utländska stordatorerna. Tiden var ännu inte mogen att klargöra marknadsvärdet för små maskiner. Åtvidaberg har inom databehandlingsområdet, under 60-talet endast tillverkat speciella maskiner för militära ändamål, parallellt som man fortsatt utvecklingen av sin kringutrustning till Facit EDB. Ett steg som kan leda till en viss återaktivering togs dock 1969 i och med lansering av Facits bildskärmar.

SAAB:s verksamhet förtjänar beröras. Detta företags militärt orienterade kontrakt har kunnat bära en del av utvecklingskostnaderna för produktion för den civila marknaden, och företaget har varit aktivt under hela 60-talet. Den första maskinen, SARA (SAAB:s Räkne-Automat), en broder till BESK, dök upp i mitten på 50-talet.

För SARA´s konstruktion svarade Börje Langefors (f n professor i Informationsbehandling vid Tekniska Högskolan/Stockholms Universitet). I några avseenden avvek dock SARA från BESK. Den försågs redan från början med ferritkärnminne och dess adresseringsstruktur ändrades något för att medge enkel kommunikation med yttre enheter. Dessutom utvecklades en teknik som möjliggjorde användning av magnetband som sekundärminne. Bandstationerna inköptes från Ampex, men systemet för bandhantering utvecklades helt av SAAB. Fast blocklängd användes, varigenom bl a skrivning inne i ett band möjliggjordes. Under sökning efter ett önskat block var centralenheten helt obelastad av sökprocessen, varvid sålunda centralenheten ostört kunde fortsätta internbearbetningen.

Ett intressant faktum rörande SARA´s magnetbandssystem må nämnas. Data lagrades på bandet med hjälp av självkorrigerande kod (Hammingkod). Härvid var 4 bitar informationsbärande och 5 bitar redundanta (för kontrolländamål). Med hjälp av denna vid denna tid nya teknik uppnåddes vid nominell, av Ampex angiven, bandhastighet så låg felfrekvens att bandhastigheten i st av SAAB fördubblades. Härmed ökades felfrekvensen till en nivå som SAAB ansåg lämpligare. Man betraktade en alltför hög säkerhet olämplig därför att användarna då, när fel vid enstaka tillfällen trots allt inträffade, skulle vara alltför otränade i felsökningsarbete, och stå mer eller mindre handfallna. Trots denna avsiktliga ökning av felfrekvensen kunde tillräckligt god pålitlighet uppvisas, vilket bl a lär ha förvånat Ampex: deras rekommenderade bandhastighet hade ju kunnat fördubblas av SAAB.

Icke långt efter 50-talets slut kunde man lansera D21 (som haft interna föregångare, bl a modellen D2), en maskin som i flera fundamentala avseenden avviker från Facit EDB. Maskinen är transistoriserad, med ordlängden 24 bitar och additionstiden 9.6 mikrosekunder. D21 arbetar med sekundärminnen av konventionell magnetbandstyp, även om man senare har möjliggjort anslutning av Facits karusellminnen.

Med D21 nådde SAAB framgångar. Man kunde sälja inte bara inom Norden, utan även till östeuropeiska länder, som av olika, delvis politiska, skäl inte låg lika öppna för amerikansk expansion. D21 visade sig dessutom pris- och prestandamässigt kunna mäta sig med utländsk utrustning. Mest omtalad torde i Sverige vara den s k länsmaskinaffären, där till att börja med kontrakten för installation vid länskontoren delades i stort sett lika mellan IBM 1401 (senare bytt mot IBM 360/30) och SAAB D21. Efter ett års drift visade det sig att D21 gjort klart bättre ifrån sig, varför IBM-utrustningarna ersattes med D21. Denna historia är desto intressantare som den inträffade mellan 1965 och 1968, en tid som annars, vad gäller statliga maskinaffärer, dominerades av IBM-kontrakt.

Med den första installationen 1968 presenterade SAAB så D22, en "medelstor till stor" efterföljare till D21. Denna modell torde ha förutsättningar att visa sig som en lika god satsning som sin föregångare. Med denna maskin har SAAB tagit steget fullt ut från hålremsa till hålkort som huvudsakligt data- och programmedium. Utmärkande för samtliga svenska maskiner från BESK till och med D21 har i detta avseende annars varit hålremsorienteringen. Detta har troligen samband med existensen av Åtvidabergs snabba och säkra remsutrustning.

1969 lanserar Data SAAB en serie mycket prisbilliga s k minidatorer. Denna serie, kallad D5, består till att börja med av tre modeller, D5/10, D5/20 och D5/30. Datorerna har avsiktligt mycket begränsad lagringskapacitet, medan den interna bearbetningshastigheten är hög. D5/10 har den fixa primärminnesstorleken 4 K 4-bitsord, med cykeltiden 1.6 mikrosekunder. D5/20 och D5/30 är utbyggbara till 32 K, med ordlängderna 8 resp 16 bitar. Modell /20 har samma cykeltid som modell /10, medan modell /30 är något snabbare, 1.1 mikrosekunder. De två "mindre" modellerna är avsedda för tillämpningar av relativt låg flexibilitet, medan modell /30 har så mycket som 50 skilda instruktioner, programavbrott, intervallklocka m m.

D5-serien, den första miniserien i Norden, är maskinvarumässigt konstruerad med s k tjockfilmsteknik, varvid komponentintegrationen drivits långt. Inte bara ledningar utan även resistanser, induktanser etc "gjuts" härvid på små plattor (1 x 1 tum). Datorerna är mycket små till formatet, vilket

bl a varit en förutsättning med hänsyn till deras planerade användningsområde. Sålunda kan D5/10 och /20 användas bl a för styrning av bankterminaler. En stor order rörande sammankoppling av D5 med Facits 1969 lanserade bildskärmar tecknades 1969 av ett bankkonsortium i Norden. - Framtida utveckling rörande D5 lovar att bli intressant.

Förutom för SARA och Facit EDB blev BESK mönsterbildande för SMIL (Siffer-Maskinen i Lund), i drift ännu 1969 (då den i Lund kompletteras av Univac 1108), samt för DASK (DAnmarks beSK, eller Dansk Aritmetisk Sekvens Kalkylator) i Köpenhamn.

Före Matematikmaskinnämndens upphörande 1963, varvid ansvar för statlig datorsamordning överflyttades till Statskontoret, hade planer presenterats på en avancerad maskin, den s k Superbesk. Superbesk avsågs bli en transistoriserad och maskinvarumässigt högt utvecklad dator. Bl a hade man parallellarbetande mikroprogrammerade aritmetikenheter i tankarna, såväl som via kort utbytbara maskininstruktioner. Dessa planer kunde emellertid inte erhålla centralt stöd, och någon produktion kom aldrig till stånd. Vissa av de idéer som utvecklades i samband med planerna på Superbesk kom senare till användning för den av AB Data System konstruerade TRASK. Denna dator, som hittills blott framställts i två exemplar, är en transistoriserad och även i andra avseenden moderniserad version av BESK. Med visst beklagande må denna begränsade produktion noteras, enär TRASK för flera tillämpningar bör kunnat visa sig som både effektiv och prisbillig.

Omkring 1957 gjorde Axel Wennergren ett försök att starta ytterligare ett svenskt datortillverkningsföretag. Ett antal exemplar av den amerikanska trumorienterade datorn ALWAC III E fraktades över, och stod som förebild till Wegematic 1000, av vilken typ ett 10-tal exemplar framställdes på AB Nyman-bolagens fabriker i Bollmora. Bl a ekonomiska begränsningar satte emellertid stopp för den utvecklingen. ALWAC-datorer installerades bl a vid Chalmers Tekniska Högskola i Göteborg, Tekniska Högskolan i Stockholm samt vid Uppsala Universitet, och bidrog därmed till att datortänkandet utvecklades vid dessa lärosäten.

Den svenska potentialen

Den kortfattade beskrivning av den svenska datorutvecklingen, som här kunnat ges, må avslutas med några ord om en i mångas ögon förlorad potential vad gäller vårt lands position. Det är ingen tvekan om att vi kring mitten av 50-talet med BESK och planer på Superbesk befann oss väl i takt med den internationella utvecklingen. Den från statsmakternas

sida successiva dämpningen av intresset kring en aktiv nationell datapolitik under slutet av 50-talet och hela 60-talet har verkat hämmande på våra möjligheter att aktivt taga del i den fortsatta expansionen utomlands. Det vore överord att mena att vi med egna datorprodukter helt skulle kunna klara oss själva, en dominerande del av vår datorkonsumtion måste komma som import utifrån. Icke desto mindre skulle bl a vårt uppträdande som kompetenta köpare ha kunnat befrämjas av en mera central satsning på området. Samarbete mellan våra universitet och centralt stödd egen dataindustri skulle kunna ha burit rik frukt. Ansvarig statlig instans under 60-talet har visat liten förståelse för synpunkter från universitetshåll. Vid nyanskaffningar har IBM-produkter dominerat. 1968 hade IBM en värdemässig andel om 72 % av våra statliga datorer. I vissa fall, främst rörande läns- och universitetsmaskinerna har agerande enligt myndigheternas linjer visat sig vara mindre ekonomiskt och måhända även ur svensk synvinkel utvecklingshämmande. Organisationer som kunnat uttrycka formell kritik mot nämnd datorpolitik har emellertid successvit avskaffats.

En av faktorerna som talar för en trots allt ökad svensk datamognad för 70-talet är den industriella satsning som kring decennieskiftet kan märkas. Icke blott SAAB utan även Facit och L M Ericsson och andra företag ökar satsningen på datormarknaden, och deras produktionskapacitet och försäljningsorganisationer kan förmodas utöva inflytande. Hopp finns även av andra skäl om en positiv svensk maskinvaruutveckling under 70-talet. Ett visst uppvaknande bland styrande politiker för dessa frågor har bl a successivt kunnat noteras.

(Ur Svenska Dagbladet. Med tillstånd av Staffan Lindén och Svenska Dagbladet.)

Utvecklingen i Danmark

De gynnsamma erfarenheterna från BARK och BESK i Sverige inspirerade till egna aktiviteter även i Danmark. B. Scharøe Petersen inledde 1954,

efter studier av BESK i Stockholm, konstruktionsarbetet på DASK (Dansk Aritmetisk Sekvens Kalkylator). Detta kunde ske med hjälp av penningmedel från Försvarets Forskningsråd och Marshall-fonden. DASK blev nära en kopia av BESK, dock med skillnaden att den redan från början byggdes med ferritkärnminne samt försågs med tre indexregister. Indexregistren kunde dock kopplas ifrån, i de fall då man önskade fullständig kompatibilitet med SAAB's SARA, vilka två maskiner backade upp varandra.

DASK stod klar 1958. Datorn kunde utföra ca 18 000 räkneoperationer per sekund, och matades med program och data via hålremsa (200-400 tecken/sek). Det sedecimala talsystemet, som BESK tvingade användarna att lära sig för att t ex kunna läsa minnesutskrifter, användes aldrig på DASK på samma sätt. I stället skedde konvertering till decimala tal före tryckning. Denna och flera andra användarvänliga åtgärder föreslogs av astronomen Peter Naur (f n professor i datalogi vid Köpenhamns Universitet), som hade tidig datorerfarenhet. Redan 1951 hade han beräknat planetbanor på EDSAC i Cambridge.

Den danska DASK förvisades först 1967 till museum (Danmarks Tekniska muséum i Helsingør). Dessförinnan hade den utgjort erfarenhetsbakgrund för aktiebolaget Regnecentralens vidareutvecklingar. GIER-datorn, en transistoriserad nykonstruktion, gavs sitt namn på grund av att det första exemplaret beställdes av Geodætisk Institut i Köpenhamn. Denna dator, vars cykeltid är 6,6 mikrosekunder och till vilken bl a synnerligen snabb hålremsutrustning småningom tillhandahölls (läsning med 2 000 tecken/sek), såldes i så mycket som ca 50 exemplar över hela Europa. Även GIER påminner om BESK/DASK i flera avseenden: ordlängden 40 bitar, det begränsade primärminnet (här 1 024 ord), hålremsorienteringen m m. Trots primärminnets litenhet är Algol 60 implementerat till nära nog fullständighet. Endast begreppet own array tillåts ej (detta begrepp kräver i viss implementeringsform tämligen stort primärminne och anses av många som en tvivelaktig teoretisk konstruktion); sålunda är GIER Algol sällsynt komplett.

GIER ersattes 1968 av RC 4000, varmed Regnecentralen tagit steget till integrerade kretsar som maskinvarumässiga byggelement. Denna dator är internt snabbare, additionstid 3 mikrosekunder, har 24-bitsord i ett primärminne som är utbyggbart till 128 K ord, och kan behandla upp till 7 användarprogram samtidigt. Till datorn kan en vid mängd perifera enheter anslutas, bl a GIER's snabba hålremsläsare (RC 2000).

Med det internationella intresset för GIER som bakgrund har även RC 4000 avsetts för utlandsmarknaden. Ett exempel härpå är att Fortran IV tillhandahålls, för de icke Algol-frälsta. RC 4000, liksom Data SAAB D22, visar att även små länder kan konkurrera på den internationella datormarknaden.

1.4 Stordatortiden

IBM:s 7000-serie - banbrytaren

I början av 1958 utgick en uppmaning från the Ballistic Missile Early Warning System (BMEWS) till datorleverantörerna att inkomma med anbud rörande större datorer. Ordern skulle lyda på ett flertal maskiner, och man lät förstå att endast transistoriserad utrustning kunde komma ifråga. Som vanligt vid dylika större affärer var tiden mycket knapp, och böterna för försenad leverans skulle bli höga. IBM lyckades vinna detta viktiga kontrakt genom att erbjuda nära omedelbar leverans av rörmaskinen 709 för att möjliggöra snabbt påbörjad programmering. Man påtog sig dessutom att inom drygt ett år ställa upp en fullständigt transistoriserad, logiskt kompatibel maskin, modell 709 TX, vars kapacitet skulle bli fem gånger högre.

För en kortare tid höll IBM tillbaka erbjudanden om 709 TX till andra kunder. Man var medvetna om att den uppenbarligen snabbt skulle konkurrera ut 709-an, som relativt nyligen börjat levereras. Konkurrenttrycket från Philco 2000 och CDC 1604 manade emellertid fram snart agerande. Man utannonserade efter en tid sin 709 TX, ny kallad 7090, och möttes därvid av ett starkt intresse och höga förväntningar.

De två första 7090-datorerna levererades helt schemamässigt i november 1959. IBM hade emellertid inte helt klarat den hart när omöjliga uppgiften att få modellen fullständigt funktionsduglig på denna korta tid, och antalet ingenjörer som cirkulerade på installationsorten Greenland varierade till att börja med mellan 20 och 200. Kommersiella leveranser startade snart därefter, men reaktionerna var blandade, maskinerna föreföll inte leva upp till förväntningarna. Det nya kärnminnet, planerat till 2.4 mikrosekunders cykeltid, visade sig än snabbare, 2.18 mikrosekunder, men tillförlitligheten var till att börja med låg. Efter det att man lyckats förbättra tillförlitligheten ändrade sig situationen. IBM 7090 blev successivt en pålitlig maskin, och dess höga beräkningskapacitet medförde småningom leveranser i hundratal. Då en typisk 7090-konfiguration låg i köpeprisklassen tre miljoner dollar är det enkelt att förstå vilken ekonomisk framgång maskinen kom att innebära för sitt upphovsföretag.

Ett flertal 7090-maskiner konverterades så småningom till den något snabbare modellen 7094, med inbyggd dubbel precision och ytterligare fyra indexregister. Den sista modellen i 7090-serien, 7094 modell II, som lanserades kring 1963, tillhandahöll ännu snabbare aritmetik, och ett interfolierat (interleaved) kärnminne. Denna modell har, tillsammans

med ursprungliga 7090, ännu mot slutet av 60-talet innehaft en normerande position vad gäller beräkningskapacitet för större datorer.

Under tiden, 1962-63, introducerade IBM bl a de populära 7040 och 7044, som uppvisade något lägre internhastighet än 7090, men till klart lägre pris. De var mer orienterade mot administrativa databehandlingsproblem än den "vetenskapliga" 7090. En kombination av en 7094 med en 7040 eller 7044, med en speciell kanal för att sammanbinda de båda maskinernas minnen, lanserades. Den mindre datorn arbetade som in/utmatningsprocessor och -övervakare, begränsande 7094 till den rena exekveringen av de inmatade och inplanerade programmen. En liknande princip visade sig även användbar i samband med en 1401 kopplad till en 7090, vilken kombination kunde säljas i icke ringa omfattning.

Med 7000-serien kom IBM att befästa sin ledarposition inom fältet på ett avgörande sätt. Det var rimligtvis också nödvändigt med en maskinell fullträff vid denna tid, konkurrenterna blev allt fler och intensivare under 60-talets inledande del. Hade IBM inte kunnat tillhandahålla en lika kraftfull maskinserie som 7000-serien, kompletterad med bl a 1401, vid denna tid hade förmodligen hela 360-systemet med sitt relativt sett troligen sämre prestandauppträdande kommit att behandlas betydligt svalare av konsumenterna.

Under 60-talets första år gick rykten om en helt ny transistoriserad maskinserie, IBM 8000. Åtminstone en prototyp byggdes, men IBM beslöt sig i stället för en satsning på den nya teknologin kring mikrokomponenter, s k integrerade kretselement. Resultatet blev System/360.

De stora Univac-datorerna

Univac's första 60-talsprodukt i medelprisklassen, Univac III, nådde trots programvara för multiprogrammering aldrig större spridning, möjligen på grund av att systemet saknade möjlighet att ansluta stora direktminnen. Företagets svaga satsning på marknadsföring av sina produkter kan även ha varit en bidragande orsak.

Den transistoriserade fortsättningen på 1105 (se tidigare avsnitt), modellen 1107, kom att nå spridning, och visade sig som en flexibel och kapabel maskin. Med första leverans 1962 kunde denna dator uppvisa, förutom konventionellt kärnminne med effektiv cykeltid 2 mikrosekunder, ett tunnfilmsminne om 128 adresserbara register och med cykeltiden 670 nanosekunder. Andra egenskaper som 1107 var bland de första datorerna att lansera var "flersyftesregister", minnesskydd och ett omfattande brytsignalsystem för multiprogrammering.

Trots att Univac 1107 jämfört med IBM 7090 inte har sålts i större upplagor har maskinen kunnat upprätthålla Univac's namn på marknaden. Det är inte minst företagets satsning på operativsystem i början av 60-talet som varit orsak härtill.

1107 har spelat en viktig roll som modell för sin yngre broder, Univac 1108, en med 1107 helt kompatibel maskin (internt ca 5 ggr snabbare) som försålts i icke ringa antal exemplar. 1969 presenterades modellen 1106, en prisbilligare och internt långsammare version av 1108. Kompatibiliteten mellan dessa sena 1100-datorer har upprätthållits. Univac har, tillsammans med huvudsakligen Control Data, kunnat tillskansa sig en stabil position inom marknadsområdet kring de stora datorerna.

Det vore missvisande att i korta ord söka sammanfatta Univac's 60-talsutveckling utan att nämna något om företagets datorer för reelltidsapplikationer. Med modellen 490, levererad 1962 och byggd på erfarenheter av militära system (bl a NTDS - Naval Tactical Data System), lades grunden för en stabil ställning på marknaden kring "civila" reelltidsdatorer.

490-systemets flexibla in/utmatningssystem, framför allt för telekommunikation, tillämpade principer som ärvdes av efterföljarna Univac 418 och 494.

Bland övriga Univac-produkter kan den lilla externprogrammerbara 1104 omnämnas - en tidig kortdator med läsare, radskrivare, stans och även möjlighet till anslutning av magnetband. Vid många installationer kom 1004 att utgöra in/utmatningssystem till större centrala datorer.

Superdatorer: NORC, LARC och STRETCH

Praktiskt taget vid varje given tidpunkt hittills i datorhistorien har industrin haft reella möjligheter att konstruera datorer med klart större kraftfullhet än de allmänt kommersiellt levererade. Det har endast varit fråga om finansiering, ett dock icke litet problem här som annars.

Ett tidigt experiment av denna typ rörde IBM-projektet kring framställningen av NORC (Naval Ordnance Research Calculator) åt US Naval Weapons Laboratory. NORC-projektet påbörjades så tidigt som 1951, och maskinen accepterades 1955. Den planerades kunna utföra 15 000 tre-adressinstruktioner per sekund. Flytande addition resp multiplikation tog endast 15 resp 31 mikrosekunder. Dessa tider är desto märkligare som NORC arbetade med binärkodat decimal representation. Den

snabba multiplikationen erhölls med den brutala hjälpen av nio separata register för lagring av produkten av multiplikanden med var och en av multiplikatorns nio från noll skilda decimala siffror.

NORC´s ursprungsminne var ett 2 000-ordigt Williamsrörminne, som först 1960 byttes ut mot kärnminne. Ännu 1966 var maskinen i drift.

Maskinen framställdes enbart i ett exemplar. Med modellen 704 på väg vägrade IBM att ta flera NORC-order för vilket intresse emellertid fanns.

Redan 1956, då det blev uppenbart att transistorer kunde komma till användning i större sammanhang, började datorindustrin satsa på forskning rörande framställning av stora transistoriserade maskiner. De stora penningbeloppen kom från USA-regeringen i samband med två projekt åt Atomic Energy Commission (AEC). Kontrakt skrevs med Remington Rand Univac om framställning av LARC (Livermore Atomic Research Computer) och med IBM angående en maskin som ursprungligen kallades STRETCH, men sedermera, under den tid IBM avsåg den för kommersiellt saluförande, fick beteckningen IBM 7030.

I december 1956 presenterades två uppsatser vid Eastern Joint Computer Conference i New York, det ena av J. P. Eckert vid Univac och det andra av S. W. Dunwell från IBM, beskrivande planerna för LARC och STRETCH. Båda talade om bearbetningshastigheter av storleksordningen 100 gånger vad de leveransklara Univac 1103A resp IBM 704 kunde uppvisa. Tidsplaneringen och den tidsmässigt simultana satsningen från de båda företagen ger här intryck av en konkurrenssituation. I viss utsträckning är detta missledande på grund av projektens skilda utgångspunkter. LARC byggdes med utprövade elektroniska komponenter, för att garantera mot obehagliga överraskningar under framställningen, medan STRETCH´s tidiga ritningar talade om maskinvaruelement, som i vissa fall fortfarande var på forskningsstadium. Vid jämförelse mellan de två projekten bör vi också hålla i minnet att LARC var över ett år tidigare, både skissmässigt och leveransmässigt.

LARC använde sig av samma representationsteknik som tidigare NORC, den binärkodat decimala, en metod som nära nog samtliga andra stora vetenskapligt orienterade datorer undvikit. Dessutom innefattade maskinen en separat input/output-processor, som tillsammans med antingen en eller två beräkningsenheter arbetade helt parallellt på 4-mikrosekundersminnet. In/ut-processorn innefattade eget minne och var programmerbar, en princip som fått många senare efterföljare i datorutvecklingen. Den höga flexibilitet som denna in/ut-konstruktion medgav, hade en mindre lyckad sidoeffekt: snart sagt vart användarprograms uppträdande i

maskinen kunde allvarligt påverkas av eventuell dålig effektivitet hos in/ut-processorns systemprogram. LARC's maskinvarukonstruktörer trodde möjligen något för mycket på förmågan hos tillgängliga systemprogrammerare att producera optimal in/ut-kod till detta mycket komplicerade hårdvarusystem. - (Denna bedömning har kunnat märkas återuppstå för 60-talets komplicerade maskinera operativsystem, man litar från maskinvarusidan alltför mycket på operativsystemkonstruktörernas skicklighet.)

Den första LARC installerades 1960, och snart därefter ytterligare en, men planerna på att saluföra maskinen kommersiellt i större skala fick skrinläggas då endast ett fåtal order kunde anskaffas.

STRETCH kunde köras igång i sitt första exemplar 1961. I de ursprungliga planerna hade angivits två separata processorer, en karaktär-orienterad och en binär, men de två kombinerades i den levererade maskinen. Dessutom hade det talats om 500 nanosekunders minne, tillgängligt i moduler om 2 048 ord, men det levererade minnet hade fyra gånger så hög cykeltid. En av de mest intressanta egenskaperna hos datorn var look-ahead-funktionen. Denna princip innebär upplockande, avkodning och beräkning av effektiva adresser flera instruktioner i förväg. En look-ahead-funktion, arbetande på ett interfolierat minne, kan tillhandahålla exekveringsklara instruktioner till en (eller flera) beräkningsenhet(er) med mycket högre hastighet än som är möjligt med ett rent sekvensiellt system. En av egenskaperna med en sådan enhet kan sägas vara att bringa en mycket snabb processor på ett relativt långsamt minne att uppträda som om den arbetade på ett mycket snabbare minne. Look-ahead leder emellertid till logiska problem för konstruktionerna, av typen

a) en redan avkodad instruktion visar sig ha modifierats av en instruktion strax före

b) vid villkorliga hopp tvingas look-ahead-enheten arbeta i flera riktningar 'samtidigt', eller inte alls

c) avbrott skall exekveras.

Av flera skäl misslyckades STRETCH med att uppnå den angivna 100 gånger 704-kapaciteten. För ett flertal tillämpningar visade den sig förvånansvärt långsam. Det var svårt att implementera ett effektivt multiprogrammeringssystem på den, och bortsett från fåtaliga mycket stora program var multiprogrammering nödvändig för att riktigt utnyttja maskinens stora kapacitet. Look-ahead-funktionen ledde till större problem än väntat, och skivminnets överföringshastighet fick minskas till hälften för att ge bättre pålitlighet åt datatransporterna.

Med order på omkring 15 system tvingades IBM i maj 1961 utannonsera att maskinen inte skulle kunna hålla specifikationerna, och som en följd därav skulle prisreduceringar genomföras för redan tecknade order. Emedan det lägre priset knappast gav IBM adekvat projektvinst, drogs 7030 STRETCH in från vidare marknadsuppträdande. Endast sju system färdigställdes.

Både LARC och STRETCH kan ses som misslyckanden, emedan både UNIVAC och IBM hoppades på stabil kommersiell marknadsföring av produkterna. Ändå innebar de en utomordentlig stimulans för datorindustrin under början av 60-talet. Utan projektet STRETCH hade IBM mycket väl kunnat bli två år senare med utvecklingen av 7090, den mest framgångsrika stordator något företag marknadsfört. Dessutom kan sägas att det är möjligt att STRETCH hade kunnat nå större kommersiell framgång om inte Philco, Control Data och andra tvingat IBM till jättesatsningen att få fram den interna konkurrenten 7090 i tid.

1.5 IBM befäster sin marknadsmässiga dominans

System/360

Linjen från konventionell hålkortsutrustning till framgångarna med bl a modellerna 705, 1401 och 7090, tillsammans med en oerhörd satsning på ren försäljningsverksamhet, garanterade egentligen IBM framgång vad man än avsåg lansera därefter. System/360 annonserades ut den 7 april 1964. Inte mindre än sex maskiner presenterades, modellerna 30, 40, 50, 60, 62 och 70. Vid denna tidpunkt var dessa datorer ritbordsprodukter. De avsåg, tillsammans med några senare kompletteringar, kunna ersätta samtlig tidigare IBM-utrustning av datorkaraktär. De sades erbjuda högre kapacitet till lägre pris. De var icke kompatibla med föregående maskiner. IBM lanserade i stället begreppet 'emulering', för att beskriva en simuleringsteknik med hjälp av mikroprogrammerade rutiner. Med emulatorer möjliggjordes körning av 1400-program på 360 modell 30 och 7000-program på de större (mikroprogrammerade) modellerna. Trots att emulatorerna visade sig effektivare än programmerad simulering, utgjorde de ett tveksamt sätt att utnyttja 360-maskinerna, och i de flesta fall har omprogrammering successivt blivit erforderlig.

360 är både ordorienterad och karaktär- (eller byte-) orienterad. Kärnminnets alla 8-bits bytes är direkt adresserbara. Ordoperationerna arbe-

tar på 32-bitsord, i vissa fall på 64-bits dubbelord. Det är fråga om en binär maskinserie, med flytande aritmetik, såväl som decimal aritmetik som arbetar på strängar av 4-bits decimala siffror. Till systemet kan tillhandahållas en mängd perifera enheter.

Ursprungligen kunde bara de mindre modellerna, upp till modell 50, arbeta med multiplexorkanaler, som är fördelaktiga för långsam in/utmatning såsom för läsare, stans och radskrivare. De större modellerna avsågs behöva en eller flera mindre separata dator(er) speciellt för att hantera in- och utmatning. Detta förhållande ändrades snart, anslutning av multiplexorkanaler möjliggjordes för alla modeller. Ett egenartat förhållande har ända fram till 1970 varit att för 360-modellerna ett alltför litet antal selektorkanaler har kunnat anslutas för att balanserat maskinutnyttjande skall kunna erhållas. Kösituationer vid kanalerna har för ett flertal tillämpningsmiljöer uppträtt och medfört låg centralenhetsbelastning.

Projektet 360 hade startats 1961. En huvudsaklig målsättning, bland andra, var att inom IBM standardisera sådant som instruktionskoder, karaktärkoder, aritmetik och liknande. Åtminstone teoretiskt skulle ett och samma program kunna köras direkt på såväl en modell 30 som en modell 70. Denna kompatibilitet sade man sig uppnå genom användandet av mikroprogrammering i samband med minnen av read-only-typ. Maskinvarans fysiska och logiska organisation på mikroprogrammeringsnivå skilde sig emellertid mellan de olika modellerna, och på sätt och vis kan de mindre modellerna sägas vara konstruerade för att simulera de större.

IBM utvecklade en ny maskinvaruteknik för 360-systemet, som man kallade Solid Logic Technology. Det är fortfarande fråga om diskreta komponenter, men av mycket litet format. De elektriska kretsarna är av hybridkaraktär snarare än monolitiskt integrerade.

Det blev efter inte alltför lång tid uppenbart att 360-linjen i sin ursprungligen annonserade form inte kunde tillfredsställa alla användarklasser. På nersidan introducerades en inkompatibel modell 20, senare följd av modell 25. I medelprisklassen kom modellen 44, avsedd huvudsakligen för vetenskapliga tillämpningar, användande sig av en delmängd av övriga 360's instruktionskod. Uppåt i serien har en stor mängd ändringar genomförts, resulterande i modellerna 65 och 75, ersättande 60, 62 och 70, samt 67, 85, 91, och 195, vilka sistnämnda nedan kommer att kommenteras.

IBM's satsning med System/360 har haft ett enastående inflytande på datorindustrin. Trots att främst programvarorna kritiserats hårt av tunga användargrupper, och systemens prestanda jämfört med t ex 7090 (relativt

sett) lämnar en del övrigt att önska, har tusentals system, främst de mindre modellerna, levererats, och flera tusen är ännu 1969 på order. Detta har än en gång för IBM och den övriga världen visat att det med skicklig marknadsföring och kundservice går att sälja datorer i stora upplagor utan att behöva fästa avgörande betydelse vid t ex bearbetningsprestanda.

Flera av 360's egenskaper har accepterats som standard av andra leverantörer. Att flera användarkategorier resonerar helt enligt IBM-terminologi, och har svårt att acceptera andra principer än 360-systemets kan ej heller anses förvånande.

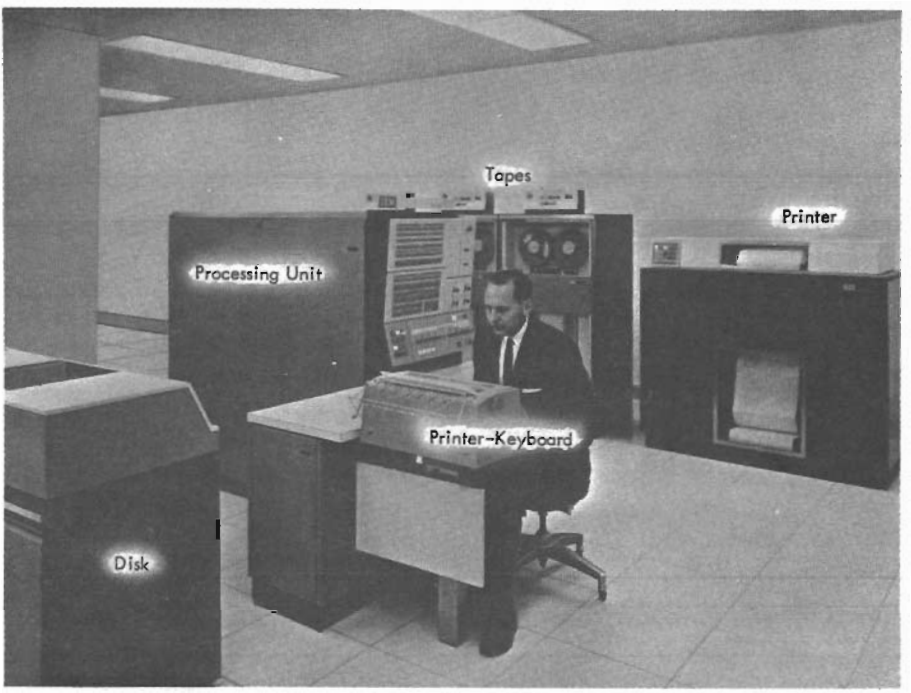


Fig 1.5:1. IBM 360/modell 30.
(Ur F.R.Crawford, "Introduction to Data Processing", Prentice Hall. Med tillstånd av IBM, New York.)

RCA, Honeywell och General Electric

Inte långt efter introduktionen av 360 utannonserade RCA sin Spectra 70, en serie maskiner som kännetecknas av nästan fullständig kompatibilitet med 360-systemet. Härmed fastslog RCA att standardiseringen inom IBM kunde vara minst lika värdefull även för andra leverantörer. RCA använde modellbeteckningarna 35, 45, 55 för att indikera prestanda mellan IBM's 30, 40, 50 och 60, förhoppningsvis till priser som skulle göra produkterna

attraktiva på marknaden. Spectra 70-serien, som sedan 1969 kompletterats med en modell 60, använder sig av integrerade kretsar av monolit-typ, till skillnad från 360. Ett förhållandevis stort antal Spectra 70-datorer har sålts och installerats bl a till mindre belåtna, men kompatibilitets-medvetna, 360-kunder.

Precis ett halvår före 360-utannonseringen anmälde Honeywell sin modell 200. Väsentligen var denna en förbättrad och mycket snabbare version av IBM 1401. Emedan IBM, enligt vad man erfarit, inte hade för avsikt tillhandahålla någon kompatibel efterföljare till 1400-serien, övertog Honeywell detta ansvar. Det efter vad det visat sig tydligen korrekta resonemanget därbakom antog att många 1400-kunder skulle föredra att inte omprogrammera, och att klart bättre ekonomi kunde erhållas med kompatibel hårdvara än via emulering. För de begränsade områden där trots allt vissa inkompatibiliteter mellan 1401 och 200 existerade tillhandahöll man dessutom ett programpaket kallat 'Liberator'.

Honeywell 200 har blivit framgångsrik, och har utvecklats till en hel serie maskiner, från den mindre 100 till den mycket stora 1200. Företaget har med denna serie nått en stabil position inom området kring administrativa tillämpningar.

General Electric, som sedan sin 200-serie lanserat modellerna 400, har även givit sig in på fältet kring mycket stora maskiner. GE 600-serien omfattar 625, 635 och 645, vilka kunde börja levereras 1965. Modellen 645 är i vissa avseenden speciell, och skall senare beröras. Med 625 och 635, huvudsakligen skilda åt av minnescykeltiderna 2 resp 1 mikrosekund, hade man, liksom så många andra leverantörer vid lanserandet av nya system, till att börja med vissa bekymmer, huvudsakligen av programvarukaraktär. Från och med 1967 och framför allt sedan 1968, har emellertid maskinerna visat sig i stort mycket kapabla, med simultan satsvis bearbetning och time-sharing. Inom området time-sharing har General Electric varit en banbrytare.

Andra nyare datorsystem

Fältet är numera så expansivt och omfattande att även en uppräkning av enbart de intressantaste maskinerna skulle bli omfattande. Det kan sägas att IBM, trots sina enorma försäljningsframgångar med 360, har tvingats lägga märke till att denna serie knappast har visat sig principiellt banbrytande i avsedd utsträckning. Den oerhörda expansionen inom datorkonsumtionen som helhet har medgivit ram för expansion för en mångfald mindre till medelstora företag.

Burroughs tillhandahåller en hel serie maskiner, upp till de stora 6500, 7500 och 8500. Framför allt på grund av det ökade intresset för multiprogrammering har 5500 tilldragit sig en icke ringa kundkrets.

Univac, som nått en god position med sina 1107, 1108, 490, 418 och 494-system, lanserade 1968 en ny s k 9000-serie till att börja med modellerna 9200, 9300 och 9400. Denna serie förefaller inledningsvis ha mötts av ett något trögt före. Den är orienterad åt satsvis bearbetning.

Control Data, som har tillskansat sig en dominerande position inom fältet kring de mycket stora maskinerna, tillhandahåller dessförutom såväl små 1700-maskiner som den medelstora 3000-serien. Modellen 3300 kan i ett avseende betraktas som en särling, i det att den var en av de första maskinerna över huvud taget med att tillhandahålla hårdvarupaging (blockutbytesteknik).

National Cash Register stannade vid sin 300-serie tämligen länge, men sedan 1968 erbjuder man även en ny systemlinje, NCR Century Systems.

Ett stort antal relativt nya mindre företag har introducerat sig på marknaden. Bland dessa märks Digital Equipment Corporation med sin expanderande PDP-serie, och Scientific Data Systems med sina Sigma-system.

Ett företag vars inträde på datormarknaden avvaktats med intresse, är det holländska Philips. Indikationer om inträdandet blev tydliga 1965 då detta företag genom ekonomiska transaktioner blev en ledande intressent i det mindre men datorutvecklingsmedvetna företaget Electrologica. Philips' väl utbyggda försäljningsorganisation kan borga för vissa framgångar med den i början av 1969 presenterade datorserien P 350. Som så många andra nykomlingar inom datorproducentkretsen har man valt att gå ut löst, med små modeller först.

Hösten 1969 visades Philips nästa datorsystem upp för första gången. P 1100 är kraftfullare än P 350, men anses dock ligga i den mindre maskinklassen, motsvarande ungefär IBM 360/40. Periferiutrustningen till P 1100 är USA-importerad, men avses successivt bytas ut mot egen tillverkning. Prestandamätningar på Philips-systemen har ännu inte kommit till offentlighetens kännedom. Detta företags datorförsäljningsutveckling kommer emellertid att bli intressant.

Sedan c:a 1968 har ökande intresse börjat fästas vid mycket små datorer (minidatorer), i köpeprisklassen 100 000 - 500 000 sv kr, vilka skulle kunna karaktäriseras som huvudsakligen adress-aritmetiska datorer. Rationell produktion tillsammans med högt utvecklad teknik har medgivit

att även dessa billiga datorer kan erbjuda många konsumenter intressant dataservice. Det är bl a time-sharing-tekniken som utgjort bas för denna expansion. Bland minidator-företag kan även nämnas Hewlett-Packard, Varian Scientific Control Corp., Data SAAB och många andra.

	Minis	Mindre	Medelstora	Stora
Inköpspris (Sv Kr)	100 000 - 700 000	1-2 milj	3-5 milj	6-15 milj

Ungefärligt inköpspris för datorer (med normal kring-utrustning) under 50- och 60-talen.
(Priserna någotsånär stabila, medan prestanda ökande.)

1.6 System för multipel access

Atlas

Manchester University i England och Massachusetts Institute of Technology (MIT) i USA, som båda tog signifikant del i den tidiga datorutvecklingen, var även initiativtagare inom vissa av den senare tidens utvecklingslinjer. Redan 1959 hade forskningsingenjörerna vid Manchester, i samarbete med Ferranti Ltd, ritningarna klara för en banbrytande nykonstruktion, Atlas. Denna maskin använder sig av komplicerad, och kostsam, maskinvara för att lösa problemen kring bästa primärminnesutnyttjande i full multiprogrammeringsmiljö, rörande allokering, överlagring och hierarkisk minnesorganisation. Atlas-konstruktörerna visade hur en-nivås minnessystem möjliggör för varje programmerare att skriva sitt program som om han/hon ensam var i besittning av ett mycket stort primärminne, större än det för konfigurationen fysiskt existerande. Det är fråga om vad som numera kallas 'virtuellt minne'.

Primärminnet är organiserat i block (pages) om 512 ord (48-bits) vardera, och varje programmerare kan använda upp till 2048 sådana block, trots att endast 32 blockutrymmen fysiskt existerar i primärminnet. Ett visst logiskt block kan växla mellan att vara inne i primärminnet eller ute på ett sekundärt trumminne en mångfald gånger under programexekveringen och kan därvid uppta olika fysiska block. Datorn innehåller maskinvara för avancerad adressöversättning (bl a innefattande ett associativt minnesregister), så att en adress som syftar på en plats i ett visst logiskt block

Datorers maskinvaruutveckling

År	Primär-minne (ord)	Op. - tider	Direkt-access-minnen (tecken)	Magnet-band-hastighet (tecken/sek)	Kort el. remsa in (tecken/sek)	Elektro-nik	Data-transm. on-line	Multi-program-mering	Time-sharing o. reell tid
1950-55	1 000	1 millisek	100 000	10 000	100	Rör	-	-	-
1955-60	8 000 - 16 000	10 mikrosek	10 milj	25 000	500	Rör/ trans.	-	-	-
1960-65	16 000 - 32 000	2 mikrosek	50 milj	60 000	1 000	Trans.	Ja	I vissa fall	I vissa fall
1965-70	64 000 - 256 000	0,5 - 1 mikrosek	250 milj	120 000	1 000-2 000	Integre-rade element	Ja	Ja	I vissa fall

automatiskt tolkas som refererande till det fysiska block där det aktuella logiska blocket just råkar befinna sig. Om det logiska blocket inte råkar vara fysiskt närvarande i primärminnet hämtas det snarast in från trumminnet. Normalt är block från skilda program inne i primärminnet, och den erforderliga tiden för att hämta in ett visst programblock kan användas för fortsatt exekvering av ett annat program. Tekniken för att organisera och planera detta på programfragmentering byggda ständiga blockutbyte kan använda statistiska urvalsmetoder.

Minnesskydd, avbrottshantering etc sköts av ett omfattande operativsystem, det första dylika med hög komplikationsgrad. Atlas´ blockutbytesteknik är attraktiv speciellt i en datormiljö där stora skaror användare behandlas simultant, och där omfördelning av primärminne sker med hög hastighet.

Projekt MAC och GE-645

Vid MIT har ett av huvudintressena inom ADB-forskningen under 60-talet rört uppdelning av maskinkapacitet mellan större användargrupper, sålunda liknande Atlas-projektet. Med massiv finansiering från forskningsmyndigheterna har MIT´s Projekt MAC uppbyggts kring denna målsättning. Till att börja med bestod tillgänglig datorutrustning av IBM 7090, en dator som inte direkt var maskinvarumässigt konstruerad för time-sharing-tillämpning. 1963-64 såg man sålunda fram emot ny utrustning, vars maskinvara förhoppningsvis på ett effektivare sätt skulle möjliggöra reliserande av de programvarusystem för multipelanvändning som man arbetade på.

MIT hade under lång tid samarbetat med IBM, och det föreföll naturligt att den nya MAC-utrustningen skulle komma att bli av detta fabrikat. Detta intryck kvarstod även efter utannonseringen av System/360, trots att detta system ej var konstruerat med tanke på t ex MAC-projektets önskemål om maskinvarufunktioner för multipel-accessystem. IBM antog emellertid attityden att MAC-projektet rörde ett enstaka system, och man ställde sig passiv i avvaktan på specifikationer från MIT-gruppen.

Vid denna tidpunkt annonserade General Electric ut ett antal modifikationer till sin 635-modell, som skulle konvertera denna till en ny maskin, modell 636, senare namngiven som 645. Modifikationerna var avsedda befrämja multipelaccess, just av den typ som planerades för Projekt MAC. En egenskap för 645 rörde modularitet, som skulle möjliggöra för multipla processorer att samarbeta med multipla primärminnesenheter och styrenheter för datatransporter. En annan innebar adopterande och utvecklande av Atlas´ blockutbytesprinciper (paging).

Projekt MAC beställde 1964 ett dubbelprocessorigt GE-645-system, och snart efteråt meddelade Bell Telephone Laboratories att man hade för avsikt teckna kontrakt på fyra sådana system (senare reducerat till tre). Det började bli uppenbart att time-sharing var något att satsa på, och att det lätt kunde bli bråttom för att hinna komma med på General Electrics leveranslista.

IBM 360 Modell 67

IBM reagerade häftigt. Uppenbarligen hade man begått ett misstag: detta var inte fråga om ett enstaka forskningssystem, eller ens en obetydlig marknad. Företagets tekniska stab hade tidigare utvärderat bl a hårdvaruorienterade adressöversättningssystem, och hade kommit till den slutsatsen att den logiska elegans som kunde uppnås skulle kosta alltför mycket i extra hårdvara, i komplicerade programvaror, och i sänkning av systemkapacitet.

IBM:s ledning beslöt nu emellertid att gå in på time-sharing marknaden. Kursen lades om, med sikte på maximalt stöd till stora time-sharing-system. Försäljningsorganisationerna anvisades att inte spara någon ansträngning i syfte att undvika att förlora flera order på stordatoranknuten time-sharing.

Det var inte alltför komplicerat att addera hårdvara för blockbytesteknik till de största mikroprogrammerade modellerna, 60 och 62, av 360-linjen. Inom kort bjöds även modellerna 64 och 66 ut, och man erhöll en order från Lincoln Laboratories på förvånande snar leverans av både hårdvara och (!) programvara. Emellertid visade det sig snart att modellsortimentet behövde modifieras. Modellerna 60 och 62 var för långsamma och dyra, och 64 och 66 enligt flera avsedda köpares uppfattning inte tillräckligt avancerade. Samtliga dessa modeller ströks av IBM ur produktionen.

I deras ställe utannonserades den snabbare modellen 65, till samma pris som en långsam modell 60; en än snabbare modell 75, samt den omtalade 67-an, den sistnämnda med associativt minne, segment, blockutbyte, modularitet och en del andra egenskaper som hade utvecklats i samarbete med bl a University of Michigan.

Time-sharing-teknik möjliggör tillämpningsorienterat utvecklingsarbete inom många intressanta områden, dialogprogrammering, avancerad filhantering, informationssökning, grafisk databehandling, datorstyrt konstruktionsarbete, datorstödd undervisning etc. Dessa och andra nyckel-

projekt för forskningen förutsätter sålunda i stor utsträckning "tids-delade" multiaccessystem. 1965 föreföll modell 67 det mest lovande av projekten kring time-sharing på stora maskiner, och ett stort antal universitet och forskningsorganisationer beställde, eller planerade beställa, 67-or som centrum i sina multipelaccessystem.

IBM satsade hårt på programvaruutvecklingen kring modell 67. Entusiastiska potentiella användare planerade installationer med hundratals terminaler av skrivmaskinstyp simultant direktanslutna. Vid slutet på 1966 blev det emellertid allt klarare att 67-ans prestationsförmåga överskattats. Simuleringsstudier indikerade att det avsedda programvarusystemet skulle få svårigheter till och med med ett ringa antal terminaler.

Många kunder drog tillbaka sina order. Ett antal 67-system installerades under 1967, med ett programvarupaket som tillhandahöll begränsad service till omkring åtta direktanslutna skrivmaskinskonsoler. Flera kompetenta kunder, främst universitet, startade arbete på egna alternativa programsystem, trots att en snar andra version av IBM-paketet utlovade förbättrad kapacitet. Endast ett mycket begränsat antal 67-or har fram till 1969 kunnat avsättas. IBM satsar från 1969 bl a på en dubbelprocessorig modell 65 för time-sharing.

Det kan förefalla osannolikt att tillfredsställande prestanda kan uppnås för något centraliserat, tungt, time-sharing-system utan avgörande hårdvaruförbättringar som kan komma ut av forskning, vars resultat vi ännu 1969 inte har fullständigt grepp om.

Andra time-sharing system

Om de större time-sharing systemen inte riktigt har kunnat leva upp till ställda förhoppningar, är läget emellertid ljusare för mindre system. General Electric har med ett programvarupaket från Darthmouth College haft stor framgång med sitt GE-265-system (GE-235 plus Datanet-30, en speciell datakommunikationsmaskin). Detta system kan, med sin goda tillförlitlighet, men mindre höga komplikationsgrad, sägas ha varit väsentligt före konkurrenterna med kommersiellt utbjuden time-sharing.

Flera av de mindre time-sharing-systemen består liksom GE-265 av moderna programvarupaket på konventionella datorer, men andra har realiserats med hjälp av hårdvara som konstruerats speciellt för time-sharing-applikationer. SDS 940, en modifikation på den mera konventionella 930, utvecklades i nära samarbete med University of California, och har framgångsrikt saluförts av Scientific Data Systems, som därjämte har time-sharing bl a på sin Sigma 7.

RCA har utökat sin Spectra 70 modell 45 med viss hårdvara för adressöversättning, och tillhandahåller den resulterande modellen 46 för timesharing-användning. Control Data 3300 och Digital Equipment PDP-10 är andra maskiner med speciellt avpassad hårdvara.

Användning av datorer via direktanslutna avlägsna terminaler av t ex skrivmaskins- eller (när dessa blivit billigare) bildskärmstyp ökar alltmer i popularitet, och skapandet av system för multipel access kan sägas utgöra en milstolpe i datorhistorien.

CDC 7600

Efter att en tid ha njutit av framgångarna med 6600 utannonserade Control Data 1968 sin 7600, en kraftfullare och i flera avseenden förbättrad 6800, vilken sistnämnda därmed utgick ur produktionslinjen. Man saluför också två mindre 6000-maskiner, 6400 och 6500, vilka liknar 6600 i de flesta avseenden bortsett från den lägre processorparallelliteten och det lägre priset.

CDC 7600 är liksom 6000-serien uppbyggd av en central processor och ett flertal perifera parallellt arbetande processorer och arbetar även med 60-bits ord. De perifera processorernas funktioner har uppsnabbats och utvidgats. En skillnad ligger emellertid i att minnesstrukturen är ny. Maskinen har tre skilda kärnminnesnivåer, ett 27.5 nanosekunders snabbregister på 12 ord vari t ex täta instruktionsloopar kan exekveras utan referens till de större primärminnena, ett 275 nanosekunders exekveringsminne på 64 K ord och ett större 1.75 mikrosekunders kärnminne på 512 K ord. Endast ett program avses normalt befinna sig inne i exekveringsminnet åt gången, även om flera program kan befinna sig där samtidigt. När ett avbrott skall behandlas eller vid programkontrollbyte av andra skäl, 'rullas' aktuellt program ut på det större kärnminnet, och ett annat, som väntar på processorkapacitet, rullas in. Härmed erhåller man ett något sämre utnyttjande av exekveringsminnet, men den mycket höga överföringshastigheten (27.5 nanosekunder/ord) mellan de båda större kärnminnena samt den jämfört med konventionella multiprogrammeringssystem rationaliserade programväxlingsadministrationen, torde ändå borga för mycket stor bearbetningskapacitet. Rapporter från körningar på det första exemplaret 7600, som installerats i Livermoore i slutet av 1968, pekar mot en centralenhetskapacitet kring utförande av 10-20 milj instruktioner per sekund. Det är tveksamt om något på den kommersiella marknaden f n tillgängligt system kan komma i närheten av en dylik hastighet. 7600-systemet är emellertid nominellt dyrt. För en rimlig konfiguration kan köpepriset c:a 50 milj kr uppskattas. Det beräknas att CD 7600 exekverar beräknings-

bundna program c:a 5-10 ggr snabbare än CD 6600, varför kostnaden per databehandlad enhet sannolikt visar en sjunkande trend. Som nämnts är CD med sin 7600 inne på en 'uniprogrammeringslinje', en intressant utveckling med hänsyn till de substansiella spilltider som de flesta leverantörer fått känning av i samband med programadministration i 60-talets multiprogrammerade datorer. Längre fram i tiden skymtar CD's superdator STAR, till vilken vi får återkomma i annat sammanhang.

IBM's modeller 90

Under flera år efter det mindre lyckade STRETCH-projektet föreföll IBM ha tappat intresset för mycket stora maskiner. Inte långt efter utannonseringen av 360-systemet var man emellertid redo igen, och presenterade 360 modell 90, en som det avsågs uppenbar konkurrent till CDC 6600. Då CDC omedelbart svarade med att lansera 6800, logiskt identisk med 6600 och till oförändrat pris men fyra gånger snabbare, tvingades IBM fortsätta karusellen, med modellerna 91, 92 och 95.

Till slut stannade man huvudsakligen för modellen 91, en maskin som återupplivade look-aheadprincipen från STRETCH. Med hjälp av interfolierad minnesteknik, mycket snabb aritmetik och ett 750-nanosekunders minne ville man uppnå en planerad exekveringshastighet av en instruktion per 60-nanosekunderscykel. Detta mål borde åtminstone kunna erhållas för modellen 95 som är samma maskin utom för närvaron av 1 Mbytes 120-nanosekunders tunnfilmsminne. Möjligen kan den effektiva hastigheten för detta stora minne ligga närmare 200 nanosekunder på grund av de fysiska dimensionerna.

1967 meddelade IBM dock att man inte accepterade fler order på modellerna i 90-serien. Endast de 20 redan beställda systemen skulle komma att levereras. Året därpå lanserade man i stället modellen 85, ett nytt mycket stort system, logiskt enklare men i många avseenden lika kraftfullt som modellen 91. IBM 360/85 arbetar med ett minnessystem i viss mån liknande CDC 7600's, med automatiska programöverföringar till ett litet (16 K - 32 K bytes) integrerat-kretsminne, en minnestyp som alltså därmed trots allt dök upp i 360-serien.

Det kan tänkas att modellerna 91 och 95 drogs in på grund av det långt framskridna arbetet med modell 85, som synes uppvisa bättre förhållande kapacitet/pris.

Ett nytt försök i den största datorklassen görs av IBM hösten 1969. Man introducerar en 360-kompatibel modell 360/195, avsedd att ta upp kampen

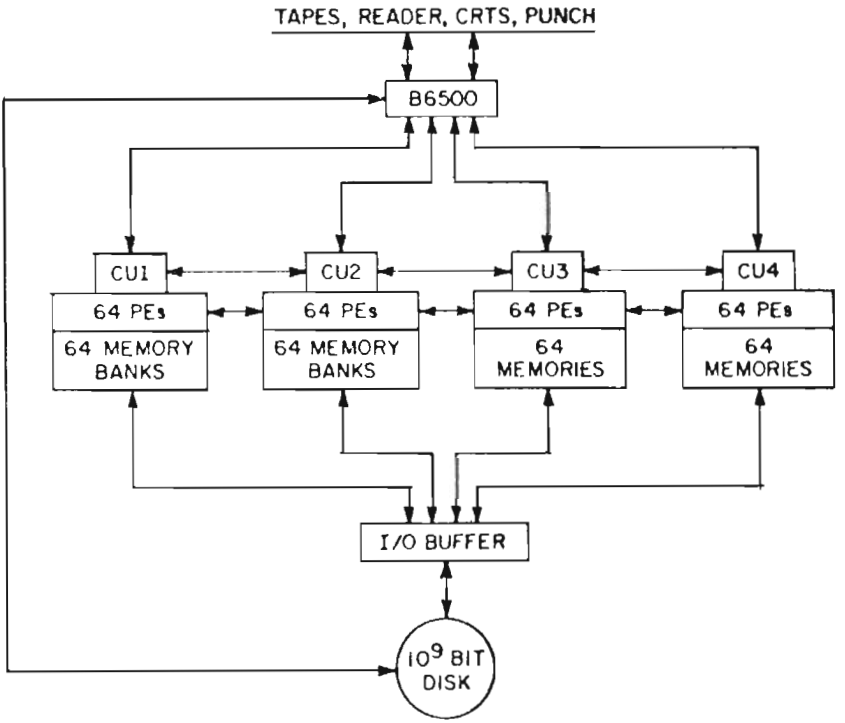


Fig 1.6:1. Principskiss över ILLIAC IV systemet.

med bl a CD 7600. Leverans av modellen 195 dröjer dock till 1971. Liksom modellen 85 använder sig 195-an av integrerade kretsar i centralenheten, till skillnad från de tidigare modellerna. En uppdelad minnesstruktur är anammad: 8, 16 eller 32 minnesmoduler om 128 K bytes vardera (756 nanosek) arbetar 8-vägs eller 16-vägs interfolierat. Synnerligen snabb datatransport aviseras. Modellen 195 anges för "vissa aritmetiska" beräkningar vara mellan 1.2 och 3.2 gånger så snabb som modell 85.

I juni 1970 utannonserar IBM de första delarna av den väntade efterträdaren till System/360. (Endast begränsad information därom finns för närvarande tillgänglig.) Den nya serien, kallad System/370, kan antas komma att omfatta ett flertal modeller, varav dock initialt endast 370/155 och 370/165 frisläpps. Den mindre av dessa, modell 155, anges lämplig som "ett steg uppåt" förn användare av 360/40 och 360/50, medan den större, 370/165, kan ersätta 360/65 och 360/75. Båda 370-modellerna arbetar med ett mindre associativt buffertminne, bl a i avsikt att snabba upp utnyttjandet av långsamma yttre kärnminnen, samtidigt som processorn hålls

verksam. 370/155 kan förses med upp till 2 Megabytes primärminne, medan 370/165 maximalt tar 3 Megabyten. Internt är 370/155 fyra gånger så snabb som 360/50, och 370/165 fem gånger så snabb som 360/65. Det är dock tveksamt om dessa internhastigheter kan motsvars av antal bearbetade jobb i praktiken, emedan därför förefaller krävas "optimal" kodning, som utnyttjar den för 370 utvidgade instruktionsrepertoiren.

System 370 anges programkompatibel med 360-serien. Intressant är att inget operativsystem för 370 annonseras ut tillsammans med maskinvaran, ett numera ej sällsynt, men dock mindre lyckligt förhållande i branschen. Mycket intresseväckande kan sägas vara att köpare av 370 från IBM ges ett års underhållsgaranti på maskinvaran. Härigenom nedbringas köparens initiala kostnader märkbart, och IBM ansluter sig därmed till en konsumentorienterad politik, som alltför länge lyst med sin frånvaro i datorsammanhang.

ILLIAC IV

Under sin tid vid Westinghouse, kring mitten av 60-talet, propagerade D. Slotnick vid ett flertal tillfällen för ett projekt med en högt parallell dator, kallad SOLOMON, men lyckades aldrig erhålla ekonomiskt stöd. Efter att ha flyttat till University of Illinois ljusnade situationen ekonomiskt, tilldelning av försvarsanslag möjliggjorde start av projektet. Maskinen döptes därvid adekvat nog om till ILLIAC IV, och den aktuella konstruktionen utförs i samarbete med Burroughs Corporation.

Denna supermaskin kommer att ha 256 processande element, vart och ett med eget tunnfilmsminne (200 nanosek) om 2 048 ord à 64 bitar, egen höghastighetsaritmetik och indexregister. De 256 processorerna är organiserade i 4 'vektorer' om 64 processorelement var. Varje vektor har sin kontrollenhet som fördelar instruktioner inom vektorn. Exekvering sker med hjälp av en 8 ords look-aheadprincip, som vid simuleringstester visat sig mycket effektiv. Program som väntar på exekvering residerar på ett skivminne (10^9 bitar) med fasta huvuden och medelaccesstiden 20 millisek.

Ett visst mått på omfånget för detta maskinprojekt erhålles genom betraktande av att en speciell Burroughs 6500 (i sig själv sannerligen ingen liten maskin) är avsedd att ingå i systemet, enbart för att handha den övervakande kontrollen.

Ett speciellt högre programmeringsspråk håller på att konstrueras för att tillvarata ILLIAC IV's avancerade möjligheter till bearbetning av parallella processer. Detta språk, som är en utvidgning av Algol, kallas TRANQUIL.

Konstruktörerna förutspår att ILLIAC IV kommer att kunna uppvisa en fantastisk kapacitet, för flytande räkning kring ett hundra gånger CDC 6600. Emellertid har principerna för maskinkonstruktionen vållat en hel del debatt bland specialister på stora system. Kritik har bl a hörts från IBM, som i ett nytt Advanced Computer System-projekt själva arbetar med en något annorlunda processorfilosofi. Deras en-processors supermaskin sägs ha planerats för att kunna utföra 100-200 milj instruktioner per sekund. IBM-projektets realiseringstakt förefaller emellertid ha dämpats något, i linje med politiken att undvika introduktion av nya maskiner som inte är programvarukompatibla med System/360.

ILLIAC IV planeras sättas i drift i sitt första exemplar 1971.

1.7 Situationen i Asien

Utvecklingen i Sovjet

Datoraktiviteten i Sovjetunionen startades inte bara tämligen sent, utan har också i vissa avseenden förlöpt i ett långsammare tempo än som kunnat väntas. Med hänsyn till den höga teknologiska standard som bevisats inte minst av framgångarna med de ryska rymdexperimenten kunde en snabb datorexpansion ha förmodats. Att så knappast ännu har blivit fallet får ses i samband med den från den ryska statens sida styrda konsumtionsutvecklingen. Kapitalvaruproduktion av icke-agrikulturellt eller av icke-militärt slag har inte befunnit sig i centrum för de ryska samhällsplanernas intresse under framför allt 40- och 50-talen. Inte heller har man synbarligen tillräckligt allvarligt uppmuntrat till privatindustrialistisk konkurrens, något som varit en av de starkaste expansionsdrivande krafterna för västvärldens datorutveckling. Det är först under senare delen av 60-talet som man i Sovjet börjar kunna märka ett visst närmande till det västliga konkurrenssamhällets funktion i detta avseende.

Sålunda har datorer kommersiellt inte utbjudits i Sovjet i större skala fram till 60-talets slut. Tveklöst har emellertid en intern utveckling inom området pågått, små som stora utrustningar har konstruerats, men de har troligen använts huvudsakligen för lösning av militära och rymdtekniska problem.

Detaljerad information om ryska datorer är här i väst svår att finna. Detta har samband med den ovan nämnda bristen på kommersialisering inom den sovjetryska datormarknaden. Manualer, broschyrer och andra beskrivningar framställs uppenbarligen endast i små upplagor. Facktidskrifter ger endast kortfattade kommentarer.

Datorutvecklingsarbetet startade i slutet av 1940-talet, ledande fram till den av professor Ljebedev konstruerade MESM, Sovjets första egna maskin. Denna blev färdig 1952, och uppgavs kunna utföra 2 000 operationer per sekund. Strax därefter uppträdde maskinen STRELA, ungefär samtidigt som det första exemplaret i BESM-serien, BESM-1, med approximativt samma beräkningsförmåga som MESM.

1950-talets fortsättning kännetecknas av en med västmått mätt relativt långsam utveckling. URAL-seriens första modell såg dagens ljus 1955, men uppvisade en medelinstruktionstid på blott ca 1 millisekund, sålunda endast hälften så hög hastighet som MESM:s. MINSK-1 kunde levereras 1958, dess bearbetningskapacitet är dock ej känd, utan får ställas i relation till de endast 6 000 operationer/sek som den senare brodern MINSK-2 presterade vid färdigställandet så sent som 1962.

Den vid Moskva-universitetet år 1959 färdigställda SETUN har ådragit sig intresse, mindre på grund av sin begränsade kapacitet på ca 4 000 operationer/sekund än på grund av sin interna arbetsprincip. Denna maskin är nämligen upphovet till rykten om 'en rysk ternär maskin'. En relativt enkel matematisk analys ger vid handen att basen 2 ej är optimal för fix talrepresentation i ett givet maskinord. Bästa bas är talet e (= 2.718 ...), med 3 som närmaste granne. På grund av maskinvaruekonomiska synpunkter har emellertid det binära talsystemet undantagslöst fått ligga till grund för digitala datorers minneskonstruktion. Det ternära systemet, med basen 3, är emellertid bättre om minnesutnyttjande är den absolut dominerande konstruktionsfaktorn, bortsett sålunda från maskinvaruframställningsekonomi. SETUN var, i motsats till ryktesspridningen, försedd med ett binärt konventionellt kärnminne, men kärnorna var grupperade enligt en princip som vid programmering gav sken av ett ternärt minnestänkande. Närmare detaljer om denna strukturering är för författarna okända. Maskinen var ännu 1967 i drift vid universitetet i Moskva, huvudsakligen använd för undervisningsändamål.

Både URAL- och MINSK-serierna har under 60-talet utökats med ett antal transistoriserade modeller med ökad kapacitet. Den medelstora MINSK 22 synes ha nått största spridning. Det är dock främst namnet BESM som i västkretsar låtit tala om sig. Den senaste kända modellen, BESM-6, är det huvudsakliga upphovet härtill.

Enligt tillgängliga källor kunde denna maskin köras igång 1966, medan erforderliga maskinvaruförbättringar indikerar 1967 som mera korrekt starttidpunkt. Maskinen har kommenterats både i Pravda och i Izvestija. Även ur västindustriell synpunkt är dess prestanda och interna arbetsorganisation intressanta, speciellt med hänsyn till tidigare sovjetutveckling.

Av intresse i samband med ryska maskiner, och i hög grad med BESM-6, är det primärminnesekonomiska tänkandet. BESM-6 är konstruerad för maximalt 32 K 50-bitsord, men kan med maskinvarutillsats utökas till 64 K. Med hänsyn till att datorn med 32 K redan 1967 enligt uppgift körts med 10 program under exekvering samtidigt måste operativsystemets minneskrav vara lågt, likaväl som assemblerns alternativt Algolkompilatorns minnesekonomi. (Algol är det i Sovjet dominerande problemorienterade programmeringsspråket. I detta fall har troligen assemblyspråk använts). Till datorn kan anslutas upp till 16 trummor om 32 K ord var. Minnescykeltiden 2 mikrosekunder samt den uppgivan nominella bearbetningshastigheten 1 miljon operationer/sekund indikerar att exekvering förlöper på ett sofistikerat sätt, utan onödiga minnesreferenser. Även om den effektiva uppmätta bearbetningsförmågan för numeriska problem säges vara reducerad med en faktor 2 i förhållande till den nominella innebär detta god kapacitet.

BESM-6 uppges arbeta med en femnivåig look-ahead, och ha 16 generella 15-bits 300 nanosekunders indexregister. Det kan anses tveksamt om look-ahead-funktionens uppdelning på fem nivåer innebär någon reell skillnad jämfört med "en enda nivå". En look-ahead-funktion leder nödvändigtvis till en strukturering som sannolikt likaväl kan rubriceras som en- som flernivåig. - Maskinen är den första en-adressdatorn i BESM-serien, samtliga fem tidigare modeller har arbetat med tre-adressord. De 50 bitarna i ett BESM-6-ord (varav 2 är kontrollbitar) möjliggör lagring av två instruktioner i varje ord.

Minnesstruktureringen innebär blockutbytesteknik, paging, i block om 1 024 ord, relokering och fragmentering med hjälp av 32 speciella register för minnesskydd. Maskinen anges vara konstruerad för time-sharing, medan information om dennas programvarumässiga utseende ej stått att finna. Sannolikt är time-sharing ännu 1969 ej i drift. Sovjetdatorer har oftast levererats utan adekvata standardiserade programvaror, varför stora staber av systemprogrammerare varit nödvändiga vid de flesta datorcentra. Leverantörerna har inte tagit ansvar för programvaror. Intresset för kompletta installationer, med både maskin- och programvara vid leverans, har emellertid vuxit sig allt starkare mot slutet av 60-talet.

Beträffande BESM-6 kan tilläggas att, enligt W. B. Holland, Datamation sept. 1969, maskinen redan hade byggts i prototyp innan allvarligt programvaruarbete startades. Härvid befanns att datorns maskinvarumässiga struktur knappast var lämpad för samarbete med önskade programvaror. Det initiala operativsystemet visade sig inadekvat, och en total omskrivning blev nödvändig. Detta har medfört allvarliga förseningar för datorsystemets spridning på marknaden. En rapport över BESM-6

med det nya operativsystemet presenterades vid Sovjetunionens "First All-Union Conference on Programming" i november 1968.

BESM-6, som i vissa avseenden påminner om Atlas-maskinen, bör anses som en kraftfull dator, även om dess prestanda bör ligga klart under CD 6600's eller IBM 360/85's.

Produktion av en ny datorserie har fr o m 1965 planerats i Severodonetsk (Ukraina). Det är fråga om samarbete mellan "the Scientific Research Institute of Control Computers" och den sovjetryska staten. Projektet går under arbetsbeteckningen ASVT (ungefärligt stående för "modulär maskinvara"). Erfarenheterna från BESM-6 har manat till försiktighet, och man dröjde därför med annonseringen av ASVT-projektet till 1968. Projektet anges innebära konstruktion av en modulär maskinserie, varav de tre första individerna betecknas M-1000, M-2000 och M-3000. Om dessa anges att de baseras på maskinvara med integrerade kretsar samt har gemensam instruktionsrepertoar. Denna repertoar uppges bli kompatibel med IBM System/360, ett intressant förhållande som indikerar möjligheten av ett kommande programvaruutbyte. De tre annonserade datorerna kan anses komma att befinna sig i den mellanstora datorklassen, att döma av angiven medelinstruktionstid på ca 10 mikrosekunder för den största, M-3000. Ordlängden anges till 32 bitar, med 16 bits halvord och 64 bits dubbel precision. Varje 8 bits byte är kompletterat med en paritetsbit, och primärminnet är fysiskt indelat i block om 36 bits ord. Det uppges att M-3000 kommer att utgöra centrum i flygbolaget Aeroflots bokningssystem "Sirena", som planeras handha samtliga "persontransaktioner" som berör Moskva.

Den allmänna trenden i Sovjetunionen har under 60-talet gått mot fabricering eller principiell design av stora datorsystem, med ett undantag. Detta undantag utgörs av verksamheten vid Cybernetik-Institutet i Kiev, ledd av V. Glushkov, en av de mera prominenta sovjetryska vetenskapsmännen. Glushkov har ansvarat bl a för framtagningen av datorserien MIR. Dessa datorer är avsiktligt små och långsamma, MIR-1 ståtar med medelinstruktionstiden 4 millisekunder. "Ett huvudmål har rört att befrämja kontakten mellan människa och maskin" anges för MIR-serien. Sålunda har man haft icke-specialister i åtanke vid systemspecificeringen. Kommunikation sker via skrivmaskin medelst ett högre-nivå-språk, som syntaktiskt är speciellt avpassat för matematiker, ingenjörer och liknande vetenskapsmän. Detta språk använder ryska och latinska karaktärer, en serie symboler för standardoperationer, samt avskiljare. Funktioner för begrepp av typen CALCULATE, LENGTH, IF, ARRAY etc existerar. En utveckling av MIR-1 är under utveckling. Denna kommer möjligen att arbeta i time-sharing. MIR är intressant såtillvida att den representerar det första allvarliga sovjetryska försöket att ge möjlighet för icke-pro-

grammeringskunnig personal att med minimal arbetsinsats nå kontakt med datorer. Någon allvarlig trend i denna riktning kan emellertid ännu inte sägas existera.

En klar ökning i antalet kommersiellt salufördа ryska datormodeller kan noteras för 60-talets senare år. I detta sammanhang har endast ett fåtal av dessa kunnat kommenteras, såväl av utrymmesskäl som av brist på detaljinformation.

Beträffande maskinvarumässig komponentpålitlighet för ryska datorer har data huvudsakligen funnits tillgängliga för situationen fram till mitten av 60-talet. Dessa data härrör sig från rörbestyckade maskiner, och angivelser om förfluten-tid-mellan-fel är låga. Transistorteknikens genombrott, som i Sovjet kom senare än i västvärlden, bör öka komponentpålitligheten. Skäl för låg systempålitlighet anges vara dels dålig kvalitetskontroll vid monteringsbanden, och dels, och framför allt, brist på kvalificerad operatörs- och underhållspersonal. Detta senare har givetvis samband med utbildningsfrågor, där man i Sovjet ännu knappast synes ha satsat tillräckligt intensivt.

Ett allmänt intryck från Moskva-utställningen AVTOMATIZATSIYA 69, enligt Datamation Sept. 1969, är att speciellt rysk periferiutrustning ännu vid 60-talets slut tycks besitta dålig pålitlighet. Konstruktionsmetoderna har synbarligen ännu inte kunnat leda fram till önskvärd produktstabilitet.

Som avrundande omdöme kan försiktigtvis sägas att sovjetryska datorer tekniskt sett bör ligga 3–4 år efter maskinerna i väst (en ökad import till öst har kunnat noteras), medan det allmänna ADB-medvetandet bland användare ute på fältet, samt definitivt programvarustandarden, bör ligga ytterligare något år efter väst-situationen.

Kinesisk datoraktivitet

I datoravseende, likaväl som inom flera andra områden av teknologisk natur, kan Kina sägas ligga några år efter Sovjet, ett gap som dock kan väntas minska under 70-talet, under förutsättning att inga än mer förvärrade politiska kriser kommer att skaka östligaste Asien. Under 1950-talet en tid av fast vänskap mellan Sovjet och Kina, delade de ryska datorexperterna beredvilligt med sig av sitt kunnande till sina kinesiska kollegor, vilket var till ovärderlig hjälp för den kinesiska utvecklingen. Under denna tid sände Sovjet inte mindre än 8 500 specialister till Kina, och dessutom bereddes 10 000 kinesiska ingenjörer och tekniker plus ca 1 000 kinesiska vetenskapsmän och 11 000 kinesiska studenter möjlighe-

ter till datorknutna studier i Sovjetunionen. Härförutom överlämnades inte mindre än 21 000 vetenskapliga och tekniskt orienterade specifikationer till Kina. (Det synes ej omöjligt att ryssarna kan ha ångrat denna tidigare meddelsamhet rörande sin elektronik, som väl kan komma att ha militär betydelse.)

Före 1956 var det kinesiska teknologiska kunnandet otillräckligt för datorkonstruktion. Man hade inte ännu hunnit smälta intrycken från de ryska kontakterna. Successivt började emellertid den detta år upplagda 12-årsplanen för vetenskaplig utveckling realiseras, och den 1 augusti 1958 annonserades den första kända kinesiska datorn. Detta historiska datum kom senare att i brist på annan benämning ge namn åt produkten. "August 1" förefaller att döma av tillgängliga specifikationer att vara en nära nog kopia av den ryska URAL-1, en av Sovjets långsammare maskiner, som där redan hunnit framställas i ca 120 exemplar. Maskinen klarade 1 000 instruktioner/sek, och det är inte omöjligt att ryssarna kan ha överlåtit fulla specifikationer för denna dator tidigare, varför händelsen i Kina egentligen bör tillmätas endast måttligt värde.

Sporrade av passerandet av denna milstolpe startade en intensiv vidareutvecklingsverksamhet vid bl a Institutet för Datorteknologi i Peking och vid ett flertal andra universitet. Smärre utrustningar under fantasieggande rubriker såg dagens ljus, oftast av processtyrningskaraktär och med västmått mätta mycket enkla apparater. Informationsspridningen från Kina har alltid varit, och är fortfarande, svår att verifiera. År 1959 producerades vid South China College en "automatisk översättningsmaskin", med 3 000 elektronrör och 1 000 halvledarelement. Denna anges direkt ha översatt en rysk via skrivmaskin inmatad textbok till på någon sorts skärm utmatade kinesiska bokstavskaraktärer. Det må tvivlas något på fullständigheten i det program som år 1959 styrde denna process.

En höjdpunkt i den kinesiska datoraktiviteten nåddes april 1959 då New China News Agency utannonserade en ny inhemsk "höghastighets universell elektronisk dator". Att döma av kortfattade beskrivningar förefaller denna maskin att vara en nära kopia av BESM-2, som man i Sovjet emellertid inte hade färdig för frisläppande förrän mot slutet av detta år! Den innehöll 4 200 elektronrör och 4 000 halvledarelement, hade inmatningsorgan för hålremsa och utmatning på papper med 900 rader/minut (troligen med ett lågt antal skilda tryckkaraktärer). Datorn kunde utföra 10 000 operationer/sek.

Det är begripligt att man i Sovjet drog öronen åt sig efter detta hugg i ryggen. Här hade man beredvilligt delat med sig av specifikationer och kunnande, och så går kineserna och kopierar utrustningen tämligen komplett och lanserar produkten t o m tidigare än i Sovjet! Detta faktum

tillsammans med en än viktigare förskjutning i den tidigare politiska broderskapen fick till följd att Sovjet mot slutet av 1959 totalt och fullständigt drog tillbaka sitt teknologiska stöd till Kina.

Detta var ett hårt slag. Kineserna hade ännu ingalunda nått tillräckligt teknologiskt mognande för att själva framgångsrikt kunna arbeta vidare. Man hade huvudsakligen ägnat sig åt plagierande, med egna modifikationer. Hur tungt man varit beroende av sovjethjälpen bevisas inte minst av det faktum att under 5 år därefter ingen känd kinesisk nykonstruktion dök upp. Först vid mitten av 1965 utrapporterades en ny elektronisk digital maskin, nu från Tsinghua-universitetet i Peking. Ungefär samtidigt talas det om viss produktion på andra håll i landet. (En 1964 presenterad schackspelande mindre maskin kan i det här sammanhanget förbigås.)

Tsinghua-datorn hade ungefärligt samma bearbetningskapacitet som den redan 6 år tidigare presenterade kopian av BESM-2, sålunda ca 10 000 operationer/sek. Denna tidsrymd, 6 år, kan alltså sägas ha åtgått för att hämta in kunskaper som gått förlorade i samband med indragandet av den ryska hjälpen vid slutet av 1959.

Under det att fram till 1965 sålunda endast ett fåtal kinesiska digitala datorer lanserades, var man inte passiv inom anknutna områden. En klar inriktning mot mera specialiserad utrustning för processtyrning kunde tidigt noteras. Apparater av denna typ presenterades också i väsentligt större upplagor än generella datorer. Dessutom har tillkomsten av ett antal analoga maskiner av mestadels mindre format noterats.

Den under senare delen av 60-talet så omtalade kulturrevolutionen innebar för kinesisk teknologi en dämpning i utvecklingstakten. Även om vetenskapsmän med elektroniskt kunnande lär ha utgjort snart sagt de enda som officiellt befriats från tvånget om aktivt deltagande i denna folkrörelse, kan dessa år knappast kunna sägas ha varit en teknologiskt lika aktiv tid som det nu mognade kunnandet kunde inneburit. Det är ingen tvekan om att framställning av digitala datorer under 60-talets sista år legat högt på den kinesiska industriella prioritetslistan. I en planangivelse rankas datorutveckling som näst viktigaste intresseobjekt, närmast efter atomkraftsforskning och -utveckling. Man har tveklöst insett den stora betydelsen av digital teknik för moderna försvarsändamål. Kommersiellt saluförande av egna datorer i större upplagor kommer dock troligen att dröja på grund av denna militära prioritet. Sekretessen kring de militära projekten kan också vara ett av skälen till att så litet meddelats om den kinesiska datorutvecklingen efter 1965.

Trots att USA under 60-talet motsatt sig datorexport till Kina har kineserna utökat sin datorimport. Det är främst Frankrike, England och Japan som kommit ifråga. Den japanska exporten har dock begränsats av att flera ledande japanska firmor är hårt bundna av amerikanska samarbetskontrakt. Kineserna lär dock, enligt tidskrifter 1966, föredra Japan som samarbetsområde framför Europa.

Bristen på information om den kinesiska datorproduktionen är beklaglig. Här kan avslutningsvis endast relateras att Kina 1969 "konstruerat och byggt en heltransistoriserad dator". Denna uppges vara av digital typ, och användbar för "alla slag av uppgifter". Den kan möjligen vara identisk med den med beteckningen DJS-21 på Kanton-mässan 1969 utställda datorn. Denna uppges kunna utföra 60 000 instruktioner/sek, är liten till fysiskt omfång, och ägnar sig på nämnda mässa åt att vokalt utropa slagord av typen "Östern är röd" och "Leve ordförande Mao", samt att rita diagram över Den himmelska fridens torg i Peking. Närmare detaljer är för författarna f n okända.

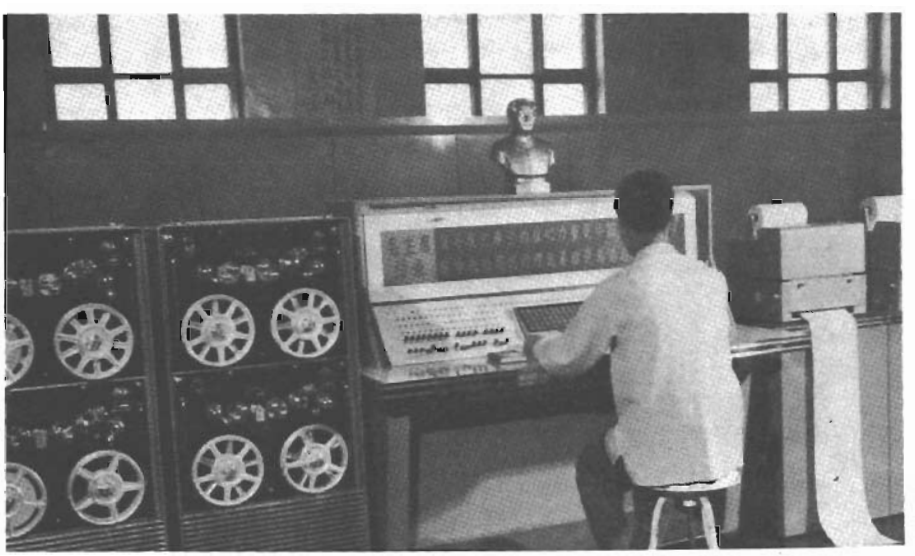


Fig 1.7:1. Kinesisk dator (prestanda för författarna okända).

Det japanska undret

Den osedvanligt höga framstegstakten för japansk teknologi under 50- och 60-talen är väl känd. Några synpunkter kring bakgrund härtill kan ges:

1) Den amerikanska industrin har beredvilligt exporterat erforderliga basprodukter. För elektronisk industri har japanerna emellertid f n nått en position då importen kan begränsas och ersättas med inhemska produkter.

2) Japanerna besitter en känsla av att icke vara underlägsna något annat land vad gäller förmåga och kapacitet.

3) Japansk industri har aktivt och permanent uppbackats från regeringshåll. Detta avser bl a kreditgivning och substansiella skattelättnader.

4) Samarbetsorganisationer inom den japanska industrin har arbetat effektivt och konstruktivt, med totalmål snarare än delmål i sikte. Samarbetet synes ha fungerat osedvanligt väl, med västmått mätt.

5) Den kraftfulla japanska satsningen på inhemsk forskning och utveckling.

6) De japanska universiteten har kunnat spela en aktiv och stimulerande roll för industrin. M. Akanuma, ENEA, formulerar 1968:

"A sort of symbiosis seems to have been established between the computer manufacturers and the universities. This results in the latter being provided with prototypes of the latest and most powerful equipment, while the former in return receives a feedback of observations and suggestions for improvements made by the university mathematicians and computer specialists."

Tillsammans bildar dessa skäl, och andra, en bas som har visat sig kunna möjliggöra utomordentliga framgångar. Framtidsforskaren H. Kahn, USA, anger det nästkommande århundradet, om inget förhindrande inträffar (världskrig e d), som Japans, vad gäller bl a teknologisk produktion. Att döma av framstegstakten de senaste 15 åren synes detta åtminstone inom datorfältet icke orealistiskt.

Satsningen på datorproduktion inleddes kring 1955, sålunda ca 10 år efter västvärlden. (På de gångna 15 åren fram till idag synes Japan mycket nära ha hämtat in detta väst-försprång, i sanning ingen dålig prestation med hänsyn till dynamiken i väst.) Den första japanska datorn, FUJIC, färdigställdes 1956 av Fuji Photo Film Co. Denna var en elektronrörs-

maskin som framställdes i syfte att snabba upp beräkningsarbete rörande linskonstruktion.

I stället för att spilla tid på rörbestyckade datorer togs steget tämligen omgående in i transistoreran. Redan 1957 kunde den transistoriserade MARK III lanseras. Denna härrörde från the Electro-technical Laboratory (ETL), tillhörande handels- och industriministeriet. Den följdes efter ca ett halvår av Mark IV, Japans första kommersiellt salufördа dator.

Vid ETL producerades 1958 MUSASHINO-1 samtidigt som Tokyo-universitetet lade sista handen vid sin PC-1, även den transistoriserad. Teknologin bakom dessa datorer inspirerade snabbt tillverkare av annan elektronisk utrustning att lägga upp planer för datorproduktion. Denna produktion ledde 1966 till införandet av integrerade kretsar som baselement i stället för separata transistorer.

1957 tillkom två viktiga samarbetsorganisationer: Electronic Industry Council, som har till uppgift att råda handels- och industriministeriet och därvid har formulerat en Promotion Act med explicit stöd åt dels forskning och utveckling och dels modernisering av produktionsmetoder. Härjämte bildades samma år the Japan Electronic Industry Development Association (JEIDA), som koncentrerar sig speciellt på datorproblem av typen

- Utveckling av programmeringsteknik
- Utbildning av systemerare och tekniker
- Marknadsundersökningar utom- och inomlands
- Standardisering
- Databehandlingstjänster.

Båda dessa organisationer har tillkommit på initiativ från den japanska regeringen, som även på en mångfald andra sätt stött den fortsatta utvecklingen inom området. Några exempel kan ges:

a) Japans tidigaste datorer (bortsett från FUJIC) har tillkommit med statsmedel.

b) Från och med 1957 har utdelats subsidier till inhemska datortillverkare i syfte att uppmuntra forsknings- och utvecklingsarbete.

c) Ett samverkande företag, baserat på de sex ledande datorföretagen, bildades 1961: Japan Electronic Computer Company. Banklån på 725 Mkr har fram till 1969 kommit detta samarbetsföretag till del.

d) Skattelättnader på 11 Mkr har medgivits datorföretagen mellan 1961 och 1967.

e) Den japanska utvecklingsbanken har 1961-65 tillhandahållit ett lån på 10 Mkr för modernisering av datorproduktionsmetoder.

f) Ett projekt för produktion av superdatorer startades 1966. Projektet berörs senare i detta avsnitt.

g) Japan Information Processing Development Center har bildats, med bl a uppgift att utveckla programvarumetoder, utbilda systemerare/programmerare, standardisera samt tillhandahålla databehandlingsservice.

Inte minst för en svensk betraktare (som kunnat se mycket litet dylik aktivitet i hemlandet) vill det förefalla naturligt att denna tunga satsning burit frukt. Den japanska datorexpansionen har blivit synnerligen kraftig. Situationen är i slutet på 60-talet den att sex japanska leverantörer, tillsammans med IBM, Japan, nära fullständigt behärskar den inhemska marknaden. Dessa sex är:

- Hitachi
- Nippon Electric
- Fujitsu
- Toshiba
- Oki
- Mitsubishi

Av dessa sex innehar de tre förstnämnda marknadspositioner före de tre senare. Det låter sig inte göras att på här tillgängligt utrymme beskriva den rika datorflora som av dessa företag utvecklats under 60-talet. Några korta kommentarer för nämnda företag och deras mot decenniets slut salufördа datorer (se tabell) kan dock ges:
(värdemässiga marknadsandelar nedan härrör från 1967)

Hitachi
(Marknadsandel ca 16 %)
Huvudsakligen medelstora datorer, sedan ca 1968 dock kompletterat med

mindre modeller. 10-årigt tekniskt samarbetsavtal med RCA tecknat 1961. Tidiga modeller RCA-influerade. HITAC 8000-serien (modeller 8200. 8300, 8400, 8500) dominerande.

Nippon Electric
(Marknadsandel ca 15 %)
Marknadsledare avseende antal installationer, huvudsakligen mindre datorer. 10-årigt tekniskt samarbetsavtal med Honeywell sedan 1962. Stark position avseende utrustning för datakommunikation. Datorserien NEAC 2200 (9 olika modeller) dominerande.

Fujitsu
(Marknadsandel ca 10 %)
Inget utländskt samarbete. Enda exporterande japanska datorföretag före 1968, export till Sovjet, Bulgarien, Filippinerna och Korea. Såväl små som större datormodeller. Datorserien FACOM 230 (8 olika modeller) dominerande.

Toshiba, Oki och Mitsubishi
(Tillsammans ca 15 % av marknaden)
Toshiba och Mitsubishi har samarbete med General Electric (en tänkbar fusion har diskuterats). Oki samarbetar med Univac. De tre företagen saluför ett rikt sortiment datormodeller, med tonvikt på den mindre till medelstora klassen.

För data angående japanska datormodeller, se tabell sid 74.

Nära nog samtliga ovan nämnda leverantörer tillhandahåller förhållandevis kompletta sortiment av yttre enheter till sina datorsystem. Merparten av periferiutrustningarna är modifikationer av importerad basutrustning, med undantag av Fujitsu, som tillverkar nära nog all utrustning självt (och därvid får extra stöd av japanska staten).

De kompilatorer som tillhandahålls är främst för Fortran IV, backat upp av olika egna språk. Noteras kan dock att Fujitsu FACOM 222, Hitachi HITAC 5020/5020 E och Toshiba TOSBAC 3400 kan leverera Algol-kompilator, samt Fujitsu FACOM 222, Mitsubishi MELCOM 3100 och Toshiba TOSBAC 4300 tillhandahåller COBOL. Fujitsu FACOM 230-45 samt 230-60 anges kunna levereras med PL/I-kompilator. Dessa angivelser om begränsad språktillgänglighet, avseende de generella språken, härrör från 1968, och har sannolikt kompletterats under 1969.

Ovan angivna värdemässiga marknadsandelar summerar sig till drygt ca 55 % av marknaden. Den övriga knappa hälften utgöres av importerad

Några data om japanska datorers maskinvara

Datormodell	Typisk månads-hyra kr	Levere-rad sedan	Cykeltid (mikrosek)	Addtid (mikrosek)	Ord-längd	Primär-minnes-storlek	Avbrotts-system	Minnes-skydd	Index-register	Flytande räkning
Hitachi										
HITAC 8210	15 000	sept-68	1.4 per byte	40 per 40 bitar	1 byte	8-32 Kb	Ja	-	-	-
HITAC 5020 E	200 000	okt-66	1.5 (över-lappn. via skiktminne)	1.6	32 bi-tar	16-256 Kord	Ja	Ja	7	Ja
Nippon Electric										
NEAC 2200-50	10 000	juni-67	2	63 per 5 bytes	1 byte	4-16 Kb	Ja	-	6	-
NEAC 2200-700	210 000	okänd	0.5 per 8 bytes	0.5 per 48 bitar	1 byte	128-2048 Kb	Ja	Ja	15	Ja
Fujitsu										
FACOM 230-10	6 000	dec-65	2	variabel	1 byte	4-8 Kb	Ja	-	-	-
FACOM 230-60	150 000	sept-68	0.92	1.15	36 bi-tar	32-256 Kord (yttre kärn-minne till-gängligt)	Ja	Ja	7	Ja
Mitsubishi										
MELCOM 3100-10	24 000	sept-66	1.75	35 per 5 bytes	1 byte	24-96 Kb	Ja	Ja	okänt	Ja
MELCOM 9100-30	30 000	mars-68	0.8	1.4	16 bi-tar	4-64 Kord	Ja	Ja	12	Ja
Oki										
OUK 9200	6 000	juli-67	1.2 per byte	151	1 byte	8-16 Kb	-Ja	-	8	-
OUK 9400	62 000	april-69	0.3 per byte	30	1 byte	24-128 Kb	Ja	Ja	32	Ja
Toshiba										
TOSBAC 1500	5 000	dec-68	2.8	42	1 byte	4 Kb	-	-	3	-
TOSBAC 3400-41	97 000	mars-68	0.8	2.7	24 bi-tar	96 Kord	-	-	3	Ja

eller i Japan framställd utländsk utrustning. Härav dominerar IBM stort, med egen tillverkning i Tokyo. En kraftfull statlig dämpning av anskaffande av icke-japansk utrustning kan noteras, vilket bl a har till följd att Japan, tillsammans med England (ICL-koncernen), är de enda områden där IBM's marknadsdel befinner sig kring "blott" 40 % (± 5 %). Utländsk investering i främmande datortillverkning inom Japan motarbetas effektivt från den japanska statens sida.

En intressant notering innebär att ca 80 % av datorer i Japan hyrs i stället för är köpta. Denna siffra är sannolikt klart högre än i Europa och USA.

Antal installerade datorer i Japan
(Siffrorna rörande 1968 osäkra, och sannolikt i underkant.)

År (Fiskalt, slut 30 mars)	Antal datorer
1958	3
1959	8
1960	26
1961	66
1962	119
1963	228
1964	485
1965	562
1966	636
1967	845
1968	ca 850
1 april till 1 okt. 68	ca 350
Summa	ca 4 170

Den japanska medvetenheten rörande datorutvecklingen i USA och Europa har även medfört att man snabbt insåg de kommande möjligheterna för mycket små datorer, s k mini-datorer. I februari 1969 presenterades Hitachi HITAC 10, omedelbart följd av Fujitsus FACOM-R. Båda dessa är försedda med 4 K 16-bits ord, och kan byggas ut till 32 Kord i moduler om 4 Kord, samt har en cykeltid nära 1,5 mikrosekunder. Fortran, såväl som speciella kalkyleringsspråk, tillhandahålls. Även i prishänseende är dessa båda konkurrenter likartade, inköpspriset ligger kring 55 000 kr för basutrustningen. Ett varierande

antal perifera enheter kan anslutas. Uppgifter om time-sharing-system har dock ännu ej blivit tillgängliga. Indikationer pekar mot att samtliga de övriga fyra japanska storleverantörerna arbetar på liknande minisystem, och export är säkerligen att vänta rörande denna expanderande marknad, som inte är fullt lika servicekänslig som större datormodeller.

Även i andra änden av spektrum har kraftfull aktivitet startats av japanerna. Ett nationellt forsknings- och utvecklingsprogram fastslogs 1966 med syfte:

> "to systematically and efficiently carry out, under state financial aid and close cooperation between industrial and academic circles, research and development of such large-scale industrial technologies of great economic significance and pressing urgency, that require enormous investments and long periods for the purpose and moreover involve too great a risk to be borne by private enterprises".
> (Computers in Japan 1969; Japan Electronic Industry Development Association)

För detta program har fem projekt startats, varav ett rör datorframställning. För detta projekt har till att börja med, åren 1966-69, avsatts 90 Mkr av statsmedel, vilket innebär att det utgör det största av de fem projekten. Den datormodell, som skall produceras, är inte avsedd att ligga efter västliga stordatorer i prestandaavseende, vilket framgår av följande jämförelsetabell:

Jämförelse mellan det japanska storprojektet och amerikanska stordatorer

	Japanska projektet	IBM 360/85	Burroughs B 8500	Control Data 6600
Leverans	mars 1972	1970	1967	aug 1964
Add-tid (nanosek)	50	80	200	300
Logiska element	LSI	IC	IC	TR
Primärminnes-utrymme	2-8 Megabytes	16-32 Kb (+ yttre primär-minne)	16-256 Kord	32-128 Kord
Cykeltid (nanosek)				
(Inre primärminne)	80	80		
(Yttre -"-)	700	1 000	500	1 000
Typiskt hyrespris				
(kronor/månad)	1 milj		875 000	625 000
Varierande mellan		550 000 - 1,1 milj	500 000 - 2,5 milj	310 000 - 775 000

Om den japanska stordatorn kan vidare anges:

- Den skall vara försedd med ett massminne på 1 000 milj karaktärer.

- In- och utmatningsutrustningen skall inkludera: karaktärigenkännande apparatur, grafisk manipulering, bildskärmar med kinesiska tecken, vokal inmatning (japanska såväl som engelska), samt snabbare konventionell utrustning.

- Programvarorna skall innefatta: operativsystem för bl a multipel access, FORTRAN, ALGOL, COBOL, PL/I etc.

Detta projekt utföres, under styrning från ett centralt råd, av samtliga större japanska datorleverantörer, bland vilka sammanslagningen Japan Software Co. Ltd (Hitachi, Nippon Electric och Fujitsu) kan noteras. Hur marknadsföringen av produkten skall organiseras synes tills vidare i detalj ej vara fullt specificerat. Resultatet från projektet kan väntas bli mycket intressant.

Det kan spekuleras över möjligheterna för japanerna att konkurrera m hj a sin omfattande datorutrustning på västmarknaden. Kortfattat bör härom kunna noteras att substansiellt samarbete med marknadsförings- och serviceorganisationer på försäljningsorterna synes som en förutsättning. Japanska mekaniska och elektroniska produkter har redan gjort inbrytningar på många andra områden, huvudsakligen baserade på lågprispolitik (härrörande från låga arbetarlöner i Japan). Det är inte uteslutet att även datorer på sikt kommer att kunna utgöra en värdefull del av den japanska exporten. Senare delen av 1970-talet kommer dock sannolikt att vara tidigaste tidpunkt för dylik verksamhet. Åtminstone separat salufördа japanska programvaror kommer troligen tidigt att visa sig av exportvärde.

1.8 Det internationella läget

Parallellt med i detta kapitel relaterade företagssträvanden avseende datorproduktion kan några noteringar göras kring internationella utvecklingslinjer.

Först kan konstateras att produktionen av datorer är en typisk halvfabrikatsbaserad bransch. De komponenter som utgör inre byggstenar i systemen produceras sällan av datorleverantörerna själva. Liksom mången övrig elektronisk kapitalvaruindustri har datortillverkarna för det mesta kommit att inse att det knappast är lönande att utöva forsknings- och utvecklingsarbete inom t ex halvledarteknik. Transistortillverkning, framställning av integrerade element (IC, Integrated Circuits, LSI, Large Scale Integration) kräver för bästa lönsamhet produktion i mycket stor skala. De flesta datorleverantörer är inte i behov av så stora mängder grundelement. Man köper därför grundelementen från, i många fall gemensamma producenter av elektronisk basutrustning.

Detta ospecificerade "samarbete under ytan" gäller för många, mindre som större, datorleverantörer även färdig periferiutrustning. Kärn-, trum- och skivminnen, radskrivare, kortläsare, hålremsläsare, bandstationer etc modifieras ofta endast måttligt av den slutgiltiga systemleverantören. Det egna arbetet sträcker sig från framarbetande av egen styrutrustning för enheterna för att passa det avsedda systemet till enbart ändrande av fabrikantbeteckning utanpå plåtskåpen. Det är uppenbart att få leverantörer har möjlighet göra egna mera fullständiga satsningar. Som ett exempel kan nämnas att konstruktion och lansering av det IBM-egna 360-systemet har sagts innebära en satsning motsvarande

storleksordningen 25 000 Mkr, ungefärligt ekvivalent med utvecklingskostnaderna för atombomben (10 000 Mkr i 1945 års penningvärde). Vem har råd med dylikt? Vi bör sålunda anse det naturligt att, om än i väsentligt mindre skala än ovan nämnt, ett utbyte av halvprodukter har kommit till stånd, i syfte att bl a begränsa erforderliga investeringar för datorproduktion.

Ett vitt nät av licens- och patentutväxling existerar. USA spelar här en fundamental roll. För närvarande finner vi blott två kraftfulla leverantörer utanför USA som i huvudsak arbetar utan amerikanska samarbetsavtal och underleveranser: ICL i England och Fujitsu i Japan.

En modern valuta för dylikt samarbete är tekniska patent. Ju tyngre en firmas forsknings- och utvecklingspatent kan anses vara, desto större har möjligheten blivit att med dessas hjälp nå gynnsam utväxling av know-how och licenser från andra firmor. Patent tas sålunda numera ej sällan ut i synbarligt senare rent utväxlingssyfte.

En mindre nackdel för dessa samarbetsavtal med sig. Det komplicerade nätet av leveranser länder emellan försvårar framtagandet av rättvisande statistik över nationella datorsituationer. Flera leverantörer motsätter sig dessutom principiellt att sprida information om antal installerade datorsystem. Dessa skäl ligger bakom frånvaron i detta kapitel av statistiskt tabellmaterial rörande datorspridning. Den information som ges i facktidskrifter o d utgörs ofta av grova skattningar. Vi har i huvudsak underlåtit att vidarebefordra dylika uppgifter i denna bok.

Internationell politik har haft betydelse för datorförsäljning länder emellan. Vi ser här en klar gränsdragning mellan öst och väst. Två faktorer kan ligga bakom en dylik separering:

1) Ena parten, hittills huvudsakligen väst, önskar inte till "motparten" sälja utrustning som med framgång må kunna användas i militärt syfte.

2) Andra parten, hittills huvudsakligen öst, önskar inte bidraga till "motpartens" ekonomiska vinst genom att placera datororder där.

Bägge dessa faktorer kan väntas luckras upp successivt, även om detta kan ta tid. Handelsutbyte länder emellan, även från olika politiska block, ger stora fördelar för ländernas utveckling, vilket avspeglas av det under 50- och 60-talen ökande allmänna internationella industriella utbytet. Hittills har emellertid östländer föredragit att köpa datorutrustning från icke-USA-beroende leverantörer. Sålunda har ICL i England under 60-talet enligt uppgift fått leverera utrustning till ett värde av nära en kvarts

miljard kr till Östeuropa (med ICL 1905 F som största installation), medan USA-leverantörer, t ex IBM, registrerat mycket måttlig försäljning där. Exportkontroll från USA har lett till att i Östeuropa endast äldre och högst medelstora amerikanska datorer installerats. 1969 finns en IBM 7090 i Belgrad, en CDC 3300 är på order till Tjeckoslovakien (möjligen installerad), men härförutom finner vi av amerikanska datorer huvudsakligen formatet IBM 1401-1620 representerade. Sveriges neutrala politiska hållning har hjälpt Data SAAB att placera order i bl a Tjeckoslovakien. Den enda datorleverantör i Japan som inte har samarbetsavtal med USA, Fujitsu, har kunnat sälja till bl a Bulgarien. De kinesiska tecken som skall kunna frammatas på bildskärmar till det gemensamma japanska superdatorprojektet har säkert syfte till försäljning i Kina. Denna japanska stordator framtages i huvudsak utan amerikanskt beroende, ens som underleverantör. För detta projekt svarar, som noterats i avsnittet om den japanska utvecklingen ovan, en samarbetsorganisation med samtliga större japanska datorindustrier som deltagare, stödd från regeringshåll.

I detta sammanhang kan nämnas ett intressant samarbetsprojekt inom östblocket som nyligen startats. Projektet, som går under benämningen Series RJAD, Joint Data Processing Program, rör framställning av en datorserie som uppges skall bli kompatibel med IBM System/360. 80 000 personer och 6 länder skall medverka. Sovjetunionen skall producera processorsystemet, Östtyskland och Tjeckoslovakien står för den perifera utrustningen, Bulgarien och Rumänien utvecklar maskinvarukomponenterna (integrerade kretsar) samt Ungern har fått det initiala ansvaret för programvarorna. - En dylik decentralisering är kanske aningen väl långt driven, med tänkbara samarbetssvårigheter som följd av geografiska och andra avstånd. Projektutvecklingen blir utan tvekan mycket intressant.

Utbudet av datorer från industri världen över har bl a varit beroende av följande faktorer:

1) Statliga nationella stödåtgärder åt datorindustrin.

2) Skillnader i produktions- och marknadsföringskapacitet, konkurrens.

3) Privat finansiell uppbackning.

4) Strukturförändringar inom datorkonsumtionen.

Bland dessa förtjänar statliga stödåtgärder speciellt noteras. Inte bara genom olika former av ekonomisk uppbackning har dylika åtgärder haft stor betydelse.

Direkt medverkan i och stöd till utvecklingsprojekt, rekommendationer rörande utredningsarbete i samband med utrustningsval, initiering av nya publika tjänster i socialpolitiska sammanhang, försvarspolitiska åtgärder, patentpolitik, allt detta summerar sig till kraftfulla möjligheter för statligt agerande. De länder som aktivt använt sig av dylika åtgärder i samarbete med privat nationell industri har i många fall kunnat se värdefulla resultat därav. Vi bör kunna utgå från att under 70- och 80-talen visst internationellt samarbete av ovanstående typ kommer till stånd. De nationella erfarenheterna talar härför.

Avslutning

I detta kapitel har endast den digitala utvecklingen behandlats. Analoga datorer, av intresse för många tillämpningar, har oftast konstruerats för specialiserade ändamål, och har därför knappast lika generellt intresse. Marknaden för analoga datorer är mångskiftande, men med den successivt ökade flexibiliteten hos framför allt 60-talets digitala datorer har de analogt arbetande kommit något i skymundan. Det är därför avsiktligt som de nära fullständigt utelämnats i denna kortfattade kommenterade historiebeskrivning.

Ej heller har utvecklingen av perifera datorenheter givits stor plats. Detta har huvudsakligen varit av utrymmesskäl, målsättningen för kapitlet har varit att ge en till omfånget måttlig kommentar till maskinvaruutvecklingen. Många intressanta perifera enheter hade kunnat beröras, men därmed hade utrymmesramen sprängts.

Helhetsmässigt kan vi konstatera att rörande den totala datorutvecklingen de senaste tre decennierna en dominant position har intagits av de amerikanska universitetens utvecklingsarbete. Många av de nykonstruktioner som ovan berörts emanerar från nytänkande inom universitetsmiljö. Detta förhållande kan noteras med visst intresse, enär datorutveckling härmed på ett näraliggande sätt knutit samman universitet och industri. Det förefaller klart att utvecklingsbefrämjande samarbete sålunda kan realiseras mellan akademiker och industri/näringsliv. Ett från universitetshåll stundtals icke-ekonomiskt tänkande har i datorsammanhang visat sig fruktbart kunna konverteras till exploatering i strikt ekonomisk industrimiljö. Emedan universitet nära undantagslöst drivs i statlig regi får ett statligt intresse för dylikt samarbete förutsättas för erhållande av goda resultat.

Det är författarnas förhoppning att 70-talets datorutveckling i Sverige skall kunna verifiera fördelar för dylikt samarbete.

Fig 1.8:1. Hur en människa kan "lura" maskinen illustreras på denna teckning. Överst t.v. förlorar mannen mot maskinen. T.h. ändrar mannen maskinens instruktioner. Nederst vinner mannen.

(Ur: "A Chess-Playing Machine" by C.E. Shannon, Scientific American, februari 1950. Med tillstånd av Scientific American, Inc.)

LITTERATUR

1. Naur, P., Datamaskinerna och samhället, Studentlitt 1969

2. Rosen, S., Electronic Computers: A Historical Survey, ACM Computing Surveys mars 1969

3. Rosen, S., Programming Systems and Languages, A Historical Survey McGraw-Hill 1967

4. Meyer, R., Utvecklingen av datamaskintekniken i Sovjet ... Tekniskt Vetenskaplig Forskning (IVA) 1967:7

5. Nyberg, P.R., Computer Technology in Communist China, Datamation febr. 1968

6. Jéquier, N., Technological gaps in the computer industry; The OECD, Observer juni 1969

7. Hibben, S.G., Soviet computer reliability, Datamation aug. 1967

8. Holland, W.B., The BESM-6 computer, Datamation aug. 1967

9. Freeman, C., Research and development in electronic capital goods. National Institute Economic Review nov. 1965

10. Politikens förlag: Bogen om EDB 1969

11. Sigurdson, J., Kinas dataindustri ... Modern Datateknik nr 10 1969

12. JEIDA, Computers in Japan 1969. Japan Electronic Industry Development Association

13. Dempa Publications: Computer makers active in developing "Mini-Computers". Japan Electronic Engineering april 1969.

14. Levine, G.B., Computers in Japan, Datamation dec. 1967

15. EAJ, Electronics in Japan 1968-69. The Electronics Association of Japan.

16. Voysey, H., Trends and tangents of east-west trade. Datamation sept. 1969

17. Holland, W.B., Sovjet computing 1969. Datamation sept. 1969.

2. PROGRAMVARUUTVECKLINGEN

2.1 Språkutvecklingen

Den maskinnära tiden

Maskinkodning

De första åren efter uppträdandet av datorer med program och data lagrade i samma minne erbjöd inte annan möjlighet till kommunikation mellan människa och maskin än direkt via maskinens eget språk, dvs i de sifferkoder som direkt tolkas och utförs i datorns processenhet. En följd av instruktioner utformade på detta sätt kallas som bekant program skrivet i maskinkod eller maskinspråk (eller objektkod).

Det låter sig inte göras att exakt placera in maskinkodning i ett större tidsperspektiv, då - som framgått av föregående kapitelavsnitt - olika maskiner under den första datordekaden tillkommit vid helt skilda tidpunkter. Vi talar emellertid, grovt sett, om åren i början av 1950-talet. Var och en av denna tids maskiner programmerades sålunda under sina första år endast direkt i maskinkod, medan detta för olika maskiner innebar olika tidpunkter.

Den typ av maskinkodning, som kan sägas befinna sig allra närmast maskinen, utgjordes av sladdprogrammering. Detta innebar att man på fasta eller utbytbara "boxar", hålförsedda tavlor t ex av storleksordningen 1 m^2, anbringade elektriska sladdar för att förbinda hålen elektriskt. Inom en dator kunde man med sladdbyten eller med hjälp av anbringande av olika sådana boxar utvälja den mängd av operationer som skulle utföras. Att byta sladdanslutningar var naturligtvis arbete närmast för ingenjörer, men var en tvingande nödvändighet för programmeringsverksamhet den första tiden.

Successivt sökte man göra programmeringen mera flexibel och därmed kom hålremsa och hålkort in i bilden. Den s k IBM Card Programmed Calculator (CPC), utvecklad de sista åren av 40-talet, använde sig av sladdprogrammering vid tolkning av de instruktioner som matades in, en per hålkort. Genom knepigt anbringande av sladdarna kunde man få

maskinen att "förefalla" som en maskin med tre-adress-logik och med standardiserad vokabulär av inbyggda rutiner för kvadratrot, sinus, exponentialfunktion osv. Erfarenheterna från programmerings-rationaliseringsförsök med denna maskin kom att bli av stort värde för den senare utvecklingen av mera sofistikerade programmeringssystem.

Det kan noteras att ännu sent på 60-talet ett antal sladdprogrammerade datorer var i drift; på grund av den begränsade flexibiliteten huvudsakligen använda för specialiserade tillämpningar som t ex in/utmatningsdatorer till större datorer. Exempel på dylika datorer är UNIVAC 1004.

En utvecklingsmässigt viktig maskin var den vid MIT konstruerade Whirlwind. Denna internt snabba maskin kunde vid denna "numeriska" tid tyvärr endast komma till användning för ett begränsat antal tillämpningar på grund av dess korta ordlängd - 16 bitar, vilket i många fall krävde programmering med multipelprecisionsteknik, samt dess begränsade primärminnesutrymme. Svårigheten att maskinprogrammera Whirlwind blev även en pådrivande faktor för den vidare programvaruutvecklingen.

Mnemoteknisk kod kommer till användning

Emedan människan (av miljöpåverkan) naturligen arbetar lättare med bokstavs- än sifferkombinationer (i basen 10, för närvarande) är maskinkodning på alfanumeriskt anpassade maskiner vänligare och enklare än på binära. Univac I var en maskin som redan från början hade framarbetats med tanke på lättkodbarhet. Dess instruktioner utgjordes av bokstavskombinationer som var sammansatta som stöd för minnet. Denna mnemoteknik kom att vara mycket värdefull, och nära nog en dämpande faktor för den fortsatta språkutveckling som under hand startades på denna maskin under ledning av dr Grace Hopper.

Många användare föreföll enligt uppgift vara i stort sett nöjda med denna dators maskinkodning, ovetande om kommande tiders språkutveckling.

På flera binära maskiner infördes förenklande benämningar på frekventa maskininstruktioner. Enkla hårdvaruhjälpmedel "filtrerade" vid programinläsningen dessa instruktionskoder, varvid de gavs sin ur maskinens synpunkt rätta innebörd. Vi kan här märka de allra första stegen mot mera generell interpretering. Detta tolkande arbete utfördes till att börja med direkt och successivt under inläsningens gång, vilket kan sägas vara en något mindre effektiv arbetsprincip ur maskinutnyttjningssynpunkt. Detta bör dock mer än väl ha uppvägts av en totalt sett mer effektiv programproduktion.

Subrutiner

Tämligen tidigt blev det uppenbart att mycket onödigt arbete behövde nedläggas om inte någon form av rationalisering kunde införas för de allra vanligaste instruktionssekvenserna. Att koda om dessa gång på gång för varje användningstillfälle var knappast ett effektivt sätt att utnyttja programmerarnas kapacitet.

Införandet av subrutinbegreppet blev det första avgörande steget framåt. Till att börja med var de använda maskinkodade subrutinerna synnerligen inflexibla, och helt utan styrbarhet via t ex från programmet tillhandahållna parametrar eller andra specifikationer. Programgenomlöpandet innebar hopp i primärminnet mellan olika permanent lagrade instruktionssekvenser, för flytande räkning, matematiska funktioner och liknande. Denna strukturering bringade synpunkter kring minnesekonomi ytterligare i förgrunden, ett redan besvärligt problem med dåtidens starkt begränsade utrymmen.

De första automatkodningssystemen

Redan i början av 1950-talet väcktes intresset för möjlighet till så flexibel hantering som möjligt av mnemotekniskt utformad kod samt subrutiner. Det tidiga utvecklingsarbetet för de initiala automatkodningssystemen var i huvudsak forskningsresultat från Univac och IBM, och det dröjde något innan de kunde föras ut på den kommersiella marknaden.

Dr Grace Hoppers arbeten vid Univac kom att få stor betydelse. Under henne utvecklades före 1954 en hel serie "språk", huvudsakligen starkt beräkningsbundna. Bland dessa kan märkas A2, ett översättningsprogram med vars hjälp tillhandahölls ett tre-adressigt flytande-räknings-system genom sammansättning av anrop på permanentlagrade subrutiner för flytande räkning. Det jämfört härmed utvidgade AT3 (Math-Matic) kom att ge värdefulla bidrag till det i Europa senare startade utvecklingsarbetet för Algol och andra högre språk, men AT3's användbarhet begränsades av det faktum att Univac I hade förlorat sin konkurrenskraft på marknaden innan språket blev helt färdigt.

Math-Matic lanserade som en av flera intressanta nyheter automatisk segmentering. Ett program som vid kompileringen visade sig för stort för tillgängligt primärminne, segmenterades av kompilatorn i mindre delar. Kompilatorn genererade också instruktioner som styrde framhämtning av aktuella segment under exekveringen. Detta blev tämligen ineffektivt ur maskinutnyttjandesynpunkt, då kompilatorn inte kunde känna till program-

CONTROL SENTENCES

(n) CONTAIN $X(m,n)$.
(n) CONTAIN $X(m,n,p)$.

(n) EXECUTE SENTENCE F .
(n) EXECUTE SENTENCES F THRU L .

(n) IF $X > Y$ JUMP TO SENTENCE F .
(n) IF $X < Y$ JUMP TO SENTENCE F .
(n) IF $X = Y$ JUMP TO SENTENCE F .
(n) IF $X > = Y$ JUMP TO SENTENCE F IF $V < W$ JUMP TO SENTENCE G .
(n) IF $X = < Y$ JUMP TO SENTENCE F IF $V = W$ JUMP
TO SENTENCE G IF $P > Q$ JUMP TO SENTENCE H .

(n) IGNORE .

(n) JUMP TO SENTENCE F .

(n) PRINT-OUT A B C . . . N .

(n) SET TO *number* A B C .

(n) STOP .

(n) TYPE-IN A B C . . . N .

(n) VARY X X_0 (X_i) X_L SENTENCES F THRU L .
(n) VARY X X_0 (X_i) X_L Y Y_0 (Y_i) Y_L SENTENCES F THRU L .
(n) VARY X X_0 (X_i) X_L Y Y_0 (Y_i) Y_L Z Z_0 (Z_i) Z_L SENTENCES F THRU L .

(n) VARY X X_0 X_1 X_2 . . . X_n SENTENCES F THRU L .
(n) VARY X Y X X_0 Y_0 Z_0 X_1 Y_1 Z_1 X_2 Y_2 Z_2 . . . X_n Y_n Z_n SENTENCES F THRU L .

INPUT/OUTPUT SENTENCES

(n) WRITE-LABEL X . . . X FOR SENTENCE F .

(n) TITLE FOR SENTENCE F X . . . X .
(n) TITLE FOR SENTENCE F X . . . X HEADINGS A . . . A B . . . B C . . . C .
(n) HEADINGS FOR SENTENCE F A . . . A B . . . B C . . . C .

(n) CHECK-LABEL X . . . X FOR SENTENCE F .

(n) CHECK-COUNT SENTENCE F IF EXCEED X . . . X JUMP TO SENTENCE L .

(n) READ A B C .
(n) READ-ITEM $X(m,p)$ LABEL X . . . X .
(n) READ A B C IF SENTINEL RESET AND JUMP TO SENTENCE F LABEL X . . . X .
(n) READ A B C IF SENTINEL REWIND AND JUMP TO SENTENCE F LABEL X . . . X .

(n) PRE-READ A B C .

(n) READ-ARRAY $X(I,J)$.

(n) WRITE A B C .
(n) WRITE-ITEM $X(m,n,p)$.
(n) WRITE EDIT X Y Z .
(n) WRITE-ITEM EDIT $X(m,p)$.
(n) WRITE CONVERT TO n DECIMAL X Y Z .
(n) WRITE-ITEM CONVERT TO n DECIMALS $A(m,p)$.
(n) WRITE CONVERT X Y Z .

(n) WRITE-ARRAY CONVERT TO n DECIMALS $X(m,n,p)$.

(n) CLOSE-INPUT SENTENCE F .
(n) CLOSE-INPUT AND REWIND SENTENCE F .

(n) CLOSE-OUTPUT SENTENCE F .

Fig 2.1:1. Exempel på instruktioner i Math-Matic. (n) står för instruktionsnummer.

strukturen i detalj. Vid kompileringsdags noterades emellertid loopar, så att segmentering undveks som ledde till uppdelning av dylika. Denna intressanta egenskap, tillsammans med andra, bl a accepterande av programdelar i maskinkod, innebar att Math-Matic får räknas som ett av de mera utvecklingsintressanta av de tidiga språken.

Arbetet med det mera för administrativa problem avsedda språket B0 (Flow-Matic) blev senare av värde för utvecklingen av Cobol. De första sorteringsgeneratorerna producerades även vid Univac så tidigt som 1951, tillsammans med ett två år senare färdigställt program för symbolisk derivering av matematiska funktioner, troligen historiens första program för symbolmanipulation.

En annan och tämligen oberoende grupp vid Univac intresserade sig på ett framgångsrikt sätt för minnesstruktureringsproblem i samband med programmering av Univac II-maskinen. Gruppen leddes av A. Holt och W. Turanski, under vilka programvarubegreppet GP (Generalized Programming) presenterades. Detta system förutsatte existensen av ett generellt subrutinbibliotek. Ett program som behandlades hade utseendet av en generator, sålunda endast bestående av parametrar och andra specifikationer till det mycket generella subrutinbiblioteket. Vart användarprogram var avsett att successivt införlivas med biblioteket, så att så småningom en lämplig rutinuppsättning erhölls. Subrutinerna i biblioteket var hierarkiskt organiserade, varvid en subrutin på en viss nivå kunde utökas med och anropa rutiner på lägre nivåer.

Om denna intressanta idé kan sägas att den, med subrutinbiblioteket på ett sekundärminne, innebar hushållande med primärminne. Även om systemet svårligen kan tänkas ha kommit till särdeles generell användning, utan snarare för mera specialiserade tillämpningar, bör betänkas att primärminnesbrist var ett avgörande problem "redan" för dåtidens maskiner, vilket därmed utan tvekan motiverar experiment som detta. Erfarenheter av stort värde kring programstruktureringsproblem, segmentering och minnesallokering erhölls. Det kanske viktigaste med detta arbete kan sägas vara den tyngdpunkt som placerades på programmeringssystemet snarare än på programmeringsspråket. I detta avseende kan GP sägas ha bidragit starkt.

IBM lanserade till sin modell 701, den första kommersiellt salufördа större binära datorn, bl a programmeringsfacilitet i form av vad man kallade Speedcode, som komplement till ren maskinkodning. Speedcode använde sig av subrutiner för bl a flytande räkning och indexregisterhantering. Speedcode, vid sidan av andra språk från denna tid, demonstrerade i vilken utsträckning användarna var villiga att uppge maskinbearbetningshastighet till förmån för programmeringsbekvämlighet.

PACT-systemet, även det utvecklat för IBM 701, kan sägas vara det första programvarusystem som utvecklades av en kommitté av maskinanvändare. Tyvärr kunde det färdigställas först sedan maskinen blivit omodern, ett förhållande som sedan upprepats vid flera tillfällen i datorhistorien. PACT fick dock inflytande senare i samband med programvaror som utvecklades under SHARE-organisationen (bestående av IBM-användare).

Kodning av de trumorienterade datorerna

Kring mitten av 50-talet hade de medelstora trum-datorerna nått stor spridning. Den mest populära av dessa var IBM 650, en maskin som var förhållandevis enkel att maskinkoda. Av detta skäl tillkom en mångfald tillämpningsprogram, särskilt av administrativ typ, utan hjälp av mera avancerade programtekniska system. Småningom tillhandahölls emellertid ett interpretativt system för flytande räkning, fleradresslogik, automatisk varvräkning och matematiska subrutiner, vilka utvecklats vid Bell Telephone Laboratories. Detta utgjorde en logisk fortsättning från CPC-boxar och 701-Speedcode. Effektiviteten hos 650-programsystemet var därmed bättre än hos föregångarna.

Allteftersom kompilerande system började utvecklas minskade intresset för de mindre effektiva interpretativa systemen. Först vid 60-talets slut, tack vare bl a förskjutningen från maskinvarueffektivitet i samband med hårdvarumässigt mikroporgrammerade maskinmoduler, har interpretativ teknik åter börjat användas i ökad omfattning.

För att utnyttja den trumorienterade IBM 650 effektivt krävdes programoptimering. Detta skedde genom en ur rotationssynpunkt lämplig utplacering av instruktioner på trumman. Ett program kallat SOAB (Symbolic Optimizer and Assembly Program) kombinerade symbolisk assemblering med viss automatisk optimering.

En trumorienterad konkurrent till IBM 650 var Datatron 205, som småningom blev en Burroughs-produkt. För Datatronen tillkom ett antal interpretativa och assemblerande system, vilka emellertid till en del blev inaktuella efter det att maskinen försågs med hårdvarumässig flytande räkning. Datatronen var f ö den första kommersiellt salufördа maskinen med inbyggda indexregister och automatisk relokering av subrutiner. Detta hade tidigare måst styras av programmerarna.

En av de första Datatronerna installerades vid Purdue University, där dr A. Perlis utvecklade den kanske första algebraiska kompilatorn. Även om Datatronen tämligen framgångsrikt kunde maskinkodas, framhärdade

Perlis i sin tro på procedurorienterade språk, detta trots att maskinerna utvecklingsmässigt sett ännu inte var särskilt effektiva vid kompilering. Perlis' algebraiska kompilator skrevs sedan om för IBM 650, varvid den gavs namnet IT (Internal Translator). Denna kompilator blev tämligen populär, och tillhandahölls småningom även för andra maskiner. Den kom att få icke ringa inflytande på den kommande utvecklingen, bl a genom att den visade att ett algebraiskt språk kunde implementeras med måttlig arbetsinsats (2 manår) på en liten maskin (primärminne 2 000 ord).

Högre språk tar över

Fortran definieras

Kring 1953-54, då det inom IBM ansågs vara dags för en efterträdare till modellen 701, hade man tillräcklig erfarenhet av programvaruteknik för att i den nya modellen kunna inkorporera substantiell hårdvara för effektivisering av programvarornas användning. Datorn IBM 704 försågs bl a med både hårdvarumässig flytande räkning och indexregister.

Intresset för språk av typen Speedcode (för IBM 701) hade börjat avta. Utanför IBM hade redan procedurorienterade språk börjat användas. Inom företaget startades nu utvecklingsarbetet på ett helt nytt språk.

Vid detta, och senare, tillfällen har IBM valt att gå sin egen väg hellre än att ta initiativ till samarbete med andra leverantörer bl a i syfte att begränsa antalet språk. Denna gång gällde det Fortran, 10 år senare PL/I.

Till IBM 704 initierades 1954 under J. Backus utvecklingsarbetet på "Formula Translating System", Fortran. Efter en för sin tid kraftfull arbetsinsats, 25 manår nedlades på drygt 2 år, kunde den första Fortrankompilatorn förverkligas. Detta språk kan ur många synvinklar anses som en milstolpe i språkhistorien.

Liksom de flesta tidigare, och senare, programvarusystem blev Fortran försenat under utvecklingsarbetets gång, och var knappast funktionsdugligt vid avsedd leveranstidpunkt. Språket var för användarna strax efter mitten av 50-talet alltför nytt och ovant till sin struktur. Dessutom fanns det vid denna tidpunkt många som tvivlade på dess realiserbarhet. Genom stark uppbackning från IBM lyckades man emellertid utveckla kompilatorn till acceptabel tillförlitlighet, och kring 1957-58 hade språket fått fotfäste för en mångfald matematiskt-tekniskt orienterade tillämpningar. Trots initialsvårigheterna, men hjälpt av bristen på allvarlig konkurrens

Statement	*Normal Sequencing*
$a = b$	Next executable statement
GO TO n	Statement n
GO TO n, (n_1, n_2, . . . , n_m)	Statement last assigned
ASSIGN i TO n	Next executable statement
GO TO (n_1, n_2, . . . , n_m), i	Statement n_i
IF (a) n_1, n_2, n_3	Statement n_1, n_2, n_3 as a less than, =, or greater than 0
SENSE LIGHT i	Next executable statement
IF (SENSE LIGHT i) n_1, n_2	Statement n_1, n_2 as Sense Light i ON or OFF
IF (SENSE SWITCH i) n_1, n_2	" " " as Sense Switch i DOWN or UP
IF ACCUMULATOR OVERFLOW n_1, n_2	Statement n_1, n_2 as Accumulator Overflow trigger ON or OFF
IF QUOTIENT OVERFLOW n_1, n_2	Statement n_1, n_2 as MQ Overflow trigger ON or OFF
IF DIVIDE CHECK n_1, n_2	Statement n_1, n_2 as Divide Check trigger ON or OFF
PAUSE or PAUSE n	Next executable statement
STOP or STOP n	Terminates program
DO n $i = m_1, m_2$ or DO n $i = m_1, m_2, m_3$	Next executable statement
CONTINUE	" " "
FORMAT (*Specification*)	Not executed
READ n, *list*	Next executable statement
READ INPUT TAPE i, n, *list*	" " "
PUNCH n, *list*	" " "
PRINT n, *list*	" " "
WRITE OUTPUT TAPE i, n, *list*	" " "
READ TAPE i, *list*	" " "
READ DRUM i, j, *list*	" " "
WRITE TAPE i, *list*	" " "
WRITE DRUM i, j, *list*	" " "
END FILE i	" " "
REWIND i	" " "
BACKSPACE i	" " "
DIMENSION v, v, v, . . .	Not executed
EQUIVALENCE (a,b,c, . . .), (d,e,f, . . .), . . .	" "
FREQUENCY n (i,j, . . .), m(k,l, . . .), . . .	" "

Fig 2.1:2. Exempel på satser i Fortran I för IBM 704.

från andra mer avancerade språk, växte Fortran-entusiasmen snart till omfattande dimensioner. Detta bidrog till att ytterligare stärka IBM´s ställning på marknaden.

Vidareutvecklingen av Fortran

Vid tiden strax efter mitten av 50-talet bedömdes en kompilator nära nog enbart av effektiviteten hos den maskinkod som genererades. "Man kompilerar bara en gång, men exekverar programmet många gåner" var en vanlig utsaga. Kompilatorerna konstruerades för att ta "erforderlig" tid för kompilering, med sikte på effektivast möjliga objektkod. Fortrankompilatorn på IBM 704 kunde visa upp en invecklad arbetsmässig teknik, med sikte på att generera minst lika god kod som en god programmerare direkt kunnat skriva i maskinspråk. För att effektivt använda de nya indexregistren på 704-an fanns en komplicerad algoritm inbakad, som innefattade en komplett analys av programflödet samt en simulering av (det ännu inte genererade) programmets exekvering. Detta var givetvis tidskonsumerande. Som ett exempel kan nämnas att för att översätta 50 Fortran-satser till ca 1 000 maskininstruktioner behövde IBM 704 sex minuter.

Objektkodsoptimeringen reste debatt tidigt. Väsentliga avsnitt överfördes dock till Fortran II, och är fortfarande i drift på vissa existerande 7090-system.

Erfarenheter visade småningom att korta kompileringstider var av värde i många sammanhang. Under felsökningen i ett program kompilerades det många gånger, medan blott engångs-exekveringar inte var ovanliga. I alla händelser insågs att den tidigaste tyngdpunkten på enbart korta exekveringstider borde förskjutas något.

Fortran II, som tillkom 1958, blev mycket utbrett. Denna version tillät sammanlänkning av program som en mängd subrutiner, styrt av ett gemensamt huvudprogram. Subrutinerna kunde kompileras och utprovas separat. Denna segmenteringsprincip, med impulser från Univac´s Generalized Programming-system, visade sig värdefull bl a därför att den hushållade med programmerarnas kapacitet. Många 704-installationer började koncentrera sig helt på Fortran II. Konkurrentleverantörerna påbörjade arbetet på egna Fortran-kompilatorer.

Efter framgångarna för Algol har i senare Fortran-versioner införts några av Algol´s generellare tankegångar, av typen villkorliga och logiska satser. Den internt inom IBM använda versionen Fortran III lan-

serades inte utanför företaget. Fortran III innefattade ett antal utvidgningar från Fortran II, bland vilka vi här endast nämner en: möjligheten att inkludera maskininstruktioner in-line i Fortran-programmet. Denna facilitet nådde inte Fortran IV, då den ansågs fullständigt spoliera kompatibilitet mellan olika datorer, även mellan IBM 704 och IBM 709.

Den senaste versionen, Fortran IV, har nått en utomordentligt stark position såväl på den amerikanska marknaden som annorstädes. Inget språk (möjligen med undantag av Cobol) kan ännu hösten 1969 sägas direkt hota detsamma. - Av många skäl, främst ekonomiska, är en konservatism märkbar i språksammanhang. För de tillämpningar där driftsekonomi och kompatibilitet är avgörande, torde en sådan konservatism vara befogad.

Assemblysystemen utvecklas

Under tidsrymden 1955-65 kan noteras en successiv och förhållandevis jämn utveckling av de maskinorienterade språken (assembly-språk). Av naturliga skäl har standardisering ej kunnat göra sig gällande beträffande dessa. De avser ge programmeraren möjlighet att på största detaljnivå (näst maskinkod) beskriva de operationer som skall utföras. Då standardisering av maskinvaror ej låter sig enkelt genomföras, bl a av marknadsföringsmässiga skäl, avviker de flesta leverantörers maskinorienterade språk från varandra.

Från början avsågs med kodning i ett maskinorienterat språk möjligheten att skriva en följd av symboliska instruktioner som vid assemblering översattes, entydigt, till instruktioner i ren maskinkod (binärt eller decimalt). Varje assemblyinstruktion hade samma struktur som maskininstruktionen den vid översättningen genererade. Ett-till-ett-förhållande rådde således mellan assemblyinstruktion och maskininstruktion. Successivt har emellertid denna definition mjukats upp. Framför allt noterar vi införandet av makroinstruktioner och möjlighet till generativ kodning. En makroinstruktion kan på assemblynivå liknas vid en subrutin/procedur hos ett problemorienterat språk. Anrop av en (globalt deklarerad) makroinstruktion innebär att ett antal maskininstruktioner genereras och infogas på platsen för anropet (motsvarande). Genereringen styrs av parametrar från anropet. Ett okritiskt användande av makroinstruktioner kan i vissa fall resultera i ett onödigt slöseri med primärminnesutrymme. Besparingar av programmerararbete uppnås emellertid oftast.

De flesta av dagens datorsystem tillhandahåller omfattande bibliotek av makroinstruktioner. Dessa instruktioner används framför allt hos program-

generatorer och för kommunikation mellan (det i detta sammanhang assemblykodade) applikationsprogrammet och olika operativsystemsfunktioner. I detta senare sammanhang syftas t ex på begäran om minnestilldelning, in/utmatningsoperationer, programadministration o d.

Vid programmering i ett problemorienterat språk har användaren normalt ingen möjlighet att själv i detalj styra kommunikationen med olika operativsystemsfunktioner. Detta har standardmässigt överlåtits till kompilatorn. Bland annat av detta skäl har de högre språken (i renodlad form) hittills ej varit direkt användbara vid implementering av komplicerade tillämpningssystem, t ex av direktbearbetande (on-line) typ.

Sedan ca 1965 kan i stort sett utvecklingen av assemblyspråken sägas ha kulminerat. Deras flexibilitet är f n tillräcklig för att ha lett till att ren maskinkodning sedan flera år ej förekommer i tillämpningsmiljö. Detta har lett till en viss benämningsförskjutning, med "maskinkodning" avses idag oftast kodning i en maskins maskinnära språk, och ej i ren maskinkod.

Det effektivare maskinutnyttjande som p g a de maskinnära språkens konstruktion kan uppnås med dessa i förhållande till de problemorienterade har lett till att under 60-talet maskinnära kodning har kommit till användning för ett stort antal tillämpningar. Detta har ofta blivit fallet även då problemorienterade språk kunnat användas. Det har för det mesta gällt program där kort exekveringstid är avgörande, dvs för tillämpningar som avsetts köras ofta, och där förhållandevis längre erforderlig tid för programframställning (programmering, felsökning, inkörning) har bedömts vara av mindre betydelse. Detta har uppenbarligen varit och är fortfarande ett ekonomiskt avgörande. Ett annat skäl för kodning i maskinnära språk har varit leveransförseningar av adekvata kompilatorer för de högre språken.

Tillverkning av ett datorsystems centrala programvaror, dvs operativsystemet, har med få undantag utförts i de maskinnära språken. Detta på grund av en strävan mot ovan nämnda effektivare maskinutnyttjande. Ett par intressanta undantag härifrån kan dock nämnas: Burroughs, som under hela 60-talet aktivt stött språket Algol, har för sina större modeller programmerat stora delar av operativsystemet i detta språk. Deras maskiner har också i viss mån hårdvarumässigt konstruerats för effektiv kompilering (möjlighet till stack-hantering etc). För MIT´s Project MULTICS valdes redan tidigt full satsning på programmering i PL/I; detta beslöts t o m innan dylik kompilator existerade. Man började sålunda med att framställa en sådan. De främsta skälen för val av detta språk för ett så avancerat projekt har varit att det arbete som nedläggs på MULTICS skall vara så

```
            START
BEGIN       BALR    15,0
            USING   *,15
            L       6,OLDYTD        Ladda OLDYTD i reg. 5
            LR      5,6
            A       6,EARN          Addera EARN
            ST      6,NEWYTD        Lägg resultat i NEWYTD
            C       5,C4800         Jmfr reg. 5 med C4800
            BC      4,YES           Reg. 5 < C4800 hoppa till YES
            SR      7,7             Nollställ reg. 7
            BC      15,STORE        Hoppa till STORE
YES         C       6,C4800         Jmfr reg. 6 med C4800
            BC      2,OVER48        Reg. 2 > OVER48
            L       7,EARN          Ladda EARN i reg. 7
            BC      15,MULT         Hopp till MULT
OVER48      L       7,C4800         Ladda C4800 i reg. 7
            S       7,OLDYTD        Subtrahera OLDYTD
MULT        M       6,C358          Reg. 6 * C358
            A       7,HALF          Avrundning i reg. 7
            D       6,CHUN          "       " "   6
STORE       ST      7,TAX           Lägg innehåll i reg.7 i TAX
            A       7,OLDSS         Addera OLDSS
            ST      7,NEWSS         Lägg resultat i NEWSS
            SVC     0
EARN        DC      F'10276'
OLDYTD      DC      F'470000'
NEWYTD      DS      F
OLDSS       DC      F'17000'
NEWSS       DS      F
TAX         DS      F
C4800       DC      F'480000'
C358        DC      F'3625'
HALF        DC      F'50000'
CHUN        DC      F'100000'
            END     BEGIN
```

Fig 2.1:3. Exempel på program i IBM 360 Assembler.

generellt som möjligt samt kunna överföras till olika datorsystem och därmed komma till nytta även utanför projektet.

Algol

Kring åren 1956-57 intresserade sig medlemmarna i den tyska organisationen GAMM (Gesellschaft für Angewandte Mathematik und Mechanik) bl a för algebraiska maskinella översättare. Det blev småningom allt klarare vilka fördelar som skulle kunna nås om man kunde enas om ett internationellt formel-språk. Härvid kunde uppenbarligen inga framgångar nås utan amerikansk medverkan. Presidenten för GAMM inbjöd därför den amerikanska organisationen ACM (Association for Computing Machinery) att starta samarbete i syfte att söka nå fram till gemensamma språknormer. ACM, som dittills inte hade engagerat sig i språkutveckling, tog emellertid positiv ståndpunkt. Man tillsatte omedelbart en kommitté för ändamålet, med representanter för leverantörer, universitet och sådana forskningsinstitut som hade intresserat sig för dylika problem. ACM-kommitténs första uppgift blev att sammanställa en samling algebraiska regler till en grammatikliknande produkt, som skulle utgöra ett amerikanskt förslag vid det första större mötet med GAMM.

Samtidigt arbetade GAMM ut sina synpunkter. Vid ett möte föreslogs från GAMM's sida användning av vissa engelska ord, såsom begin, end, for, do och if, som standard. De vidare konfrontationerna visade att få grundläggande skillnader existerade mellan de båda arbetsgruppernas önskemål. Diskussion uppstod emellertid om hur långt ett dylikt språk borde sträcka sig, och om hur vissa avancerade begrepp skulle tacklas. Strängmanipulering, vektor- och matrisräkning, multipel precision, segmentering samt minnesutnyttjande var ämnen som ledde till debatt.

ACM-kommittén fattade ståndpunkten att det var klokt att avvakta något med standardisering av samtliga komplicerade begrepp. Trots allt var maskinvaran ännu inte mogen alltför avancerad språkteknik. Man beslöt sig för att dela upp arbetet i två delkommittéer. Den ena skulle arbeta med att söka komma fram till sådana egenskaper som kunde förmodas tillfredsställa önskemål på kort sikt, medan den andra småningom skulle specificera ett språk som innebar det mest avancerade tänkandet inom datorvärlden.

Kortsiktskommittén mötte GAMM i Zürich våren 1958, och detta möte resulterade i en Preliminary Report on an International Algebraic Language, senare känt som Algol 58. För att sprida information om och debatt kring rapporten inbjöds intresserade att kommentera den-

samma i den då nystartade periodiska utgåvan Algol Bulletin, editerad av Peter Naur i Danmark. I USA ägde debatt rum i Communications of the ACM. Som en sammanfattning av de inkomna synpunkterna avhölls ett antal möten under 1959, och en slutlig sammankomst planerades till januari 1960 i Paris.

Vid detta sistnämnda internationella möte fastslogs normerna för Algol-språket, och diverse tillägg till Algol 58 gjordes. Resultatet, numera känt som Algol 60, eller kort och gott Algol, kom att sträcka sig väsentligt längre än som förmodats vid de tidiga mötena. Begrepp som rekursiva procedurer, dynamisk minnestilldelning, blockstrukturer, s k own-variabler etc ställde höga krav på kommande kompilatorkonstruktörer. Den debatt som följde indikerade att vissa av dessa komplicerade begrepp möjligen skulle komma att skymma de mera grundläggande fördelarna i språket.

En viktig händelse var valet av publiceringsform för språkgrammatiken. Här presenterades den s k "Backus´normal form", med vars hjälp, utgående från elementär teckennivå, Algolspråkets hela flexibilitet uppbyggdes med rekursiva definitioner. Detta beskrivningssätt (meta-språk) tilldrog sig stort intresse, både på grund av sin kompakta form och sin logiska strikthet och har sedermera kommit till användning även i andra sammanhang.

Allt eftersom språket började användas framkom successivt antydningar om olämpligheter och smärre motstridigheter i definitionen. Det i april 1962 i Rom avhållna mötet korrigerade dessa, och lät publicera en "Revised Report on the Algorithmic Language Algol 60". Denna är sedan dess grunddokument för användare och kompilatorbyggare rörande Algol-60.

En begränsning för språket, som sannolikt legat till grund för det endast måttliga intresset i USA, är den kompletta frånvaron av specifikationer för in- och utmatning. Dessa är av avgörande betydelse för ett språks spridning. Det faktum att Algol från början var avsett som ett huvudsakligen rent algoritmiskt språk, sålunda med matematiskt/teknisk anknytning (där i början på 60-talet endast måttliga behov för in/utmatning förmodades), försvarar inte helt denna brist.

Den fortsatta debatten ledde till att småningom in/utmatningsproblemen fick en lösning. Under ordförandeskap av D. E. Knuth utarbetades i ACM Programming Language Committées regi under 1963-64 ett förslag till in/utmatningsprocedurer. Detta förslag, som brukar gå under benämningen "Knuth´s förslag", har accepterats av många datorleverantörer och införts i deras Algol-kompilatorer.

Algol-60 har inte fullt visat sig tillräckligt effektivt för problem rörande stora datamängder på sekundärminne. På grund av att språket saknat möjlighet tillhantering av datastrukturer vanliga för administrativa tillämpningar har det endast i undantagsfall kommit till användning i dylika sammanhang.

Algol-60 har emellertid varit föregångare i ett flertal språkliga avseenden:

- blockstrukturer
- variablers definitionsområden
- definition av formellt språk
- rekursiva procedurer

samt flera andra. Språkets betydelse för programmeringsspråkens utveckling har varit avsevärd.

Språkdilemmat

Med Fortran på väg att bli tämligen väl etablerat kring 1958 kan man fråga sig varför den amerikanska språkkommittén inte rekommenderade att det planerade internationella språket skulle utformas som en någotsånär kompatibel utvidgning av Fortran. Det fanns ett flertal skäl. Ett hade att göra med Fortrans grammatikaliska begränsningar. Några Fortranegenskaper var inflexibelt utformade på grund av den begränsade erfarenhet av kompilatorspråk som existerade när detta språk definierades. Fortran var inte konstruerat för att vara helt oberoende av egenskaper hos den dator det skulle implementeras på. Det var från början avpassat för IBM 704. Hanteringen för vissa satstyper, särskilt DO och IF-satserna, speglade hårdvaruegenskaper hos just denna dator. Fortran-specificeringen hade också i viss mån skett med tanke på förenklad objektkodoptimering för 704-an.

Ett annat, och intressantare, skäl för att ACM-kommittén nästan totalt ignorerade Fortrans existens hade att göra med IBM's position på marknaden för större datorer. Kring 1957-58 fanns inga allvarliga konkurrenter till IBM. Univac II, Bizmac, Datamatic 1000, Univac 1103/1105, alla dessa maskiner såldes, trots goda internprestanda, i små upplagor i förhållande till IBM 701/704/705/709. Det fanns en känsla hos ett flertal personer inom ACM att Fortran representerade en del av IBM-väldet. Accepterande av Fortran som bas för en internationell standard skulle ytterligare förstärka företagets marknadsposition. ACM's reserverade hållning gentemot Fortran kan ses som ett försök att öka konkurrensen på den redan vid den tiden relativt "snedfördelade" datormarknaden.

Kring 1959 tillkom flera transistoriserade datorer. Ökad maskinkapacitet, nya utvecklingslinjer; för många stod det klart att tiden var mogen för experiment med nya språk. Många algebraiska kompilatorer lanserades efter hand, och de flesta presenterades som dialekter av Algol.

Burroughs tillhandahöll till sin nya modell 220 en snabb enpassage-kompilator för Algol. För säkerhets skull gav man sina språkansträngningar en egen beteckning, Balgol. Burroughs har varit tämligen ensam Algol-supporter på den amerikanska marknaden. De flesta övriga leverantörer har helt accepterat Fortran som basspråk för beräkningstillämpningar. Burroughs har emellertid till och med haft Algol i tankarna vid maskinvarukonstruktion. Detta är anmärkningsvärt som nära nog ingen annan större leverantör ens använt det mera maskinnära Fortran för detta ändamål, ren maskinkodning eller assemblykodning har dessutom varit allenarådande för kodning av operativsystem.

Vid tiden kring 1960 dök även andra Algol-liknande språk upp. Algo, avsett för Bendix G15-datorn, JOVIAL-utvecklat av System Development Corporation - ursprungligen för en större militär IBM-dator, senare implementerat på såväl IBM 7090 som Control Data 1604, Philco 2000 och Burroughs D825. Neliac är ytterligare ett av dessa Algol-influerade språk, som implementerats på en mångfald datorer.

Flera språkuttrycksmässiga fördelar för Algol - jämfört med Fortran - kan noteras. Algol medger en mera generell uttrycksformulering, en mer flexibel hantering av iterativa förlopp, logiska variabler och satser samt villkorliga uttryckssätt. Med dessa fördelar för ögonen, visserligen med in/utmatningsbristerna och strängbegränsningarna medtagna, är det trots leverantörernas avvaktande hållningar något förvånande att Algol under hela 60-talet så nära nog totalt har negligerats vid tekniskvetenskaplig databehandling i USA. Troligen har många användare avskräckts av påståendena att Algol-kompilering är tidskrävande och att objektprogrammen blir mindre effektiva. Det första påståendet är knappast helt korrekt, det existerar idag flera Algol-kompilatorer som översätter snabbare än motsvarande dators Fortran. Kompileringstidsåtgång är i högre grad avhängigt kompilatorkonstruktion än språkegenskaper. På grund av Algol's större generalitet har det andra påståendet, om objektkodens mindre goda effektivitet, sannolikt fog för sig.

Effekten härav blir i praktiken dock ej så markant då en stor del av de jobb som bearbetas på en tekniskt-vetenskapligt orienterad datacentral är dels korta, och dels av engångskaraktär.

En mer aktiv upplysning från ACM's sida hade möjligen kunnat ändra situationen något, men som helhet är förhållandet mellan Algol och Fortran

ett bevis för styrkan i och mängden av annat än språkfaktorer och -intressen som påverkar spridningen av programmeringsspråk.

COBOL

Den sannolikt största användaren av databehandlande utrustning i världen är sedan länge USA-regeringen med dess anknutna statliga organ. Man håller styvt på att icke preferera någon enskild datorleverantör, och har därför kanske mer än någon annan fått lida av bristen på kompatibilitet mellan olika typer av datorer, antingen de tillhandahållits av samma eller av skilda tillverkningsföretag.

Våren 1959 påbörjade man arbete på att allvarligt göra något åt den besvärliga kompatibilitetssituationen avseende programvaror. Försvarsministeriet sammankallade representanter för tunga leverantörer och användargrupper till ett möte i Washington för att diskutera problemen kring frånvaron av ett standardiserat programmeringsspråk för administrativa databehandlingsproblem. Detta möte kallades the Conference on Data-Systems Languages (CODASYL). Konferensen var begynnelsen för missionärsarbetet att söka möjliggöra programmering i något som nära liknade engelska språket.

Erfarenheterna från högre programmeringsspråk för administrativa problem var vid denna tid begränsade. Flow-Matic, hörande till Univac I, tillsammans med IBM's Commersial Translator var i stort sett den enda bakgrunden. Liksom i fallet Algol två år tidigare fann man det lämpligt att tillsätta olika, skilda språkdefinitionskommittéer, i detta fall tre stycken. En skulle arbeta på "kort sikt", en annan på "intermediate range", och en tredje på "lång sikt". Det ursprungliga mandatet för den kortsiktiga kommittén rörde undersökning av existerande språk och teknik, och rapportering till CODASYL med rekommendationer om hur dessa skulle kunna användas i samband med ett nytt språks definition. Kommittén arbetade snabbt, bl a beroende på kundtryck på leverantörsmedlemmarna i densamma att tillhandahålla adekvata programvaror till redan tecknade datorer. Man beslöt sig för att önskvärt var att omedelbart starta arbetet med att specificera det nya språket. Kommittén blev känd som COBOL-kommittén.

Preliminära specifikationer för det nya språket släpptes fram vid slutet av 1959. Flera datorleverantörer, bl a RCA och UNIVAC, startade omedelbart implementeringsarbetet.

Härefter inträffade det märkliga att COBOL-kommittéerna kom i opposition mot varandra. Den "intermediära" kommittén hade studerat den kortsiktiga kommitténs specifikationer, och funnit dem otillräckliga. Särskilt vid jämförelse med Honeywells då relativt nya FACT-språk ansåg man dittillsvarande COBOL komma till korta. Man rekommenderade helt enkelt FACT som en bättre bas som standardspråk.

Kortsiktskommittén tänkte inte låta sitt arbete gå till spillo. Under retorisk debatt, och framför allt med CODASYL-styrelsens stöd, utmanövrerade man 1960 motståndet, och COBOL-språket konstaterades definierat. Givetvis behövdes förbättringar, men språkets bas skulle stå fast. Kortsiktskommitténs mandattext reviderades, för att säkra sig mot tänkbara kommande referenser till kortsiktiga mål i samband med revisionistiska nyförslag.

Specifikationerna från 1960 befanns snart vara behäftade med åtskilliga brister och motsägelser, och omarbetades i många detaljer. Den nästföljande år utkommande uppdaterade rapporten, COBOL 61, har sedermera legat till grund för de flesta leverantörers implementeringar.

Den snabba maskinutvecklingen samt den kraftiga användningen av språket, som snart växte fram, ledde 1962 till utsläppandet av den s k "COBOL 61 Extended". Formellt skiljer sig denna huvudsakligen från COBOL 61 genom att tillägg gjorts för att medge förenklad hantering av vissa standardbetonade uppgifter som t ex rapporthantering och sortering. Dessutom gjordes ett antal förklarande grammatikaliska redigeringar.

Under året just efter det att COBOL 60 hade presenterats kunde på vissa håll ett motstånd mot språket noteras. Honeywell och IBM befann sig då i slutfasen på sitt arbete med egna administrativt orienterade språk, vilka skulle mötas av mycket litet intresse om COBOL accepterades allmänt. Vid årsskiftet 1960-61 beslöt sig emellertid USA-regeringen för att backa upp COBOL med hela sin prestige och enorma köpkraft. Allt motstånd mot språket försvann. Detta åstadkoms genom det enkla kungörandet att regeringen inte hade för avsikt tillstyrka vare sig köp eller hyra av datorutrustning från någon enda leverantör om inte COBOL där fanns implementerat, såvitt inte leverantören kunde bevisa att utrustningens prestanda inte skulle effektiviseras genom detta språks tillgänglighet. Inget sådant bevis har uppvisats rörande större datorsystem. Detta kan anses som en signifikant och intressant händelse i samband med språkets utveckling.

En helt ny rapport över COBOL publicerades i slutet av 1965. Det där definierade COBOL 65 utgör en utökning av den utvidgade 61-versionen, huvudsakligen beträffande språksatser för hantering av data på direkt-

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'SORT360'.
REMARKS. THIS PROGRAM WAS WRITTEN TO DEMONSTRATE THE USE OF THE SORT
         FEATURE. THIS PROGRAM PERFORMS THE FOLLOWING TASKS -
           1. SELECTS, FROM A FILE OF 1000-CHARACTER RECORDS, THOSE RECORDS
              HAVING FIELD-A NOT EQUAL TO FIELD-B.
           2. EXTRACTS INFORMATION FROM THE SELECTED RECORDS.
           3. SORTS THE SELECTED RECORDS INTO SEQUENCE, USING FIELD-AA,
              FIELD-BB, AND FIELD-CC AS SORT KEYS.
           4. WRITES THOSE SORTED RECORDS HAVING FIELD-FF EQUAL TO FIELD-EE
              ON FILE-3 AND WRITES SELECTED DATA OF THE OTHER RECORDS ON FILE-2.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360 F50.
OBJECT-COMPUTER.  IBM-360 F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT INPUT-FILE-1 ASSIGN TO 'F401' UTILITY.
    SELECT SORT-FILE-1 ASSIGN 'SF1' UTILITY.
    SELECT FILE-2 ASSIGN 'F402' UTILITY. SELECT
    FILE-3 ASSIGN 'F403' UTILITY.

DATA DIVISION.
FILE SECTION.
FD INPUT-FILE-1 BLOCK CONTAINS 5 RECORDS
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS INPUT-RECORD.

01  INPUT-RECORD.
    02  FIELD-A     PICTURE X (20).
    02  FIELD-C     PICTURE 9 (10).
    02  FIELD-D     PICTURE X (15).
    02  FILLER      PICTURE X (900).
    02  FIELD-B     PICTURE X (20).
    02  FIELD-E     PICTURE 9 (5).
    02  FIELD-G     PICTURE X (25).
    02  FIELD-F     PICTURE 9 (5).
SD  SORT-FILE-1 DATA RECORD IS SORT-RECORD.
01  SORT-RECORD.
    02  FIELD-AA    PICTURE X (20).
    02  FIELD-CC    PICTURE 9 (10).
    02  FIELD-BB    PICTURE X (20).
    02  FIELD-DD    PICTURE X (15).
    02  FIELD-EE    PICTURE 9 (5).
    02  FIELD-FF    PICTURE 9 (5).
```

Fig 2.1:4. Exempel på sorteringsprogram i Cobol 61.

```
FD  FILE-2 BLOCK CONTAINS 10 RECORDS
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS FILE-2-RECORD

01  FILE-2-RECORD.
    02  FIELD-EEE    PICTURE $$$$$9.
    02  FILLER-A     PICTURE X (2).
    02  FIELD-FFF    PICTURE 9 (5).
    02  FILLER-B     PICTURE X (2).
    02  FIELD-AAA    PICTURE X (20).
    02  FIELD-BBB    PICTURE X (20).
FD  FILE-3 BLOCK CONTAINS 15 RECORDS
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS FILE-3-RECORD

01  FILE-3-RECORD PICTURE X (75).

PROCEDURE DIVISION.

    OPEN INPUT INPUT-FILE-1, OUTPUT FILE-2, FILE-3.
    SORT SORT-FILE-1 ASCENDING FIELD-AA DESCENDING FIELD-BB,
    ASCENDING FIELD-CC INPUT PROCEDURE RECORD-SELECTION OUTPUT
    PROCEDURE PROCESS-SORTED-RECORDS. CLOSE INPUT-FILE-1, FILE-2,
    FILE-3. STOP RUN.

RECORD-SELECTION SECTION.
PARAGRAPH-1. READ INPUT-FILE-1 AT END GO TO PARAGRAPH-2.
    IF FIELD-A = FIELD-B GO TO PARAGRAPH-1 ELSE
    MOVE FIELD-A TO FIELD-AA MOVE FIELD-F TO FIELD-FF
    MOVE FIELD-C TO FIELD-CC MOVE FIELD-B TO FIELD-BB
    MOVE FIELD-D TO FIELD-DD MOVE FIELD-E TO FIELD-EE
    RELEASE SORT-RECORD. GO TO PARAGRAPH-1.
PARAGRAPH-2. EXIT.
PROCESS-SORTED-RECORDS SECTION.
PARAGRAPH-3. RETURN SORT-FILE-1 AT END GO TO PARAGRAPH-4.
    IF FIELD-FF = FIELD-EE WRITE FILE-3-RECORD FROM
    SORT-RECORD GO TO PARAGRAPH-3 ELSE
    MOVE FIELD-EE TO FIELD-EEE MOVE FIELD-FF TO FIELD-FFF
    MOVE FIELD-AA TO FIELD-AAA MOVE FIELD-BB TO FIELD-BBB
    MOVE SPACES TO FILLER-A, FILLER-B WRITE FILE-2-RECORD.
    GO TO PARAGRAPH-3.
PARAGRAPH-4. EXIT.
```

Fig 2.1:4. (forts)

minnen. COBOL 65 utgör en så omfattande samling av språkelement att det inte effektivt låter sig implementeras på mindre till medelstora datorer. COBOL 61-rapporten angav två språkdelmängder för dessa situationer: REQUIRED och ELECTIVE COBOL. Denna strukturering finns emellertid ej medtagen i COBOL 65, där man i stället överlåtit åt standardiseringsorganen att träda i aktion i bl a nämnda avseende.

1969 lanserades av USASI (amerikanskt standardiseringsorgan) acceptans av en ytterligare utvidgad version av COBOL. Denna version inkluderar uttryck för hantering av terminalkommunikation (teleprocessing facilities).

Den språkmässigt stabila tidsperioden

I och med tillkomsten av COBOL 61 befann sig datoranvändarna i situationen att kunna disponera tre problemorienterade programmeringsspråk. Som tidigare nämnts har Algol endast nått måttlig framgång i USA, medan läget i Europa för detta språk var gynnsammare för tekniska tillämpningar. Under hela 60-talet har språkvalsfrågan i sig som ett viktigt element haft en maskintillgänglighetsbetraktelse. Även om Algol, Fortran och Cobol sedan decenniets början varit klart definierade har ett utmärkande faktum för branschen som helhet rört anmärkningsvärda programvaruförseningar. Nya datorer har i mångfald presenterats, men endast i undantagsfall har programvara kunnat levereras i samband med de första installationerna. Detta gäller såväl kompilatorer som operativsystem.

Stora men svårberäknade penningbelopp bör ha gått förlorade för användarna genom dessa förseningar. För projekt av brådskande karaktär har ofta annat programmeringsspråk än det avsedda fått väljas på grund av frånvaro av, eller olämpligheter hos, kompilator för det avsedda språket.

Nästan samtliga leverantörer har haft problem med försenade programvaruleveranser. Utan tvekan har de förlorat prestige på grund härav. Mest markanta förseningar har dock troligen IBM-kunder fått känning av, speciellt gäller detta för operativsystem för serien 360. (Beträffande OS/360 har förseningarna möjligen givits orättvist stor publicitet.)

Sextiotalet har varit en betydelsefull tid vad gäller utvecklingen av användarnas "språkkänsla". Den relativa frånvaron av utbud på nya generella språk (en värdefull andhämtningspaus) har där möjliggjort ett mognande. Man känner nu i betydligt högre grad än tidigare fördelar och nackdelar hos de olika existerande generella språken. Den ursprungligen utvecklingsmässigt lägre fackkännedomen hos användarna än hos leverantörerna har sakta men säkert börjat utjämnas, vilket bör ha varit till fromma

för branschen som helhet. En kompetent köpkader är utan tvekan värdefull som motpol till medvetna leverantörer.

Ett av de problem som delvis kunnat analyseras är frågan om önskvärda prestanda hos kompilatorer. Två som det framkommit motstridiga önskemål är här hög kompileringshastighet samt genererande av effektiv maskinkod. Ett tredje önskemål är och har alltid varit tillhandahållande av god diagnostik.

Emedan en och samma kompilator svårligen till önskad uppfyllelsegrad kan tillmötesgå alla dessa önskemål, har olika kompilatorer för samma språk sedan viss tid tillhandahållits, främst av IBM, gällande Fortran. Denna utveckling synes sund, och kan väntas fortsätta, åtminstone under 70-talet, då maskintidskostnad ännu kommer att räknas som en betydelsefull ekonomisk post vid problemlösning med datorer. Optimerande kompileringar kommer sannolikt till att börja med i ökande utsträckning att användas för programinkörningarnas avslutande faser. Den längre tidsåtgången härför uppvägs av de kortare exekveringstiderna. På längre sikt kommer språksystem[1]) med mindre god verkningsgrad, dvs sådana som genererar mindre effektiv maskinkod, troligen att accepteras mera allmänt. Dessa system får antas bli väsentligt mer användarvänliga än de idag tillgängliga.

Detta kan väntas i samband med att datorerna, och därmed deras maskintid, blir allt billigare och uppväger kostnader för det mänskliga arbetet för att kommunicera med dem, och få hjälp av dem.

Sextiotalet har även varit värdefullt för jämförelsestudier mellan problemorienterade språk och assemblyspråk. Försenade leveranser av kompilatorer för högre språk har ofta tvingat till användning av assemblykodning.

Därmed har en medvetenhet avseende programmeringskostnader vuxit fram. För- och nackdelar för

- läsbarhet
- lätthet att rätta
- korta produktionstider
- programmeringseffektivitet

---

1) Det är ej säkert att begreppet kompilering kommer att bestå under längre tid.

- kompatibilitet

- osv.

har måst studeras, vilket småningom lett till ett allmänt accepterande av högre språk i allt större utsträckning. Ännu 1969 finns dock många situationer då assemblyspråk måste eller med fördel bör användas. Detta gäller framför allt kodning av operativsystem, reelltidsrutiner samt vid användning av icke-konventionella yttre datorenheter.

Ett intressant försök att ena fördelarna hos Algol med fördelarna hos Cobol utfördes kring 60-talets mitt av Data SAAB i Sverige. Man lanserade till sin dator D 21 språket Algol-Genius (Genius: Generellt In-Utmatningssystem), ett språk som till en kärna av Algol knöt en Cobolliknande datadefinitionsdel, anropbar från Algol-programmet. Språket blev mycket uppskattat och användbart, men anammades tyvärr inte av någon annan leverantör. Trots de interna framgångarna har företaget därför till sin nuvarande maskinmodell D 22 levererat både Fortran och Cobol, även om dessa två av Data SAAB betraktas som "komplement" till Algol-Genius. En rimligt framgångsrik marknadsföring kräver idag Fortran och Cobol.

Algol-Genius har sannolikt noterats i Sovjetunionen. Vid den s k First All-Union Conference on Programming, november 1968, presenterades ett nytt språk, kallat Algol-Cobol. Denna kombination avses lämpad för kommersiella Sovjet-tillämpningar. Någon kompilator synes dock ännu 1969 ej ha framställts.

Det språkmässigt förhållandevis stabila 60-talet har givit Algol och framför allt Cobol och Fortran en stark ställning bland datoranvändare i allmänhet. Nyare språk, som lanserats, kommer att få svårt att bryta sig in och erövra mark, trots i många fall uppenbara teoretiska och praktiska fördelar. Det må kunna uttryckas (den personliga) hypotesen att ännu åtminstone fram till senare hälften av 70-talet de tre ovan nämnda språken ännu kommer att befinna sig i användning.

Nya programmeringsspråk

PL/I

Som noterats i föregående kapitel utannonserades IBM System/360 våren 1964. Programvaruutvecklingen till denna datorserie har varit både omfattande och intressant. För att citera Saul Rosen i Programming Systems

and Languages: "To an observer outside the company it is almost unbelievable that they could have started their ambitious software effort so late with such unrealistic delivery schedules". Utvecklingen av operativsystem till 360-serien kommer senare att beröras. Här skall vi närmast betrakta språkutvecklingen.

Den huvudsakliga språksatsningen rörde utveckling av det programmeringsspråk som senare har blivit känt som PL/I. Detta hade sin begynnelse som Fortran VI, ett försök inom IBM att tillhandahålla en kompatibel utvidgning av Fortran IV. Detta försök övergavs, och 1963 tillsattes en Advanced Language Development Committe, bestående av tre personer från IBM och tre representanter för användarorganisationen SHARE. Deras avsikt var att konstruera ett helt nytt språk, då kallat NPL (New Programming Language). Det ansågs mindre lämpligt att använda ordet "new" i namnet på något som avsågs existera tämligen lång tid (dessutom anlände protester mot denna bokstavsföljd från National Physical Laboratories) och namnet ändrades snart till MPPL (Multi-Purpose Programming Language). Inte heller denna beteckning befanns lämplig på sikt, och det slutliga namnet fastslogs till PL/I (Programming Language One). Det uppges att IBM har copyright på namnen PL/2 till PL/100, för att täcka eventuella framtida behov.

Alla tidigare generella språk har utkommit i successivt förändrade versioner, då med sifferbeteckningar som suffix, Fortran II/IV, Algol 58/60/rev., Cobol 60/61 ext. Det var därför naturligt för IBM att välja en sifferangivelse efter PL i namnet. Att man underlåtit att välja namnet med syftning på någon speciell användning - som de tre tidigare språken - kan ses som en indikation på den avsedda generella användningen.

IBM rapporteras ha investerat flera tusen miljoner dollar i utvecklingen av maskinvaran för System/360. Konstruktionen av PL/I, som mycket väl kan komma att visa sig viktigare för datorindustrin som helhet, avsågs klaras inom knappt ett års tid, av sex personer, av vilka somliga hade annat ansvar under tiden.

I tillgängliga källor anges ingen kommentar rörande IBM´s och SHARE´s eventuella övervägande att utvidga Algol 60 i stället för att skapa ett helt nytt språk. IBM har icke offentligen visat nämnvärt intresse för Algol, varför detta avvägande kan synas perifert. Algols betydelse som utvecklande bas framgår emellertid tydligt vid betraktande av PL/I´s nuvarande struktur. Det kan betraktas som mindre lyckligt att kompatibilitet inte ägnats större intresse vid PL/I-arbetet, med hänsyn till tidigare erfarenheter i språksammanhang.

Kommittén för utveckling av det nya språket arbetade utgående från följande allmänna mål:

1) Språket skulle tillfredsställa behoven hos mycket stora grupper programmerare, med olika tillämpningsinriktningar.

2) Det skulle vara enkelt till konstruktionen, så att så få felaktigheter som möjligt introducerades vid dess användning.

3) Det skulle vara ett språk inte bara för dagens utan även för morgondagens datorer, operativsystem och tillämpningar.

I mars 1964 presenterades den första versionen av NPL för SHARE-organisationen. Språkkommittén begärde, och fick i hög grad emotta, kritik och förslag om förbättringar. Denna första NPL-version var i vissa avseenden direkt avpassad för System/360, vars karaktäristik under löfte om hemlighållande (systemet var ännu ej utannonserat) i förväg avslöjats för kommittén. En något ändrad kommitté producerade därefter en andra version (utsläppt i juli 1965), drastiskt ändrad, som även den möttes av kritiska röster.

Successiva ändringar har sedan genomförts i icke ringa utsträckning. Inte förrän 1968 kan språket sägas ha varit slutgiltigt definierat. Denna sena tidpunkt har samband med det i många fall svaga intresse som mött språket. Inte minst inom SHARE har debatten gått het angående för- och nackdelar. Dagens tillämpningar kan i de flesta fall klaras acceptabelt med de tidigare språken, och allt större vikt har kommit att fästas vid kompatibilitetsfrågor (vilket i visst avseende talar mot övergång till nya språk).

Icke desto mindre är PL/I på frammarsch, huvudsakligen på grund av att det stöds av marknadsledaren IBM. Språket innehåller många värdefulla programkonstruktionsmöjligheter, men lider i viss mån av sitt omfång, målsättningen att klara i stort sett alla tillämpningstyper har lett till ett omfattande grammatikaliskt resultat.

Som ett tillägg till diskussionen om mål och måluppfyllelse för operativsystem i ett senare kapitel kan det vara av intresse att här betrakta de "design criteria" som PL/I-kommittén arbetade fram ur de tre ovan angivna högre målen för språket: (kommentarerna till de sex kriterierna är hämtade ur Radin/Rogoway: "Highlights of a new programming language").

1. Tolkningsgeneralitet. "Om en speciell symbolkombination visar sig ha syntaktiskt meningsfull innebörd, bör denna innebörd också vara semantiskt giltig. En PL/I-kompilator bör normalt undvika diagnostik av typen 'Det här är fel, men jag är så smart så jag förstår vad du menar' till förmån för meddelanden av typen 'Är du verkligen säker på att du avser denna underliga betydelse?'"

2. Full tillgång till maskin- och operativsystemsfaciliteter. "Om det visar sig finnas funktioner i datorsystemet som en programmerare kan nå bäst/endast från assemblynivå, så bör PL/I ej anses ha lyckats helt avseende detta kriterium."

3. Modularitet. "Jag finns om du behöver mig, i annat fall behöver du inte ens känna till min existens, och du behöver inte befara att få betala för det med kompileringstid eller maskintidseffektivitet. Manualer skall kunna konstrueras för delmängder av språket, för skilda tillämpningsmiljöer och skilda komplexitetsnivåer. Man bör sällan bli utsatt för kompileringsfel genom att utelämna något."

4. Relativt maskinoberoende. "Trots att språket speglar IBM-datorer (360), skall språkparametrar, som t ex talstorlek, typer av yttre maskinenheter etc, vara oberoende av karaktäristiken hos någon speciell dator."

5. Vänlighet mot nybörjare. "Trots allmän språkspecifikation syftande på kraftfullhet, strikthet, expansivitet etc skall specialsituationer beskrivas som explicit redundanta. En kompilator skall maximera effektivitet för normala situationer."

6. PL/I är ett programmeringsspråk, inte ett algoritmiskt språk. "Program skrivs oftast på kodblanketter, stansas på hålkort, och trycks på radskrivare. Trots att specifikation av ett publikationsspråk är väsentligt, måste stor vikt fästas vid att erhålla läsbara programlistor, samt att söka göra skrivning och stansning så enkel och felfri som möjligt."

De (så exakt som möjligt) citerade kommentarerna speglar en i vissa avseenden framsynt inställning, i andra en viss trångsynthet.

PL/I är ett mångsidigt språk. I det har införts ett flertal nya begreppsmöjligheter som tidigare inte existerade i generella programmeringsspråk. Som exempel kan nämnas användande av logiska uttryck som aritmetiska operander, omdefinieringsmöjligheter, möjlighet att arbeta med parallella processer, viss listhantering osv. Om språkets framtid

Problem: Inventory control program for company with 20,000 stock items.

Program:

```
INVCTL: PROCEDURE;
     DECLARE (OLDMAST INPUT, NEWMAST OUTPUT) BLOCK (FIXED,432,8),
          PFILE OUTPUT,
          1 WORK,
               2 PARTNO CHARACTER (7),
               2 DESCR CHAR(12),
               2 (QOH, QOO, RP, RQ) FIXED (5),
               2 UP FIXED (6),
               2 YTDSALE FIXED (8),
               2 CODE FIXED,
          1 TRANS,
               2 TNUMBER CHARACTER (7),
               2 TCODE FIXED,
               2 TQ FIXED (5),
          CODEIS (4) LABEL;
     ON ENDFILE (STANDIN) BEGIN; TNUMBER = '9999999';GO TO WRITNM; END;
     ON ENDFILE (OLDMAST) BEGIN; IF TNUMBER = '9999999' THEN DO; CLOSE
                                        OLDMAST DISCARD, (PFILE, NEWMAST)
                                        STORE, DISPLAY ('JOB FINISHED');
                                        END;
                                   ELSE ERROR: DISPLAY ('FILE OR DATA
                                        ERROR'); EXIT; END;
     ON SUBSCRIPTRANGE BEGIN; DISPLAY ('BAD CLASS CODE JOB HALTED');
          EXIT; END;
     READ (TRANS)(A);
READM: READ FILE (OLDMAST), (WORK) (A);
TESTM: IF PARTNO < TNUMBER THEN WRITNM: DO; WRITE FILE (NEWMAST),
                                   (WORK) (A); GO TO READM; END;
       IF PARTNO > TNUMBER THEN GO TO ERROR;
            /*THEN PARTNO = TNUMBER*/
       GO TO CODEIS (TCODE);
CODEIS(1): QOH = TQ; GO TO JOIN;
CODEIS(2): QOH = QOH + TQ; QOO = QOO - TQ; GO TO JOIN;
CODEIS(3): QOO = QOO + TQ; GO TO JOIN;
CODEIS(4): IF QOH < TQ THEN DO; WRITE ('ONLY', PARTNO, 'AVAILABLE',
            QOH, ' REQUESTED') (3A, F(5), A); TQ=QOH; END; QOH =
            QOH-TQ; IF CODE = 1 THEN YTDSALE = YTDSALE + TQ*UP;
JOIN: IF QOH + QOO =RP THEN WRITE FILE(PFILE), (PARTNO, CODE, RQ) (3 A);
     READ (TRANS) (A); GO TO TESTM; END INVCTL;
```

Fig 2.1:5. Exempel på lagerkontrollprogram i PL/I.

kan inget ännu sägas säkert, men mycket talar för att det kommer att vara ett accepterat språk vid senare delen av 70-talet.

Algol 68

Den internationella bakgrunden för tillkomsten av Algol 60 borgade för en snar och berikande debatt om för- och nackdelar för språket. Tidigt efter publiceringen av den "reviderade rapporten" startade också ett meningsutbyte rörande hur språket skulle kunna förbättras i olika avseenden. Allt eftersom denna debatt om tekniska förbättringar fortlöpte pågick också arbete på en mera fullständig omarbetning av språket. Diskussionen rörde inte minst kompatibilitetsfrågan: skulle en vidareutveckling vara en direkt utvidgning av Algol 60, eller skulle man satsa på ett mera separat och nytt språk? Här skilde sig uppfattningarna i många fall, vilket bl a återspeglades i språkets debattforum, publikationen Algol Bulletin.

IFIP (International Federation for Information Processing) arbetade sedan 1963 med en efterföljare till Algol 60. Detta skedde huvudsakligen via dess WG2.1 of TC2 (Working Group 2.1 of Technical Committee 2). Det betraktade språket gick länge under beteckningen Algol X, där X avsågs stå för det okända publikationsåret. Efter mycken debatt och intensivt arbete kunde X tilldelas värdet 68. Språkbeskrivningen Report on the algorithmic language Algol 68 publicerades under överinseende av IFIP vid årsskiftet 1968-69. Under arbetets gång hade man prövat språkets utlärbarhet och övriga egenskaper vid sju skilda kurser med högt kvalificerade deltagare. Erfarenheterna från dessa kurser fick påverka successionen av de olika versionerna av det nya språket. (Tänkbart alternativ hade varit att i stället pröva språket på kurser med avsiktligt sammankallade normalkvalificerade deltagare.)

Algol 68 är ett eget och nytt språk, helt skilt från, och inkompatibelt med, Algol 60. Vissa likheter existerar, men de är i de flesta fall endast av formell art. Man har strävat efter att i Algol 68 inkludera vidareutvecklingar av de framgångsrika begreppen från Algol 60, dock som specialfall av mera generella konstruktioner, medan man infört helt nya begrepp beträffande tidigare mindre framgångsrika företeelser (av typen own-kvantiteter och heltalslägen), samt kompletterat i ett flertal avseenden.

Det låter sig inte göras att på ett begränsat antal rader specificera Algol 68's avvikelser från Algol 60. Endast ett par omnämnanden får i detta sammanhang anses lämpligt:

- Typ-begreppet har generaliserats till ett obegränsat antal "modes" (tillstånd).
- Hantering av binära bitar, bytes och symboliska formler är möjliggjord i och med att de kan ges värden.
- Referenser mellan olika värden kan upprättas, sålunda kan kedjor av indirekta adresser byggas upp.
- Algol 60's procedurbegrepp har generaliserats.
- Godtyckliga typer av konstanter kan deklareras.
- Nydefinition eller omdefinition av operatorer är möjlig.
- Dynamiskt föränderligt antal variabelvärden kan byggas upp vid exekveringsdags.
- Parallellt arbetande processer är tillåtna.
- Själva begreppet "tilldelning" kan ges ett värde.
- Listhantering är möjlig.
- In- och utmatning är definierad.

Utan tvekan kan dessa egenskaper sägas indikera att Algol 68 utgör ett kraftfullt och flexibelt språk. Den tillgängliga första beskrivande rapporten avses emellertid inte vara det slutgiltiga dokumentet över språket. Tvärtom inbjuds till debatt och meningsutbyte, syftande mot framtida förbättringar.

Det indikerade meningsutbytet har inte uteblivit. Speciellt till ett bestämt avseende har det kanaliserats, och detta rör språkets beskrivningssätt. Algol 68 presenteras i den ovan nämnda rapporten med hjälp av ett helt nytt publikationsspråk, som använder sig av en än kraftigare komprimering än Algol 60's s k Backus' normalform. Det är sålunda synnerligen svårt för en normalbevandrad potentiell användare att ur Algol 68-rapporten kunna bilda sig en klar uppfattning om och överblick över språkets möjligheter. Rapporten är skriven för fackmän i avancerad mening. Som utlärningsmedel för nybörjare är rapporten oanvändbar. Denna som det kanske kan synas tekniska fråga har medfört att en minoritetsgrupp inom den för språket ansvariga arbetsgruppen år 1969 reserverat sig mot rapportens utseende. Frågan är uppenbarligen inte betydelselös, med hänsyn till den vikt för kompilatorbyggare och som slutgiltigt referensmaterial som kom att fästas vid motsvarande Algol 60-rapport.

En falang av språkexperter, bland vilka märks N. Wirth och T. Hoare, har arbetat vidare på en huvudsakligen kompatibel utveckling av Algol 60

i stället för att helhjärtat stödja Algol 68. Falangens presenterade språkversion, kallad Algol W, utgörs av Algol 60 plus sträng- och listbehandling samt ett antal övriga tekniska förändringar av typen förenklade repetitionssatser, annan talavrundning m m.

Kritiken mot Algol 68, sådant det presenterats i den första rapporten, kan ännu ej klart överblickas. Först när aviserad tilläggslitteratur finns tillgänglig, och språket kan diskuteras i ett större forum, kan debatten bli konstruktiv i större skala.

Det förefaller emellertid klart att Algol 68 innehåller många egenskaper som utgör en intressant ansats till ett programmeringsspråk för framtida användning. När språket kunnat testas i full skala, kan konstateras om det parallellt med PL/I (eller PL/X) skall kunna komma att utgöra en värdefull del av generella programvaror.

Dedikerade språk

List- och strängspråk

Allteftersom under 60-talet datorernas flexibilitet och stora användbarhet i de mest skiftande situationer blev alltmer uppenbar, växte också kraven på deras programvaror. De tre allmänt accepterade generella språken (Algol, Fortran, Cobol) hade konstruerats utgående huvudsakligen från 50-talets tillämpningstyper, och visade sig i vissa avseenden mindre lämpliga för specialiserade problemkategorier. Det var främst symbolbehandling och simulering som påkallade mera direkt avpassade språkfaciliteter. (Till simulering återkommer vi nedan.)

Även om t ex Cobol medger möjlighet till textmanipulering, blir program däri för ett flertal tillämpningar relativt långa och svårhanterliga. Som exempel på problemutseende som påkallar särskilda språkmöjligheter kan vi bl a nämna:

1. Problemet består huvudsakligen av manipulering av symboler som har icke-numerisk anknytning.

2. De speciella minnesbehoven för problemlösandet kan inte specificeras i förväg.

3. Komplexa datastrukturer formas under programexekveringens gång, t ex styrda av varierande data- eller parameterinmatning.

4. Relationer mellan dataelement omstruktureras under utförandet.

5. Den problemlösande proceduren modifieras väsentligt under programutformningen. Detta speglar användningen av datorer som hjälp för studium av själva problemets uppbyggnad.

Den mest påtagliga av dessa punkter kan vid första åsyn sägas vara den första. Det är också denna som fått ge samlande namn åt de språk för den åsyftade problemkategorin som under 60-talet växte fram. De kallas sålunda ofta textbehandlande. Den nära liggande benämningen listspråk syftar på bearbetning av symboler, angivna i form av (länkade) listor. En viss skillnad kan sägas existera mellan dessa begrepp.

En lista är ett sätt att representera information i en dator, vanligen genom att i varje minnesord lagra inte blott ett dataelement utan även adressen till nästföljande element i listan. Vi kan tänka oss, t ex, ett ord uppdelat i två lika delar, där ett informationsbärande datum lagras i den första delen och succedentadressen i den andra. Representationsprincipen kommer till sin fördel genom att den medger sammanbindning av helt generella datastrukturer.

En sträng är kort och gott en följd av karaktärer. Vi kan alltså säga att en sträng kan representeras i en dator i form av en lista. Begreppen lista och sträng är sålunda icke synonyma.

De tillämpningar, för vilka behandling av strängar och listor visat sig lämpliga, brukar ofta benämnas symbolbehandlande eller symbolmanipulerande.

Utmärkande drag för de symbolbehandlande språken kan grovt tänkas sammanfattas på nedanstående sätt:

a) Datarepresentationen. Data må innehålla symbolisk såväl som numerisk information. Informationsinnehåll ges inte bara ur det symboliska innehållet i data, utan även ur den relationsmässiga strukturen mellan data.

b) Minnesutnyttjandet. Minnesbehovet för exekvering av ett program kan inte förutsägas, utan minne tilldelas allt eftersom det erfordras. Vanligtvis bildar minnescellerna ej konsekutiva sekvenser i minnet, utan snarare komplicerade mönster. Varje minnescell anknyts till strukturen via en eller flera länkadresser, visare, som ovan nämnts. Under programexekveringens gång kan nya element tillfogas till strukturen, samt icke längre erforderliga utrymmen ledigförklaras. De ledigförklarade utrymmena måste givetvis kunna göras tillgängliga igen under

exekveringens gång, och härför tillämpas olika algoritmer för olika språk. Man talar om lumpinsamling (garbage collection), avseende insamling av lediga celler till den "fria listan", en hjälplista med adresser till outnyttjade tillgängliga celler. - Tillgängligt minnesutrymme för exekvering av ett program begränsas fysiskt naturligen av datorns primärminne. Då emellertid ej programmets minnesbehov kan förutsägas, uppbackas ("utvidgas") primärminnet under exekveringen ofta direkt av lämpligt sekundärminne.

c) Stackutnyttjandet. En stack eller en "pushdown store" kan ses som en sträng med egenskapen att endast det första (översta) elementet är direkt tillgängligt. (Som exempel på en stack kan vi tänka oss ett provrör med en spiralfjäder i botten. I detta fylles kulor /element/ på uppifrån. Endast det översta kan direkt nås.)

d) Rekursiv teknik. Program som arbetar rekursivt är ofta erforderliga för bearbetning av liststrukturer av godtyckligt utseende. En rekursiv procedur R är som bekant en procedur, inom vars procedurkropp proceduren R själv anropas. När detta inträffar måste "gamla" argument och delresultat sparas (räddas) medan den inre bearbetningen fortskrider. Sparade värden placeras i en stack. Det sist undanstoppade värdet i stacken kommer att först behövas (bli "upp-poppat") när rekursiviteten till slut är färdigbehandlad.

Ett flertal textbehandlande språk har lanserats. Bland de tillämpningar, för vilka de kommit till användning, kan nämnas:

- informationssökning
- dokumentationsteknik
- beslutsfattande (beslutsträd etc)
- språköversättning
- formelmanipulation
- bildhantering (grafisk databehandling)
- automatisk bevisföring
- artificiell intelligens
- spel

m m.

Vi kan här knappast gå in på de speciella egenskaper som utmärker eller skiljer de olika språken från varandra. En kort karaktäriserande

uppräkning av några typiska språk får anses vara tillfyllest. Sålunda noterar vi:

- IPL-V, Information Processing Language. Operationellt redan 1959. Assembly-liknande utseende. Implementerat på ett stort antal datorsystem.

- L^6, Bell Telephone Laboratories Low-Level Linked List Language. I funktion 1965. Utvecklat av K.C. Knowlton. Datablockhanterande, högt kodifierat.

- LISP 1.5, List Processing Language. I funktion 1961. Utvecklat under J. Mc Carthy vid MIT. Ovanlig programgrammatik. Ett språk för finsmakare. Existerar i time-sharing version.

- COMIT, publicerat 1957 av V. Yngve, MIT. Lingvistiskt påverkat strängspråk.

- SNOBOL, utvecklat 1962 vid Bell Telephone Laboratories av Farber-Griswold-Polonsky. Påverkat av, och utvecklat från, COMIT. Strängspråk, lämpligt för mönsterproblem.

- TRAC, Text Reckoning And Compiling. Utvecklat 1960 av C. Mooers, implementerat 1964. Lämpligt för interaktiv hantering av ostrukturerad text. Programutseende: linjära kapslade textfunktioner. (Vidareutvecklat 1968 av Lindecrantz-Thorelli vid Tekniska Högskolan, Stockholm.)

- FORMAC, Formula Manipulation Compiler, utvecklat 1962 av J. Sammet, IBM. Språk för matematisk formelhantering. Utvidgning av Fortran m fl.

Flera av dessa språk använder s k polsk notation, vilket främjar dem från skrivsätt i de generella problemorienterade språken, och gör dem tekniskt något mera svåröverskådliga. Begreppet polsk notation (från den polske filosofen Lukasiewicz) innebär att man i stället för t ex a + b skriver + a b eller sum (a, b), sålunda

<operator> <operand> < operand >

vilket beteckningssätt ofta kommit till användning vid konstruktion av formella språk. Vanligt är även, i andra sammanhang, s k omvänd polsk notation, innebärande a b +, eller

<operand> <operand> <operator>

Beträffande framtiden för de speciella textbehandlande språken kan följande utveckling skönjas: De nyare generella språken innehåller eller kommer i icke ringa grad att innehålla faciliteter för textmanipulering och hantering av mer generella datastrukturer. Därför kan intresset för de speciella språken förmodas avta. En utveckling som emellertid motverkar denna trend är det alltmer ökande intresset för små datorer. Dessa kommer av köpar-ekonomiska skäl företrädesvis att förekomma i konfigurationer, vilkas minnes- och bearbetningsresurser knappast är lämpade för tunga och omfattande generella språk. För dylika maskiner blir specialiserade språk väsentliga. Framtiden för bl a de textbehandlande språken är sålunda relaterad till eventuella framgångar för existerande och kommande mindre datorer. Enligt författarnas uppfattning är de små datorernas framtid lovande, varför en enkelriktning av hela språkkomplexet knappast är att vänta.

Problem: Reverse a list

Program:

```
DEFINE ((
(REVERSE (LAMBDA (X) (COND
    ((NULL X) NIL)
    (T (APPEND (REVERSE (CDR X)) (LIST (CAR X)) )) ))) ))
```

Problem: Test whether two list structures are equal, where "equal" means they have the same hierarchical structure and the same elementary symbols in corresponding positions.

Program:

```
DEFINE ((
(EQUAL (LAMBDA (X Y) (COND
    ((ATOM X) (COND
        ((ATOM Y) (EQ X Y)) (T NIL)))
    ((ATOM Y) NIL)
    ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
    (T NIL) ))) ))
```

Fig 2.1:6. Exempel på program i LISP 1.5.

Simuleringsspråk

För studium av många omfattande och komplexa problem har analytiska lösnings- och beräkningsmetoder ännu ej fullt utvecklats, eller också kräver sådana alltför hög approximationsgrad. Det kan t ex gälla studium av komplicerade produktionsapparater, datorprocesser, trafiksystem, sociala eller ekonomiska system m fl. Många experiment med dylika systems beteende har gjorts genom att konstruera en logisk modell av systemet och sedan beskriva den med hjälp av ett datorprogram. Vid

exekvering av programmet kan observationer göras avseende modellens tillstånd i tiden.

Normalt avses med "simulering" att vi förflyttar en logisk modell av ett system i tiden, och därvid observerar modellens tidsbeteende. Härur kan sedan extrapolationer göras tillbaka till det ursprungliga systemet. Egentligen sker lösning (eventuellt med hjälp av dator) av varje problem via en simulering eller avbildning i form av en analytisk eller logisk modell.

Utmärkande för system som simuleras bör vara:

- de är dynamiska, dvs har tiden som en viktig variabel.
- i dem ingår matematiskt mindre "vackra" funktioner (styckvis kontinuerliga, begränsat diskontinuerliga etc), liksom komplexa, tidsberoende relationer mellan delsystem.
- problemförloppen är ofta slumpberoende, eller kan approximativt avbildas med hjälp av slumpvariabler.

Vid betraktande av dylika problem är ofta explicita eller ens optimerande lösningar ej möjliga att erhålla. Med simulering söks då en satisfierande lösning. Även om härmed osäkerhet ofta införs i ett kanske ursprungligen exakt problem, kan i de flesta ingenjörstekniska fall tillräcklig lösningsnoggrannhet uppnås. Lösningsnoggrannheten kan givetvis också varieras, då den bl a är kopplad till tillgängliga datorresurser (minne, tid etc).

Vid betraktande av klassen av simuleringsproblem låter sig en naturlig uppdelning göras mellan system vars tillståndsvariabler antas variera kontinuerligt och de som antas variera diskret. Ett kontinuerligt problem kan t ex röra studium av ett elektriskt flöde eller ett vätskeflöde. Diskreta problem avbildar systemkomponenter med finitiserade funktioner, t ex föremål förflyttar sig stegvis genom det betraktade systemet. Exempel kan här vara informationsflöde, flödet av material genom en verkstad eller flödet av fordon inom ett trafiksystem.

För simuleringsproblem i allmänhet är intresset knutet till att söka få grepp om det betraktade systemets "kapacitet", dvs hur många föremål som flyter genom systemet på en given tid, som funktion av systemets struktur. Dessutom söks ofta få grepp om var de genomströmmande föremålen befinner sig i systemet vid aktuella tidpunkter. Dylika processer kan ej sällan ses som stokastiska, varför statistisk metodik är av stor betydelse.

Kontinuerligt varierande problem kan ofta med tillräcklig noggrannhet approximeras med diskreta modeller, vilket har fått till följd att de simuleringsspråk som utvecklats företrädesvis har varit avpassade för diskreta problem.

Innan vi nämner något om de speciella simuleringsspråken bör vi göra klart för oss att de generella programmeringsspråken nära nog alltid kan användas för programmering av simuleringsproblem. Vid val av lämpligt språk får språktillgänglighet, kompatibilitet, tillgänglig datortid etc ställas mot erforderlig programmeringstid och -bekvämlighet.

Tämligen få simuleringsspråk för rent kontinuerligt varierande problem har lanserats. Vi kan bland dem lägga märke till DYNASAR samt DYNAMO, det senare utvecklat 1959 vid MIT. Ingetdera av dessa har emellertid tillnärmelsevis nått samma spridning som språken för diskreta problem, av skäl som i någon mån ovan antytts. De mest spridda av de diskreta språken är GPSS (General Purpose Simulation System) samt SIMSCRIPT. GPSS konstruerades och lanserades av IBM vid slutet av 50-talet, och kan ses som ett problemblockorienterat assemblyspråk för simuleringsproblem. Trots att språket besitter vissa användningstekniska brister, såsom t ex att det icke är kombinerbart med något högre generellt språk, har bl a dess lättanvändbarhet givit språket en vid spridning.

Ett annat tidigt simuleringsspråk, SPS-1 (Simulation Programming System), utvecklat vid Rand Corporation, fick tillsammans med GEMS (General Electric Manufacturing Simulator) bilda bakgrund till det kring 1962 lanserade språket SIMSCRIPT. Detta språk, som utvecklades med hjälp av amerikanskt militära RAND-penninganslag, är nära anknutet till FORTRAN, bl a avseende programmeringsteknisk formatbundenhet och formellt utseende. Genom sin högre generalitet, jämfört med GPSS, har det kunnat komma till användning för en vid kategori av simuleringsproblem. SIMSCRIPTS karaktär av högre programmeringsspråk har bl a lett till att det implementerats på ett flertal olika datortyper, för nära nog samtliga större utrustningsleverantörer. Det är f n, åtminstone i USA, troligen det mest allmänt accepterade simuleringsspråket.

Under 60-talets gång har dessförutom en mångfald andra språk för simuleringsproblem konstruerats, vilka i detta sammanhang ej kommenteras. Vi kan dock lägga märke till SIMPAC, utvecklat vid System Development Corporation, samt CSL (Computer Simulation Language), för vilket IBM, via kundmedverkan, står som fadder. CSL är ett FORTRAN-påverkat språk speciellt avpassat för studium av datorkonfigurationer under olika givna jobb-belastningar, vilket av IBM bl a används i samband med beräkningar kring offerering av datorutrustning.

I framför allt Norden har knappast något av de ovan nämnda simuleringsspråken, förutom GPSS, tilldragit sig det största intresset. I stället har detta visats det vid Norsk Regnesentral av Nygaard och Dahl konstruerade SIMULA. Detta språk är en direkt utvidgning av ALGOL, och medger sålunda tillgång till detta generella språks fulla flexibilitet tillsammans med speciella simuleringsfaciliteter. Språket har varit i drift sedan 1964, implementerat på Univac 1107 och den kompatibla 1108. De positiva erfarenheterna från användningen av SIMULA ledde till att vidareutvecklingsarbete snart startades, resulterande i den 1967 frisläppta definitionen av SIMULA 67 (varvid det ursprungliga SIMULA omdöptes till SIMULA I). SIMULA 67 är i själva verket ett helt nytt språk, med endast begränsade formella anknytningar till SIMULA I. Det är betydligt mer än endast en utvidgning av ALGOL, med ett antal nya begrepp och faciliteter, som medfört att det rönt stort teoretiskt intresse. Dess praktiska användning har ännu ej kunnat prövas i någon större omfattning, emedan den första kompilatorn färdigställts 1970.

```
begin integer nmg; read(nmg);
SIMULA begin integer array available [1 : nmg];
   set arrary que [1 : nmg];
activity order(n); integer n;
begin integer i, mg; integer array mgroup[1 : n];
   array ptime[1 : n];
   read(mgroup, ptime);
   for i:=1 step 1 until n do
   begin mg := mgroup [i];
   if available[mg] = 0 then
   begin wait (que[mg]);  remove(current) end
   else available[mg] := available[mg] -1;
   hold(ptime[i]);
   if empty(que[mg]) then available[mg] := available[mg] + 1
   else activate first (que[mg])
   end path through shop
   end order;
integer n;  real T;
read(available);
next:  read(n, T);  reactive current at T;
if n > 0 then begin activate new order(n); go to next end
end SIMULA end program
```

Fig 2.1:7. Exempel på program i SIMULA 1.

I SIMULA 67 har ALGOL's block- och procedurbegrepp generaliserats till ett "klass"-begrepp, som medger generella programstrukturer. Den parameterinmatning till ALGOL's procedurer, som medger styrning av programutförande, har i SIMULA 67 utvidgats till att speciella "tillämpningsspråk" enkelt kan definieras. Detta innebär att användaren via identifieraranrop av en önskad klass utan vidare får tillgång till samtliga definitioner och begrepp från denna tidigare uppkodade klass. För speciella tillämpningar kan således enkla användar-"språk" en gång för alla definieras, avsedda för hantering från personal utan detaljkännedom om annat än sin speciella problemkategori. Härmed har två egenskaper uppnåtts, för det första undvikandet att av användaren kräva kunskap i mer detaljerad simuleringsprogrammering (vilken kan ses som något svårare än "vanlig" programmering) samt för det andra har problemet med sammanknytning av enkla tillämpningsspråk med ett generellt programmeringsspråk lösts. Detta tilltalande faktum, tillsammans med ett flertal andra intressanta egenskaper hos SIMULA 67, har medfört att språket som nämnts tilldragit sig stort intresse. Planer föreligger rörande implementering på IBM 360, syftande till realisering 1971-72.

Tillämpningsspråk

Alltsedan början av 50-talet, då intresset för assemblyspråk, och senare högre språk, väcktes, har man funnit att kommunikationen mellan människa och maskin väsentligt kan underlättas med hjälp av en rad mer eller mindre specialiserade språk. Detta har huvudsakligen varit betingat av önskvärdheten att kunna få access till problemlösningshjälp av dator via så användarvänlig programmering/data- eller instruktionsinmatning som möjligt. För stora användargrupper har det eftersträvats att kräva minimal programmerings- och datorteknisk kunskap för att kunna beskriva problemet för datorn.

För programsystem av tillämpningsorienterad typ är det svårt att dra en klar gräns mellan "språk" och "program". Skillnaden kan sägas ha att göra med de möjligheter användaren har tillgängliga för att styra bearbetningen i datorsystemet.

En vanlig indelning av programmeringsspråk är i:

1) procedurorienterade språk

2) icke-procedurorienterat språk.

Hos ett procedurorienterat språk krävs att användaren i detalj beskriver de algoritmer enligt vilka bearbetningen skall utföras. För ett icke-

procedurorienterat språk krävs ej denna detaljspecifikation av användaren. De instruktioner som tillhandahålles (i form av användarprogram) har icke direkta motsvarigheter i maskininstruktioner, utan genererar (vid översättning) sekvenser av maskininstruktioner. Definitionen av icke-procedurorienterade språk är icke entydig. Svårigheter föreligger att dra klara gränser mellan begreppen, då procedurorienterade resp icke-procedurorienterade delar numera innehålls i snart sagt varje språk. I typiska procedurorienterade språk, som Algol, Cobol, Fortran, PL/I etc, är t ex in/utmatningsoperationerna av icke-procedurorienterad typ.

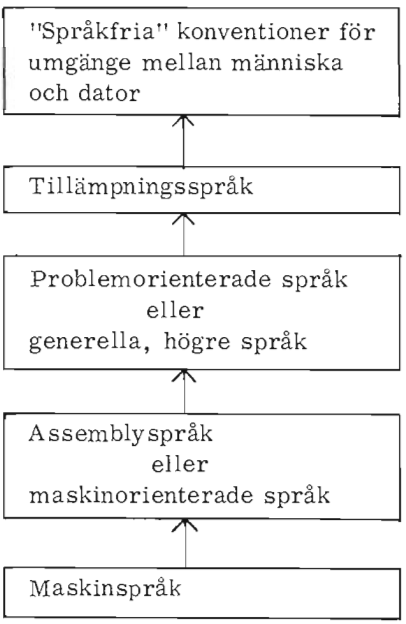
En viss nivåsynonymitet kan ses mellan tillämpningsspråk och starkt icke-procedurorienterade språk. Ett minimum av instruktioner krävs här av användaren. Dessa språk befinner sig på nivå som är långt avlägsen från maskininstruktionsnivå. Sekvenser av maskininstruktioner genereras med hjälp av från användaren tillhandahållna (styr)-parametrar. Exempel härpå är sorteringsgeneratorer, rapportgeneratorer o d.

Karaktäristiskt för tillämpningsspråk är att de kräver mera programvarustöd i form av specialiserade tillämpningsprogram än de generella språken. Många tillämpningsspråk kräver förutom lejonparten av de faciliteter som finns i vanliga generella språk dessutom ett antal speciella rutiner. Tillämpningsspråk tillhandahåller oftast någon form av aritmetik, styrinstruktioner och testinstruktioner. Härförutom behövs ofta rutiner av typ koordinatberäkning, optimering o d. Detta senare åstadkommes ofta med hjälp av subrutin/proceduranrop från generella språk. Vi kan således ofta se tillämpningsspråken befinna sig på en våning (nivå) över de generella (vilka sistnämnda i sin tur befinner sig en nivå över maskininstruktionsnivå).

Ju starkare specialiserat ett tillämpningsprogramsystem konstruerats, desto enklare kan det sägas vara hanteringsmässigt. Från användarsynpunkt kan vi sålunda konstatera en "omvänd proportionalitet" mellan hanterbarhet och språkgeneralitet. Ett språk som är utformat för att klara en vid kategori av tillämpningar är förhållandevis mera komplicerat att lära sig använda än ett starkt specialiserat.

En mycket stor mängd tillämpningsspråk har lanserats under 50- och 60-talen. Då dessa ligger utanför denna skrifts problemområde kommenteras de ej närmare. En viss klass- och typindelning, av karaktären interpretativa i förhållande till kompilerande språk samt med olika tillämpningsorientering och generalitetsgrad, bör dock kunna vara av intresse i flera sammanhang. Ett dylikt arbete, av komplett typ, väntar dock ännu på sitt utförande.

Nivåuppdelning mellan språkklasser



Dialogspråk

Den utveckling av maskin- och programvara, som kunnat noteras under huvudsakligen 60-talets senare hälft har ställt språkfrågorna delvis i en ny dager. Datorutvecklingens, och något senare även operativsystemsutvecklingens, framsteg har i detta sammanhang rört begreppet timesharing. Möjligheterna till mera direkt kontakt mellan människa och maskin, samt snabbare datorer, har förskjutit tyngdpunkten tillbaka något från kompilerande till interpretativa språk. Vid kort omloppstid mellan instruktionsgivande och instruktionsutförande sker själva översättningen till maskinkod naturligen på ett interpreterande sätt. Kompilering, dvs klumpvis översättning av större mängder instruktioner en gång för alla, före maskinexekveringens början, blir i time-sharing drift icke ett tidsekonomiskt och normalt sätt för umgänge mellan användare och dator. Detta hindrar dock inte att en begäran om kompilering skall kunna utföras om användaren så skulle önska. Skillnaden kan sägas ha att göra med hur stora programbitar som begärs exekverade, och vilka krav som ställs på omloppstid, maskinutnyttjande etc.

I time-sharing drift, med korta svarstider, är omfattande och generella språk ofta mindre lämpliga för ett effektivt utnyttjande av datorsystemets resurser. Snabba växlingar mellan stora programblock kräver mycket datorresurser. För många problem räcker något mindre generella språkresurser. 60-talets uppvaknande intresse för små datorer, i samband med decentralisering av maskinresurserna, har som följd att de maskinresurskrävande tunga språken i många fall kommer på undantag. Datorernas konfigurationer räcker knappast till resursmässigt. Detta kan dock knappast betraktas som någon väsentlig nackdel, då användarna ofta umgås med de mindre datorerna på ett mera specialiserat sätt. Användningen av datorer i time-sharing miljö sker ofta med syfte till tillgång till texthanterare eller intellektförstärkande räknesnurror med i många fall begränsade flexibilitetsönskemål.

Bland olika typer av interaktion mellan användare och datorsystem kan nämnas:

- Rättning av fel i program
- Syntaxkontroll för varje rad eller sida
- Inläsning av data under exekvering
- Indirekt resp direkt exekvering av satser (indirekt exekvering innebärgivande av styrkommandon)
- Möjlighet att avbryta exekvering
- Möjlighet att undanlagra/hämta fram program- och datadelar.

Ett flertal språk för dialoghantering har lanserats. Det för närvarande kanske mest spridda, BASIC (Beginner's All-purpose Symbolic Instruction Code), har utvecklats vid Dartmouth College i USA kring 1964. Detta språk salufördes till att börja med av General Electric, till sitt GE 265-datorsystem (GE 235 sammankopplad med en kommunikationsdator Datanet 30), men har sedermera successivt anammats av övriga större leverantörer. BASIC är ett Algol-liknande språk, dock med ett flertal starka begränsningar jämfört med Algol 60. Ett viktigt mål såväl för BASIC som för andra liknande språk, avsedda för tillämpningar i time-sharing miljö, har varit enkelhet i utlärning och löpande hantering. Sålunda bör en orutinerad användare efter några få timmars språkbekantskap via sin terminal (vanligen av skrivmaskinstyp) behärska språket tillräckligt för att kunna få meningsfullt arbete utfört.

```
LET ⟨variable⟩ = ⟨expression⟩
GOTO ⟨statement number⟩
GOSUB ⟨statement number⟩
RETURN
IF ⟨expression⟩ ⟨relation⟩ ⟨expression⟩ THEN ⟨statement number⟩
FOR ⟨unsubscripted variable⟩ = ⟨expression⟩ TO ⟨expression⟩ STEP ⟨expression⟩
NEXT ⟨unsubscripted variable⟩
READ ⟨variable⟩, ⟨variable⟩, . . . , ⟨variable⟩
PRINT ⟨literal or expression⟩, ⟨literal or expression⟩, . . .
STOP
END
DIM ⟨variable⟩ (⟨integer⟩ [, ⟨integer⟩])
DATA ⟨number⟩, ⟨number⟩, . . .
REM ⟨any string of characters⟩
DEF FN ⟨letter⟩ [(⟨unsubscripted variable⟩)] = ⟨expression⟩
```

Fig 2.1:8. Exempel på satstyper i BASIC.

Andra time-sharing-språk, ofta avsedda för aritmetiska beräkningstillämpningar, är JOSS-systemet, utvecklat 1964 vid RAND Corporation, QUICKTRAN (IBM, 1963), en specialiserad on-line Fortran-version, Digital Equipment Corporations FOCAL (Basic-liknande), FIGARO från Cambridge University, USA, samt CPS (Conversational Programming System), det sistnämnda framställt av Allen-Babcock Corp samt IBM och i drift 1967, ursprungligen en delmängd av PL/I.

Många dylika språk har sålunda sett dagens ljus sedan datorer med time-sharing började saluföras i större upplagor. Gemensamt för dem kan grovt sägas vara

- enkla aritmetiskt/textmässiga språkfaciliteter
- förekomst av enkla styrinstruktioner, avsedda för datorsystemet (dvs ej problembeskrivande instruktioner) som en direkt del av språket.

Även för speciella tillämpningar för time-sharing-system har separata språk konstruerats. Som exempel kan nämnas det ökande intresset för datorstödd undervisning (DU) och konstruktionsberäkning. För DU existerar bl a språken COURSEWRITER, framställt vid IBM, PLANIT (Programming Language for Interaction and Teaching) från Systems Development Corporation, PLATO (Programmed Logic for Automatic Operation) och SOCRATES från University of Illinois.

Time-sharing-system har befunnits väl lämpade för text/strängbehandling, bl a med sikte på det uppvaknande intresset för dokumentation och informationssökning. I ett tidigare avsnitt har dessa språktyper kommenterats.

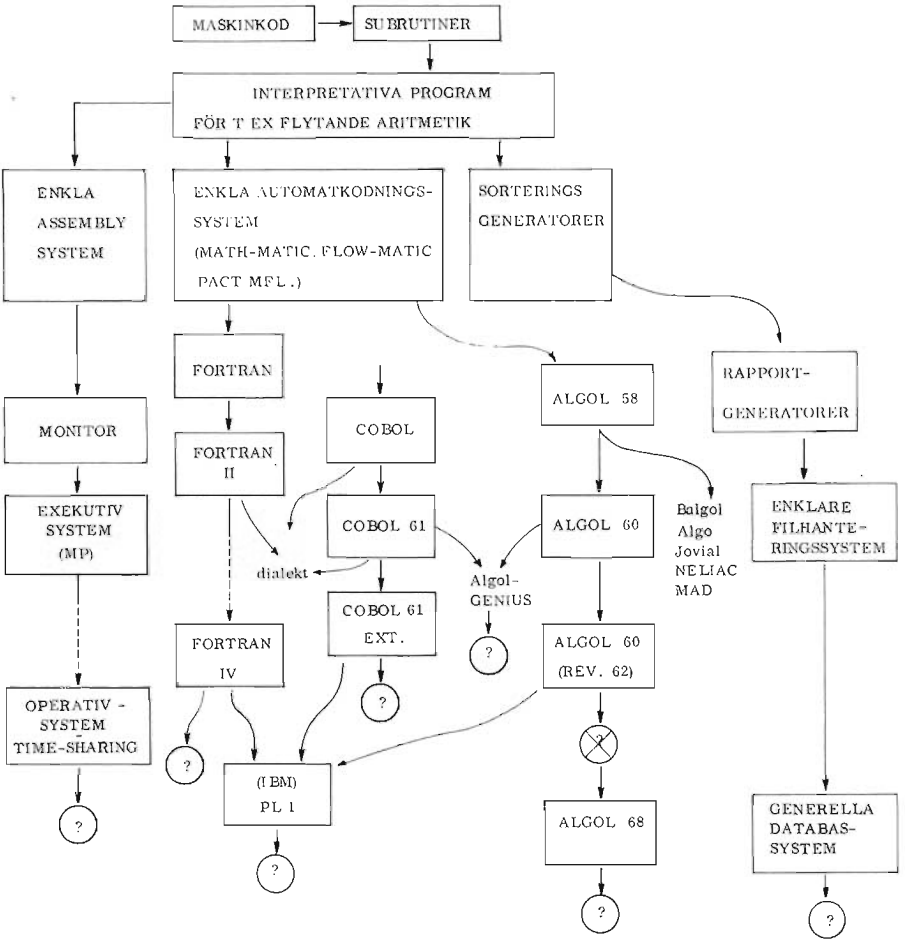
Den framtida språkutvecklingen

De i språkfrågor engagerade standardiseringsverken arbetar av naturliga skäl i ett långsamt tempo. Vi kan knappast räkna med någon snar och enande direktiv rekommendationsverksamhet från denna sida. I stället kommer branschen långsamt och naturligen att bli självsanerande, i så motto att användarintresse främst knyts till de språk som kan förväntas nå spridning genom att tillfredsställa mer eller mindre reellt underbyggda önskemål. Enär spridning nås via användarintresse må detta cirkelresonemang synas tvivelaktigt. Någonstans måste initiativen tas. Det är troligt att leverantörerna av maskin- eller programvara, åtminstone till att börja med, kommer att stå för dessa. Användarorganisationerna blir ännu under 70-talet knappast tillräckligt kraftfulla för allvarligt kunna påverka leverantörerna. På längre sikt bör vi dock räkna härmed.

För 1970-talet kan förmodas att de allmänt accepterade problemorienterade språken Algol, Cobol, Fortran i viss utsträckning kommer att kunna hålla undan för PL/I, Algol 68 och SIMULA 67. De tre dagsaccepterade problemorienterade språken har nått en sådan spridning, och så mycket har både bland användare och leverantörer investerats i dem, att någon snabb intresseomsvängning knappast är att vänta. Denna synpunkt stöds också av det allmänt uppvaknande intresset för detaljstudier av datorsystems användarekonomi. Man är numera på det klara med att det kan vara synnerligen kostsamt att byta programmeringsspråk inom en användarsfär. Kostnader i samband med omprogrammering av existerande rutiner, utbildning, kompatibilitet etc är av en omfattning som är fullt jämförbar med systemets maskinvarukostnader. En avvaktande och i viss mån konservativ hållning i språkfrågor är därför även bland användare naturlig.

Denna konservatism kommer dock knappast att kunna dämpa intresset för forskning kring programmeringsspråk. Snart sagt samtliga universitet och leverantörer arbetar på ett eller annat sätt med dessa problem. Inom Control Data lanserades 1968 det generella språket GPL (Generalized Programming Language). Detta språk, som innehåller Algol 60 som en delmängd, är uppbyggt kring en minimal språk-kärna, och kan betraktas som "self-extending". Språket medger bl a implicita symboldefinitioner och listhantering. GPL har ännu ej saluförts med någon större tyngd av moderföretaget. "Utvidgbara" (self-extending) språk kommer på lång sikt troligen att visas stort intresse.

Användning av datorer för direkt reelltidsmässig styrning av administrativa och industriella processer kommer att öka kraftigt. Denna småningom



" SOFTWARE " - UTVECKLINGEN

alltmer dominerande kategori av problem är svår att tackla med idag existerande generella språk. Assemblyspråken kommer förmodligen ännu under början av 70-talet att dominera on-line-programmeringen. Önskemålen om ett gemensamt språk för reelltidsprogrammering är starkt uttalade - dock har få ansatser ännu gjorts för att få en samordning till stånd. Skäl talar för att PL/I kan utvidgas till ett PL/X som besitter de egenskaper vi idag kan bedöma erfordras för reelltidsprogrammering. Maskinvaran i morgondagens datorer kan möjligen tänkas anpassas till PL/-linjen.

En annan intressant utvecklingstrend gäller "språk för definition och beskrivning av informationssystem". Forskningsprojekt pågår avseende dylika språk, och målsättningen är att en utförlig systembeskrivning i ett sådant språk skall vara tillräcklig för automatisk konstruktion och implementering av informationssystemet (struktur, filer, program etc).

Ett aktuellt problem rör formellt språkutseende. Sedan datorernas tillkomst har det blivit allt tydligare att det finns nära nog lika många sätt att säga:

addera a till b

som det finns personer kapabla att säga det. Var och en trivs bäst med sin egen version. Försöken att med enande åtgärder tillfredsställa alla har knappast lyckats, användarna tenderar att i stället skapa egna språkversioner. En okomplicerad förbättring jämfört med dagens situation vore att t ex som ovillkorlig hoppinstruktion tillåta endera av

GO TO, JUMP eller TRANSFER TO.

Införande av dylika valmöjligheter i större skala i språkgrammatikor, vilket i viss utsträckning finns tillgängligt i PL/I, skulle öka användarvänligheten till priset av endast ringa ökning av besvär/tidsåtgång för kompilatorn. Fortsättande på denna tankegång förefaller det klart att intresse kommer att knytas till system som möjliggör för användaren att själv definiera formella språk. Härför krävs en flexibelt utvidgbar språkkärna, samt ett system för kompilatorsyntes, med vilket en kompilator för det användardefinierade språket kan genereras. Kompatibilitet reser dock substantiella problem härvid. Forskningsarbete rörande dylika projekt pågår sedan ca 1967 bl a vid MIT, USA.

Fördelen med att låta användaren själv definiera sitt språk ligger i att datorsystemet därmed tvingas anpassa sig till mänskliga konventioner - i stället för tvärtom som hittills varit fallet. Erfarenheter bekräftar svårigheterna med att finna generella språk som tillfredsställer alla.

Somliga önskar en utveckling i riktning mot naturliga språk, t ex engelska, andra talar om de naturliga språkens begränsningar och tvetydigheter, och önskar kommunicera med datorer via formellt matematisktlogiska språk. Genom att tillåta användarna att välja själva kan på längre sikt den ena eller andra skolan komma att visa sig värdefullast.

Den fullaste generaliteten rörande språkgenerering medför emellertid problem rörande spridning av språken. Många användare anpassar sig hellre till lämpliga konventioner än ägnar sig åt egen språkdefinition. Vi ser att kompromisser blir önskvärda.

För umgänget med framtidens datorer kan två utvecklingslinjer tänkas:

1) Utveckling av starkt modulära, parametriserade applikationspaket, som flexibelt kan kommunicera med varandra.

2) Görande av var och en till sin egen programmerare, samtidigt förenklande de konventioner som krävs för umgänge med en dator. Vi kör normalt hellre själv våra bilar än anlitar professionella chaufförer, telefonerar hellre själva osv.

Det förefaller klart att metoden 2) är den mest lovande. Varje individ, och ej enbart programmerarexpertisen, bör ha möjlighet att inspektera, skydda och i viss utsträckning manipulera den information som rör individen själv. Datorerna bör närma sig människorna, och inte tvärtom. Individen har rätt till datorservice, liksom vatten, elektricitet, sophämtning osv. (Införandet av postnummer nyligen i Sverige är ett /beklagligt/ undantag som bekräftar denna regel. Inom få år kan datorer i ekonomisk och praktisk drift läsa tryckbokstäver /och senare skrivbokstäver/ på brev. Avskaffandet av postnummerbegreppet blir då givet.)

För den stora nytillkommande kategori användare av datortjänster som önskar kommunicera på enklaste sätt blir naturligt språk sannolikt det enda realistiska alternativet på sikt. När tvetydiga instruktioner ges får datorn begära förklarande information. När alltför omfattande resurser begärs får datorn stillsamt protestera. Längre bort skymtar önskvärdheten att kunna kommunicera med datorer via mänskligt tal i stället för skrift. Diskussion härom faller emellertid utanför denna boks syfte.

Innan vi når fram till kommunikation via naturligt språk kommer en rad stadier rörande programmeringsspråk att behöva passeras. Låt oss avsluta detta avsnitt med att ange några av de önskemål vi kan ställa på framtida programmeringsspråk:

1) Enkelhet, vänlighet mot användaren.

2) Generalitet, användbarhet för olika tillämpningar. (Unifiering till ett enda universellt språk är dock knappast önskvärd.)

3) Utbyggbarhet, språkuppbyggnad i expansionsmöjlig och flexibel modulform, användbar utan kompilatoromskrivning.

4) Inkrementalitet och icke-procedurorientering vid problemdefinition. De delar av problemet som ej vill eller kan fixeras vid initialprogrammeringen bör kunna lämnas öppna för komplettering vid exekvering. Problem bör kunna ges i grov och till att börja med odetaljerad form.

5) Självdokumenterbarhet, ett program bör utgöra programbeskrivning över sig själv.

6) Säkerhet, språket bör tidigt och med hög pålitlighet indikera syntaxmässiga programmeringsoklarheter.

Framtiden får utvisa i vilken utsträckning dessa önskemål kommer att realiseras i kommande språk.

LITTERATUR

1. Kallin, S., FORTRAN, Studentlitteratur 1967

2. Lysegård, A., COBOL, Studentlitteratur 1967

3. Ekman-Fröberg, ALGOL, Studentlitteratur 1967

4. Higman, B., A comparative study of programming Languages, Elsevier Publ. comp. 1967

5. Wilkes, M.V., Time-sharing Comp. Systems, Elsevier Publ. comp. 1968

6. Naur, P. etc, Rev. report on the algorithmic Language Algol 60, Regnesentralen, Phpn 1964

7. Markowitz-Hausuer-Karr, SIMSCRIPT, Prentice-Hall 1963

8. IBM, GPSS-manual

9. Nygaard-Dahl, SIMULA, Norsk Regnesentral 1967

10. Lindecrantz, N., Datamaskinförmedlad undervisning, Student-litteratur 1968

11. van Wijngaarden etc, Report on the algorithmic Language Algol 68, Mat. Centrum, Amsterdam 1969

12. Palme, J., Krav på framtidens programmeringsspråk, FOA-rapport 1969

13. Rosen, S., Programming Systems & Languages, Mc Graw-Hill 1967

14. Sammet, J., Programming Languages, Prentice-Hall 1969.

2.2 Operativsystemens utveckling

Inledning

I denna översiktliga beskrivning över operativsystemens utveckling från 1950-talets början till 60-talets slut kommer en kronologiskt orienterad organisation att följas. Alternativt hade en funktionell indelning kunnat ligga till grund för framställningen. För detaljstudier kan den sistnämnda indelningen vara den mest adekvata, men enligt författarnas uppfattning erhålls en lämpligt översiktlig uppfattning om utvecklingens förlopp med hjälp av en kronologisk genomgång. Vi återkommer senare till diskussioner om en funktionell strukturering av operativsystem.

Vi vill i detta avsnitt inte göra anspråk på att täcka in hela fältet kring utvecklingen. Ett dylikt arbete skulle vara av mycket stort omfång. Inte heller kan vår beskrivning helt och fullt täcka det tidsmässigt allra första uppträdandet av en speciell teknik eller angreppsmetodik för lösning av uppträdande styrproblem. Vi har valt att välja ut några i vårt tycke representativa systemprinciper, och låta dessa spegla utvecklingen.

I den mån värderingar rörande betraktade system uppträder vill vi betona att de härrör från författarna enbart. Vi har dock så långt möjligt sökt undvika att presentera annat än verifierbara sakförhållanden.

Den huvudsakliga utvecklingen för operativsystem har påverkats av otillräcklighet hos de system som omedelbart föregått betraktade system tidsmässigt. En tämligen klar succession kan rörande vissa systemfunktioner således spåras. Detta samband gäller dock ingalunda systemen som helhet.

Ett operativsystem kan mycket grovt sägas utgöra en på en given dator simulerad modell av ett önskat och annorlunda system. Detta annorlunda system kan sägas vara ett funktionellt sett utökat datorsystem. Ett antal funktioner har tillförts datorsystemet i syfte att dels underlätta människans kommunikation med systemet, och dels effektivisera dess drift. Flexibilitetskrav har medfört att dessa funktioner hittills så gott som uteslutande implementerats programvarumässigt.

Tiden före 1956

Enkel jobb-för-jobb bearbetning

De tidigaste operativsystemen utgjordes av det samlade agerandet mellan operatör och programmerare. Det vore kanske mera adekvat att be-

nämna dessa system "operatörssystem". De relativt låga bearbetningshastigheterna för datorer före mitten av 1950-talet, samt de icke överlappade datatransportoperationerna resulterade i långa bearbetningstider, varav icke ringa delar härrörde från manuella operatörsingripanden. Det kan också tilläggas att de flesta körningar vid denna tid var av typen "beräkningsarbeten" med relativt sett stor belastning på systemets processenhet, och ringa belastning på in/utmatningsenheterna. Denna bild ändrades dock successivt i och med att allt fler administrativa rutiner lades över till automatisk databehandling. Ett typiskt användningssätt gick ut på att i förväg (ofta dagar, ibland veckor) beställa maskintid på ett av driftsavdelningen noggrant uppgjort tidsschema. En inbokad t ex halvtimme garanterade då att användaren fritt och ensamt, med operatörens hjälp, kunde disponera datorn. Han kunde sålunda helt lita på att inget annat program skulle belägga någon del av datorsystemet under den beställda maskintiden. Användaren debiterades också för denna fulla tid, oberoende av hur han använt denna sin halvtimme. Utnyttjandetaxor löpte rent linjärt med tiden, x kronor per timme.

Förhoppningsvis skulle datorn vara maskinmässigt i funktion, "uppe", under den beställda tiden. Driftsavbrott, då icke sällsynta, kunde leda till att tidsintervallet inte kunde utnyttjas på avsett sätt, med förskjutningar och löpande ändringar i tidsplaneringen som följd. Kontroverser kunde uppstå mellan användare som på olika sätt blivit lidande av driftsstörningar. Systemservice var till att börja med endast i måttlig omfattning preventiv, dvs regelbunden genomgång och byte av tekniska komponenter (radiorör, kondensatorer, motstånd etc). Utrustningsfel åtgärdades oftast när de påverkade ett användarprogram, med hårt pressade arbetsförhållanden för den servicetekniska personalen som följd.

För den för datorns drift ansvariga personalen innebar denna tid svåra planeringsproblem, med prioriteringar, reservtider osv som ett dagligt återkommande dilemma. (I viss mån existerar dessa problem fortfarande.) En användare kunde ofta praktiskt tvingas till att inte bara beställa maskintid för den planerade körningen, utan också ett eller flera reservtidspass att användas om datorn ej var fullt "uppe".

En dators driftstillförlitlighet var vid denna tid direkt avhängig tillhörande servicepersonals kunnande och den tid som eventuellt ägnades åt systemets preventiva service. Därför kunde högst varierande tillförlitlighetsvärden noteras för skilda installationer av samma fabrikat.

Kontakt människa - maskin

Användarmässigt innebar denna tid en intim kontakt mellan programmerare och dator. Det normala arbetsförhållandet innebar att programmeraren själv skötte styrning av datorns agerande, eventuellt med hjälp av operatören för de mera tekniska handgreppen. Programmeraren startade sålunda sin körning med att rensa maskinen från bl a eventuella data från tidigare körning (nollställning av primärminne, register m m) samt initierade därefter inläsning av de egna programmen. Det är uppenbart att datorns processenhet under denna direktinläsning var lågt belagd. Huvuddelen av beräkningskapaciteten, eventuella sekundärminnen samt utmatningsenheterna utnyttjades därvid icke. Den efter inläsningen och laddningen av programmet startade exekveringen styrdes ofta med hjälp av manuell parameterinmatning av programmerare/operatör. Vanligtvis var de tidiga datorerna försedda med både "blinkande lampor" och ofta även högtalarutrustning. De till de aritmetiska registren inkopplade visuella/akustiska organen (lampor, högtalare) uppvisade under körningens gång mönster, som i viss mån kunde igenkännas av en erfaren programmerare. Med de relativt begränsade diagnostiska hjälpmedel som stod till buds under programinkörning följde relativt frekventa inkörningstillfällen. En programmerare lärde därvid successivt känna sitt program ingående och lade ofta intuitivt på minnet hur datorn betedde sig under programhanteringen.

En oavsiktlig oändlig slinga igenkändes av det ljud- eller blinkmönster som härrörde från den begränsade adressväxlingen i de aritmetiska registren. Det var naturligtvis omöjligt att exakt tolka maskinens agerande med hjälp av ljus- och ljudsignaler, men det är ingen tvekan om att god vägledning ofta erhölls. Härför talar också a priori det faktum att datorernas styrkonsoler vid denna tid tämligen undantagslöst verkligen utrustades med lampor, högtalare och en imponerande uppsättning av olika knappar och reglage. Dessa lampor och knappar användes ej sällan i samband med den "exklusiva" felsökningsmetoden "one-step". Detta innebar att man manuellt kunde stega sig fram genom programmets instruktioner, och därvid steg för steg avläsa innehållet i olika register. Denna dyrbara felsökningsmetodik har sedan dess ersatts av "tracing" - en steg för stegmässig utskrift av olika programtillståndsdata.

Det är rimligt att anta att flera potentiella köpare av datorer slutligen övertygades om en dators förträfflighet efter ett besök i maskinrummet. Många tillverkare synes än i dag lägga vikt vid design av operatörspanelen, där i flera fall inslaget av lampor, knappar och olika förkromade detaljer är stort.

Detta ur de flesta synpunkter ineffektiva hushållande med maskinresurser förde dock en viktig fördel med sig, som ofta inte påpekas tillräckligt klart. Vi tänker här på kontakten mellan programmerare och dator. Det intima samspelet dem emellan möjliggjorde ofta korta inkörningstider och totalt sett låga programmeringskostnader. På ett tidigt stadium kunde fel noteras och ofta omedelbart åtgärdas. Programmerarna utbildade sig dessutom mer eller mindre automatiskt till goda kännare av ett maskinsystems olika egenarter, och kunde dra fördel härav bl a vid systemarbetet. - Denna viktiga kontakt mellan man och maskin har undertryckts vid dagens datorcentraler, där en sluten driftsprincip (closed-shop) normalt tillämpas. Användaren befinner sig idag på flera sätt långt avlägsen från aktuell dator, och har därvid, trots tillgång till effektiv maskinkapacitet, ofta begränsad kännedom om, och möjlighet till kontakt med, det maskin- och programvarumässiga datorsystemet. Även om tekniska maskinkunskaper givetvis icke bör krävas i nämnvärd utsträckning av dagens datoranvändare, påverkar sådana kunskaper alltid applikationsprogrammens effektivitet i gynnsam riktning.

Tiden 1956-1959

Tidiga satsvis arbetande system

Allteftersom bearbetningskapacitet, och även kostnad, ökade, blev det allt tydligare att de tidigare driftsprinciperna var ineffektiva. En operatör tilldelas småningom en användare för att på ett alltmera aktivt sätt assistera vid körningstillfället, och ofta helt överta körningsansvaret. Icke desto mindre var omloppstiderna (turn-around-times) långa. Den erforderliga tiden för att sätta upp ett jobb, dvs för att montera magnetband, nollställa datorns minnen samt läsa in och ladda programmet, var ej sällan längre än den tid maskinen använde för assemblering och/eller exekvering.

Det stegvisa förfarandet med

1) laddning av assemblern

2) assemblering av programmet (ev med flera "genomgångar")

3) laddning av det genererade objektprogrammet

4) initiering av exekveringen

blev sålunda successivt ett så vanligt förfarande att det blev tydligt att det, tillsammans med flera andra operatörsfunktioner, borde kunna ersättas av ett styrprogram.

Utvecklingen av de första programvarumässiga operativsystemen kan ses som ett naturligt steg. Man hade tagit hjälp av själva datorsystemen för översättning (assemblering) av symbolorienterad programmering. Det föreföll naturligt att låta datorn även vara behjälplig kring användningen av översättarprogrammet.

1955 startades samarbete mellan General Motors och North American Aviation i syfta att producera ett operativsystem för IBM 704, baserat på erfarenheter från tidigare arbeten på 701-an. Idéerna spred sig. FORTRAN's tillkomst 1957 motiverade ytterligare dessa ansträngningar. Nu hade man minst två allmänna översättare, assemblern och FORTRAN-kompilatorn, i frekvent användning i maskinrummet.

Satsvis sammanbuntning av jobb automatiseras

De tidiga system som nu växte fram bestod huvudsakligen av en s k "monitor" med uppgift att

1) vid önskade tidpunkter kalla in assemblern eller kompilatorn till primärminnet

2) ladda de genererade objektprogrammen

3) handha växlingen mellan de konsekutiva jobben.

Här förutom ingick subrutiner, i antingen källkod eller objektkod, erhållna från en biblioteksfil på magnetband, samt ofta en "post-mortem" minnesutskriftsrutin (dump). Jobb grupperades samman till vad som kallades en sats (batch), och bearbetningen fortlöpte från satsens början till dess slut, då samtliga dess jobb behandlats. De relativt låga hastigheterna hos on-line anslutna hålrems- eller hålkortsläsare och radskrivare (normalt en av varje per datorsystem) kunde överbryggas genom användning av magnetband för såväl satsens jobbinmatning som för dess resultatutmatning.

Satsen mediakonverterades till band med hjälp av en off-line kort- till-band-enhet och utmatningen konverterades medelst en off-line band-till-radskrivare-enhet. Dessa off-line enheter, satellitenheterna, matades manuellt med magnetbandsrullar, och även kommunikationen mellan de båda satelliterna och huvuddatorn sköttes manuellt.

Om speciella köråtgärder krävdes av ett jobb angavs dessa på köranvisningskort, som också tjänade syftet att identifiera jobbet för operatören.

Härvid erhöll alltså operatören order om vilka band som behövde monteras eller om vilka möjligen speciella åtgärder som skulle vidtas vid tänkbart onormalt programuppträdande. Emedan tillräckligt många bandstationer normalt inte fanns tillgängliga vid en installation för att i förväg kunna montera upp samtliga databand som en sats av jobb krävde, fick operatören utnyttja tiden för exekvering av ett jobb för montering av det nästkommande jobbets band. Vid korta jobb, eller då jobb misslyckades redan efter kort tid, kom detta förfarande att innebära ett inte ringa arbete för operatörspersonalen. Ofta fick satsens bearbetning i huvudmaskinen tillfälligt stoppas, i väntan på erforderlig bandmontering.

I vid bemärkelse kan vi här tala om de första stegen mot buffrad in/utmatning, ur totalsystemets synpunkt sett. Datorns bearbetningsenhet uppehölls inte av långsam in/utmatning i samma utsträckning som tidigare. Ökningen i hastighet från on-line läsare/skrivare till de väsentligt snabbare magnetbanden innebar en markant ökad systemkapacitet och effektivitet. Härtill bidrog även i någon mån alltmer standardiserade manuellt orienterade driftsprinciper. Programmeraren var inte längre tillåten närvara i maskinhallen vid körningstillfället, och inte heller erforderlig i samma utsträckning som tidigare. Detta system, med endast operatörer i maskinhallen, kom att kallas closed shop, i motsats till tidigare open shop-system.

Den ökade rationaliseringen av driften ställde högre krav på programmerarna än förr. Inte längre kunde man med personlig närvaro bedöma körningens normala eller onormala fortlöpande. Kontrollfunktioner rörande programmets exekvering fick därför i högre utsträckning byggas in i själva programmet. Speciellt under inkörning av program blev därvid användning av en mångfald resultatmässigt redundanta kontrollutskrifter ett normalt förfarande. Programvarorna var vid detta stadium av utvecklingen knappast felsökningsvänliga i tillräcklig utsträckning, varför införande av closed shop kan sägas ha medfört ett ökande arbete för programmerarna.

Omloppstiderna för jobbhanteringen kunde emellertid förkortas i och med införande av denna driftsrationalisering, vilket kom datorutnyttjandet i stort till godo. Trots omloppstidernas förkortning är det dock tveksamt om totaltiden[1] för inkörning av ett program minskade. Minsta triviala stansfel medförde att man fick vänta till körning av nästa sats, som ej sällan ägde rum påföljande dag.

---

1) I dagar räknat. Något statistiskt underlag för detta påstående föreligger dock ej.

Kontroll över yttre enheter

Det ökade ansvaret för programmerarna kom även att beröra den yttre filverksamheten. Alla in/utmatningsoperationer fick programmeras på ett mera enhetligt sätt i och med att man ej med manuell hjälp lika flexibelt kunde styra exekveringen. Den manuella styrningen via från styrpanelen inmatade parametrar kom således att minska i betydelse, i den mån inte exakta besked om styrningen kunde ges till operatören på förhand.

Programmeringen av de bandorienterade in/utmatningsoperationerna fick utföras med ökad försiktighet. Vissa bandenheter var förbjudna, nämligen de som innehöll systemkomponenter. Översättare, subrutinbibliotek, intermediära filer för översättarna osv fick givetvis ej skrivas över/förstöras. Av programmeraren krävdes också till att börja med att efter exekveringens slut återlämna kontrollen till operativsystemet, ofta genom att läsa in en mindre och självladdande instruktionssekvens, som befann sig vid begynnelsen av ett visst systemband. Det var också nödvändigt att han/hon såg upp med läsning från jobbinmatningsbandet. Ingen läsning kunde givetvis accepteras förbi det filslutsmärke som där skilde det aktuella jobbet från det nästkommande.

Det var uppenbart att stor möjlighet fanns för en programmerare att störa eller avbryta det oskyddade systemets drift. Genom olämplig fil-läsning eller -skrivning kunde lätt systemet bringas ur fas. Bl a för att minska riskerna härför infördes i systembiblioteket rutiner för hantering av systemets in/utmatning, för återlämning av kontrollen till systemet efter avslutad körning, och för begäran om minnesutskrift före utstämpling, om så erfordrades.

FORTRAN's inträde underlättade för programmeraren att använda dessa systemrutiner, då all in/utmatning i FORTRAN går via subrutiner. Systemrutinerna utformades även för användning från andra språk än FORTRAN. För de olika språken gemensamma aritmetiska subrutiner kom till användning. Rutiner för sinus, kvadratrot, exponentiering etc lagrades för det mesta i absolut objektform, för att möjliggöra utnyttjande från olika språk utan modifiering.

Användning av i absolutform lagrade biblioteksrutiner medförde i början vissa problem för användaren. Han var tvungen utforma sitt program så att det inte använde biblioteksrutinernas utrymme i primärminnet. Dessutom kunde inte två rutiner anropas i programmet, vilka var avsedda för samma absoluta primärminnesutrymme. Laddning av programmet med dess erforderliga rutiner skedde ju en gång för alla före exekveringens start.

För att effektivt utnyttja tillgängligt primärminne skrevs eller genererades användarprogrammen i relativ i stället för i absolut form. Alla adresser var därvid relativa till minnesadressen noll, och till samtliga relativadresser adderades en lämplig adress-storhet av systemet före laddning. Härvid angavs ett antal programegenskaper på ett (eller flera) s k program-kort, som omedelbart föregick själva programmet. Dylika egenskaper var t ex

- programmets längd
- identifikation för alla erforderliga subrutiner
- information för utförande av adressmodifieringen osv.

Tidigt i utvecklingen stod det klart att ett jobb kunde bestå av mer än ett enda källprogram. Ett huvudprogram (det till vilket kontroll initialvis överlämnas) samt ett flertal egna subrutiner i källform blev ett normalt programutseende. Det blev också (successivt) önskvärt att kunna inplacera egna subrutiner i form av objektkortbuntar i stället för i källform. För att klara dessa ting blev det nödvändigt att fixera en speciell fil på ett s k exekveringsband, för systemanvändning. Även detta band var otillåtet för programmeraraccess. Allt eftersom de i källform tillhandahållna programsegmenten översattes lagrades deras objektversioner upp, på lämpligt sätt blandat med de rutiner som redan färdiga kunde tas från systemrutinbandet. Vissa system krävde att av användaren tillhandahållna rutiner i objektform alltid skulle följa efter samtliga källprogrambuntar, och i dessa fall kunde objektrutinerna laddas direkt från jobbinmatningsbandet, utan att tidigare ha kopierats över till exekveringsbandet.

Länkning och laddning

Erfarenheterna växte successivt kring användning av de primitiva operativsystemen, och vissa gemensamma principer utkristalliserades. Bland dessa kan vi lägga märke till förfaranden kring länkningen och laddningen av aktuella program. Ett normalt bearbetningssätt kan sägas ha inneburit följande:

De programsammanlänkande systemrutinerna, länkaren (linkage editor), kallades in till primärminnet efter det att inläsningen av ett jobb från exekveringsbandet, genererat enligt beskrivning ovan med relokerbara objektprogram, nått fram till jobbets första datakortbild. (Data lagrades nära undantagslöst som hålkortsbilder på bandet.) För tydlighets skull begärdes hos många system av programmeraren att mellan program och data infoga ett speciellt styrkort som otvetydigt klargjorde när länkaren skulle anropas.

Länkaren återspolade exekveringsbandet till jobbets start, dvs omedelbart före (när vi nu spolar baklänges) jobbets inledande identifikationskortbilder. Härefter lästes sekvensiellt framlänges på bandet de objektprogram som tillhandahållits av användaren. En av systemet given startadress i primärminnet, basadressen, användes som startpunkt för länkarens samarbete med laddaren vid laddning av de färdiga programmen i jobbet. - På detta stadium i utvecklingen hade ännu inte någon helt klar gräns dragits mellan de systemfunktioner som utförde länknings- och laddningsarbete. Vi använder emellertid redan här de båda skilda begreppen för att term-mässigt inleda till kommande uppdelning. - (Det första segmentets på exekveringsbandet) huvudprogrammets adresser ökades var och en med den givna basadressen (relokerades), vilket innebär övergång från relativa till absoluta adresser, och huvudprogrammet kunde skrivas in i, laddas, till primärminnet, med start i den givna basadressen. Samtidigt upprättades två tabeller, en för "erforderliga subrutiner" och en för "närvarande subrutiner" i ett speciellt avpassat minnesutrymme.

I tabellen över "erforderliga subrutiner" inskrevs identifierarna för de subrutiner som fanns anropade i huvudprogrammet, dvs huvudprogrammets externa referenser noterades.

Efter det att huvudprogrammet laddats, ökades basadressen med längden av det laddade programmet.

Härefter laddades eventuella andra delprogram med start i den nyligen korrigerade basadressen. Därvid inplacerades detta delprograms (subrutinens) identifierare i tabellen över "närvarande subrutiner". Om denna subrutin i sin programkropp anropade andra "icke närvarande" subrutiner, inplacerades dessas identifierare i tabellen "erforderliga subrutiner".

På detta sätt fortlöpte länkningen och laddningen delprogram för delprogram. Vid laddningen av varje delprogram inskrevs i tabellen över "närvarande subrutiner" dennas identifierare, samt infördes i "erforderliga subrutiner" identifierarna för de (eventuella) rutiner den aktuella subrutinen anropade. När jobbets slut på exekveringsbandet nåtts var det naturligt att mängden "erforderliga subrutiner" var större än mängden "närvarande subrutiner". Säkert erfordrades några, (minst två) in/utmatningsrutiner, och ofta andra matematiska systemrutiner, som t ex kvadratrot. Systembiblioteket söktes därvid sekvensiellt efter de erforderliga rutinerna, och allteftersom de påträffades laddades även dessa konsekutivt i primärminnet. Härvid infördes för varje laddad systemrutin dess identifierare i tabellen "närvarande subrutiner", och om dessa i sin tur krävde externa rutiner note-

rades detta i "erforderliga subrutiner". För den händelse tabellen "erforderliga subrutiner" efter den avslutade kontrollen i systembiblioteket innehöll fler identifierare än som fanns i tabellen "närvarande subrutiner", avslutades laddningen och jobbet avslutades (kontrollen gavs till nästa jobb) tillsammans med en felangivelse om vilka subrutiner som anropats men ej kunnat påträffas vare sig i systembiblioteket eller bland de av användaren tillhandahållna rutinerna.

Denna länknings- och laddningsprocess med sökande på systembandet efter de erforderliga subrutinerna var tämligen tidsödande. I senare system ersattes detta sökande med genomgång av en speciellt uppgjord tabell över identifierare för de på systembandet närvarande rutinerna. Dessutom uppgjordes i dessa senare system en annan tabell över erforderliga subrutiner redan samtidigt som den snabba jobbinmatningsfilen skapades. På så vis kunde utan explicit systembandsökning detekteras om jobbet erfordrade "icke existerande" system- eller användarrutiner.

Den relativt begränsade tillgången på primärminne (då som nu) ledde till utveckling mot uppdelning av ett jobb i flera delar, samt sammankedjande av dessa delar i en s k överlagringsstruktur. I amerikansk terminologi talas här om "overlay", ett ord som möjligen kan försvenskas till "överlagring". Härvid användes ett speciellt kontrollkort för att indikera slutet på oberoende laddningar till primärminnesutrymme, vilka behandlades separat av laddaren. I stället för att exekvera dessa direkt stackade laddaren dem först i en speciell fil (undanlade dem sekvensiellt), och det aktiva huvudprogrammet kunde kalla in dem successivt till ett därtill avsett primärminnesutrymme, varvid endast ett överlagringsprogram kunde befinna sig i primärminnet vid en viss tidpunkt. Vissa möjligheter fanns för användaren att avsätta speciella primärminnesutrymmen för data som avsågs kunna göras tillgängliga för dessa "överlagringar" gemensamt.

Systemuppföljning och debitering

För uppföljning av datorsystemets olika aktiviteter avsattes den till detsamma on-line anslutna skrivfaciliteten (normalt en elektrisk skrivmaskin), som alltså inte användes för resultatutmatning, till kontrollutskrifter över de arbeten systemet utfört (the system log). Vid anrop av fundamentala systemkomponenter skedde en notering på denna logg. Om en klocka fanns ansluten hårdvarumässigt noterades också tidpunkten för de olika aktiviteternas inträffande. Detta förfarande möjliggjorde god kontroll i efterhand att olika uppgifter verkligen utförts, för de fall programmerarna inte kunde dra slutsatser härom ur resultatutmatningarna.

Dessutom kunde systemets tidsmässiga effektivitet samt dess kommunikation med operatören studeras med hjälp av denna logg.

Förekomsten av en klocka i systemet började även användas för debiteringsändamål. Vid de allra tidigaste datorerna hade separata stämpelur använts manuellt för den rent linjära maskintidsdebiteringen. Detta stämplande innebar givetvis ineffektivt arbete för operatörerna. Systemen konstruerades för att i samband med varje jobbs start- resp sluttidpunkt göra fyra noteringar, som vanligen utmatades via ett hålkort:

- jobbets identifikation
- jobbets "kontonummer"
- jobbets starttidpunkt
- jobbets sluttidpunkt.

Dessa kort bearbetades sedan periodiskt av ett speciellt debiteringsprogram (vanligen en gång i månaden), varvid lämpliga debiteringsunderlag för faktureringen producerades. Fortfarande användes dock linjär debiteringsprincip, med x kronor per utnyttjad maskintidstimme (verkligt förfluten tid) på detta stadium i utvecklingen.

Det bör göras klart att flera av de ovan beskrivna styr- och utnyttjningsprinciperna fortfarande används i dagens system. Flera grundläggande principer för operativsystem introducerades vid denna tid. Senare tiders mer avancerad operativsystemteknik vilar på arbeten och erfarenheter från dessa tidiga system.

Det kan också framhållas att lanserandet av satsvis arbetande system fick stor betydelse för spridning av intresset för automatisk databehandling som helhet. Den ökade systemeffektiviteten var framför allt viktig för körning av små jobb. Inte längre behövde spill- och ställtiderna för sådana jobb bli längre än eller ens av samma storleksordning som själva bearbetningstiderna. Först därmed kunde små jobb bearbetas med acceptabel ekonomi på de större datorsystemen. Emedan de små jobben, oftast på grund av fel, avbrutna kompileringar eller assembleringar, utgör en väsentlig del av ett flertal olika användningsprofiler kunde fältet som helhet dra väsentlig nytta av den ökade effektiviteten hos datorsystemen.

Tiden 1959-1961:

Samspel med maskinvaruutvecklingen

Den viktigaste nyheten på maskinvarusidan, organisationsmässigt sett, under denna tid kan sägas vara utvecklingen av datakanalen. En datakanal kan betraktas som en separat liten "dator", med begränsad kapacitet, avsedd att styra olika in/utmatningsoperationer till perifera enheter. En sådan "bearbetningskunnig" kanal innefattar vanligen en primitiv instruktionsstruktur, räknare, minnesregister samt faciliteter för kommunikation med in/utmatningsenheternas anpassningsenheter (styrenheter). Kanalen hämtar order från samma minne som processenheten och överför data till eller från detta minne enligt dessa order. Sedan exekvering av vissa centralenhetsinstruktioner har medfört selektering av avsedd periferienhet, överföringssätt samt initialvärden för kanalräknaren m m, arbetar kanalen i huvudsak oberoende av centralenhetens vidare verksamhet tills överföringen avslutats. Under överföringen till/från primärminnet "konkurrerar" kanalen dock med processenheten om primärminnets åtkomstcykler - s k "cycle stealing".

Användning av datakanaler möjliggjorde "överlappning" mellan central bearbetning och in/utmatning. Från början kunde detta emellertid endast uppnås från assemblyspråknivå, då FORTRAN och andra problemorienterade språk då normalt saknade möjlighet till ordergivning om asynkrona operationer. Från dessa språk anropades systemrutiner för initieringen av in/ut-verksamheten, varpå den centrala bearbetningen kunde fortsätta, detta förutsatt att de t ex inläsningsbegärda data inte omedelbart behövdes i programmet. Omedelbart före användning av inlästa data utfördes i programmet kontroll på att överföringen var avslutad, eller tillräckligt avslutad, vanligen med hjälp av en kort instruktionsloop. Vid starkt in/utmatningsbundna jobb där möjlighet till överlappning ej beaktats på ett effektivt sätt kunde icke ringa tid åtgå för väntan i denna loop.

Överlappning av in/utmatning och internbearbetning åstadkoms genom buffertteknik. Flera buffertar för varje in/utmatningsfil medförde ökat minnesbehov, vilket dock i stort kompenserades av ökad effektivitet hos datorsystemet.

Den ökade systemeffektiviteten åstadkoms bl a genom införandet av bryt-signalsmekanismer. Hos de tidiga datorsystemen med autonoma in/utmatningskanaler fick en in/utmatningsoperations status, dvs huruvida den var pågående, avslutad eller avbruten på grund av fel, kontrolleras av programmet genom avläsning av vissa givna indikatorer. Man kan påstå att mycket tid åtgick i programmerade "loopar" som testade indikatorers

status. Införandet av brytsignalsmekanismer eliminerade i stort sett detta arbete. Brytsignaler signalerade normalt avslutande av in/utmatning, bandslut, filslut, paritetsfel, overflow/underflow etc. Brytsignalshanteringens utveckling kom att leda till utveckling av principer för hantering av flernivåiga avbrott, icke avbrytbara instruktionssekvenser osv.

Den starkt ökade komplikationsgraden för hantering av datatransporter fick som följd att ansvaret för in/utmatningsoperationernas huvudsakliga administration överflyttades till operativsystemens konstruktörer. Sålunda kom systemprogrammen för styrning av datatransporter att tillmätas allt större betydelse, och operativsystemen (som innefattar dessa systemprogram) växte i omfång och komplexitet. De tidiga operativsystemen hade som en av de väsentligaste funktionella komponenterna just datatransport- och brytsignalshantering.

Uppdelning och skydd av operativsystemen

De standardiserade och frekvent anropade systemrutinerna av ovan nämnd typ var uppenbarligen permanent erforderliga i primärminnet. Systemen var för stora för att i sin helhet konstant befinna sig i primärminnet, medan vissa delar av dem var högprioriterat önskade där. Härmed började en uppdelning av operativsystemen märkas. Det blev därför vanligt att uppdela operativsystem i en resident del (permanent i primärminnet) samt resterande transienta delar, lagrade på det snabbaste av tillgängliga sekundärminnen (till att börja med magnetband, i brist på annat). Bland de transienta operativsystemsdelarna återfanns bl a länkaren, laddaren, översättarna, användarorienterade systemrutiner osv.

Denna uppdelning blev bl a nödvändig då man erfarit att t ex många FORTRAN-program som tidigare fått plats i primärminnet nu visade sig erfordra mer minne än som fanns kvar då operativsystemet "tagit sitt". Här fästes för första gången uppmärksamheten på ett betydelsefullt kapacitetsproblem, det för användarprogram tillgängliga primärminnesutrymmet. I proportion till det totala primärminnet har detta användarutrymme till storlek visat en minskande tendens allt eftersom fler funktioner inneslutits i operativsystemen.

Det blev snart - ur praktisk synvinkel - obligatoriskt för användarna att utnyttja systemrutinerna för hantering av datatransporter. En användare som "arbetade på egen hand" på brytsignalsnivå var otänkbar ur central administrationssynvinkel, och kunde snabbt bringa hela systemet ur fas.

Centraliseringen av in/utmatning, avbrottshantering etc kom även att medföra en del icke förutsedda problem. Tillgången till snabbare maskiner (transistorer började användas som byggstenar vid denna tid), och den ökade systemkapaciteten som följd av de effektivare in/utmatningsfaciliteterna och i viss mån även operativsystemen, var några faktorer som ökade efterfrågan på databehandling. De flesta av de nya kunder som "svärmade kring" datacentralerna var inte längre samma maskinorienterade programmeringstekniska experter som tidigare. De användarvänliga programspråken attraherade i stor utsträckning snabbutbildade programmerare som visade sig ha tämligen liten respekt för systemen i datorerna. Maskinerna utsattes för en ur programsynpunkt omild behandling, med frekventa systemsammanbrott som följd. Inte sällan kunde en kund läsa in hela jobbinmatningsbandet i tron att detta var hans/hennes data, eller fastna i långa icke avsedda interna beräkningsslingor utan att operatören, som ej kände programbeteendet, kunde ingripa. Ej sällan producerades mängder av irrelevant information på radskrivarna, som följd av olika programmeringsfelaktigheter. Nödvändigheten av en från systemens sida utövad kontroll och styrning av användarprogrammen blev uppenbar.

Även operativsystemen själva behövde skyddas. Det var till att börja med möjligt för en användare att av misstag skriva över delar av den residenta systemdelen i primärminnet, med break-down som given följd. I brist på maskinvarumässigt minnesskydd fanns ingen helt tillförlitlig metod för att klara dessa problem. Att från början kontrollera samtliga användarprograms totala adressområde var nära nog omöjligt, då absoluta adresser t ex kunde läsas in som data.

En kontroll som dock var möjlig, och snart anammades, var bevakning av systembanden. Använda bandenhetsnummer och läs/skrivoperationer fick endast förekomma avseende "tillåtna" perifera enheter. En icke tillåten begäran om datatransport resulterade här i utmatning av felmeddelande samt avslutning av jobbet.

In/utmatningen kunde begränsas av systemet till tänkbara dimensioner, och även använd beräkningstid kunde med hjälp av den maskinvarumässigt inbyggda elektroniska klockan begränsas. Jobbinmatningsbandet kunde också kontrolleras mot oavsiktligt lång läsning genom att vid dess skapande ett brytsignalsgenererande märke (filslutsmärke) inskrevs efter varje jobbslut, samt att läsning av detta band ständigt ombesörjdes av en speciell systemrutin.

Man tvingades alltså gradvis i systemen bygga in allt fler kontroller och säkerhetsmekanismer för att förebygga "urspåringar". Det alltmer rest-

riktiva behandlandet av användarprogrammen, och dessas upphovsmän, kan sägas innebära början till en utveckling fram emot dagens paradoxalt nog ofta mindre användarvänliga operativsystem. Det torde vara en allmän förhoppning att denna linje brytes för 70-talets operativsystem, i och med den ökade förståelsen för förskjutning mot värdering av systemkontruktionstid, som ett komplement till fixering vid rena maskintidskostnader.

Underhåll av systemen

Uppdelningen av ett operativsystem i en resident del samt ett flertal yttre transienta delar skedde naturligen på det viset att i den residenta delen lades de frekvent erforderliga operativsystemsfunktionerna, av typen rutiner för brytsignalshantering, datatransportstyrning, processortidsadministration, tidtagning etc. Den residenta delen blev naturligtvis mycket känslig för påverkan. Minsta avsiktliga eller oavsiktliga ändring i dess minnesceller kunde få allvarliga verkningar.

De residenta rutinerna behövde korrigeras och uppdateras liksom övriga systemdelar. Normalt var det då nödvändigt att assemblera om hela systemet varje gång en ändring behövde göras, oberoende av ändringens omfång. (De centrala delarna av operativsystemen skrevs då, och skrivs fortfarande med få undantag, i assemblykod.) En annan lösning hade samband med att lätta något på kopplingen mellan systemets olika delar, och dynamiskt länka varje avgränsad systemkomponent till de residenta rutinerna vid varje anrop. Båda dessa alternativ hade nackdelar: den första var tidskrävande vid varje systemmodifikation, och den andra medförde längre spilltider (overhead-tider) vid normal jobb-bearbetning. En kompromiss mellan de båda nådde spridning: den residenta delen försågs med standardiserade ingångar och systemdatafält, som var fixerade och oberoende av systemmodifikationer. Härvid kunde ändringar införas i den residenta delen (omassemblering) utan att de till utseendet fixa systemdatafälten berördes. Ingångarna (i systemdatafälten) innehöll ovillkorliga hopp till anropade rutiner. Med denna kompromiss krävdes sålunda normalt inga ändringar av systemets yttre delar i och med en erforderlig omassemblering av den residenta delen.

Ett liknande förfarande möjliggjorde för samtliga (även de transienta) systemkomponenter att kommunicera med varandra via fixa kommunikationsareor inom den residenta systemdelen. Härmed kunde t ex kompilatorer återlämna kontrollen till styrprogrammet via en speciell återhoppsadress och därvid sätta lämpliga idikatorer som t ex meddelade om översättningen genomförts framgångsrikt. Dessutom underlättade denna teknik inkorporering av nya språköversättare i systemet, vilket naturligtvis var ett önskemål hos många installationer.

En modulär organisation av operativsystemen blev vanlig, som en konsekvens av bl a ovan nämnda resonemang. Det blev obligatoriskt att in/utmatningsfunktioner och andra systemfunktioner, speciellt språköversättare, kommunicerade inbördes medelst standardiserade "kopplingsrutiner" (interface routines), en teknik som tillämpas även för dagens system.

Styrspråk blir erforderliga

Omfattningen av de införda operativsystemsfunktionerna krävde utveckling av adekvata metoder för deras utnyttjande. Till att börja med erfordrades för varje speciell funktion från användarens sida tillhandahållande av ett antal specifikationshålkort eller styrkort. Vid närmare betraktande måste dylika styrkort rubriceras som de första ansatserna till en helt ny språktyp, styrspråk. Omfattningen och flexibiliteten hos detta för varje maskintyp speciella språk stod till att börja med i viss proportion till själva systemets flexibilitet.

Styrspråken växte i omfång. Långsamt började det stå klart att uppmärksamhet borde riktas mot logisk konstruktion och användarvänlighet hos dessa språk. Styrspråksinformation är huvudsakligen avsedd för operativsystemet, och i mera begränsad omfattning för översättare o d. Enhetligheten och utformandet av styrspråken var inte utsatta för lika tungt intresse från användarnas sida, då allt färre användare hade full inblick i systemens uppbyggnad och funktionssätt. Man accepterade (resignerat) kraven på komplicerad styrinformation. Och detta trots att styrspråket var det enda språk som för ett givet datorsystem var nödvändigt för samtliga användare.

Utvecklingen av styrspråken har följt ett relativt långsamt tempo, som knappast står i proportion till den maskinvarumässiga eller den programvarumässiga utvecklingen i övrigt. I flera fall förefaller styrspråken helt enkelt ha "blivit till" i efterhand, då systemet för övrigt färdigställts. Detta har inneburit ökad belastning för användarna, och sannolikt sämre systemutnyttjande än datorsystemen varit värda.

En viktig punkt att ta hänsyn till vid styrspråkskonstruktion är språkets användarvänlighet. Enkla fel av typen "ett mellanslag för mycket" i styrinformationen borde kunna upptäckas och rimlighetstestas och inte resultera i oavsedd jobbavslutning, avsevärt ökad körtid, onödiga objektkortbuntar etc. De tidiga systemen, såväl som många senare, har lämnat en hel del övrigt att önska beträffande bl a styrspråkens karaktäristika i dessa avseenden.

Det begynnande 60-talets operativsystem krävde styrinformation i form av före och inom ett jobb placerade styrkort med tämligen rikt facetterad information. (Dagens situation torde vara värre, men systemen har också blivit väsentligt mer komplicerade.) Styrkorten inleddes med ett speciellt avpassat tecken i hålkortskolumn 1, t ex ett dollartecken ($), för att indikera att kortet är ett styrkort, och inte programinstruktioner eller data. Dollartecknet indikerade även att kortet skulle tolkas (interpreteras) av operativsystemets jobbövervakare, varefter åtgärder rörande jobbets vidare öden inom datorsystemet skulle vidtas.

En väsentlig del av den inledande styrinformationen användes för debiteringsändamål. Man "beställde" där, eller bokade där in, önskade systemresurser, av typen beräkningstid (maximal), minnesutrymme, perifera enheter etc. Denna resursbehovinformation avsågs bl a motarbeta oavsiktligt ineffektivt eller onödigt systemutnyttjande, men ställde i många fall krav på användaren som han/hon hade svårt att leva upp till. Under ett programs inkörning, i uttestningsfasen, var det många gånger svårt, eller ogörligt, att fastställa vissa av programmets resursbehov. Detta gällde minnesbehov och maximal beräkningstid. En följd blev att användaren blev tvungen att ta till i överkant, för att inte riskera oavsiktligt jobbavslutande på grund av otillräckliga systemresurser. Dessa överbud kom att medföra mindre gott systemutnyttjande, i och med t ex längre residens inom systemet än avsett i händelse av tidigt uppdykande av mindre och icke-katastrofala programfel. Även jobbplanering försvårades härav.

För situationer som inte kunde bedömas på förhand tillhandahöll systemen normalt vissa standardvärden för systemresursparametrarna rörande "normalt" resursbehov. Fastslagande av dessa standardvärdens storlek, vilket till att börja med utfördes av tillverkaren, men senare av driftspersonalen, kunde i någon mån påverka datorsystemets totala kapacitet, i form av t ex antal bearbetade jobb per dag.

Efter de inledande styrparametrarna för ett jobb krävdes beskrivning, med "deklarations-styrkort", av de följande program- och datakorten. Koder behövde anges, befintlighet för program och data omtalas, hålkortsformat definieras osv.

Även här kunde utelämnande av viss information innebära att standardparametrar avsågs. Dessutom krävdes att man angav i vilken form resultatutmatningen skulle ske (papper, hålkort, hålremsa, band etc), samt i vissa fall maximala värden för utmatningsvolymen, t ex max. 50 sidor, max. 1000 kort osv.

Av vissa system krävdes ett avslutande styrkort, för att definiera jobbets slut. Andra system kunde själva draga den (till synes naturliga) slutsatsen att ett jobb var avslutat när nästföljande jobbs inledande identifikationskort påträffades, varvid kontrollen gavs till jobbövervakaren för inplacering av jobbet i in-kön (ännu normalt på magnetband). Endast ett avslutningskort för "sista jobbet" erfordrades då (om begreppet "sista jobb" kunde definieras, ex. slut på en sats av jobb).

Hantering av all denna styrinformation kom sålunda att innebära icke obetydliga krav på systemanvändarna. Mot slutet av 60-talet har röster gjort sig hörda med krav på ur totalsystemssynpunkt (användare + datorsystem) mera lämpligt konstruerade styrspråk.

Det fundamentala resultatet från systemutvecklingen under dessa år kan sägas vara en medvetenhet om att ett datorsystem inte endast bestod av maskinvara. Det system, som användaren betjänade sig av, hade även programvarukomponenter, vilkas egenskaper kunde vara av avgörande betydelse för ekonomi och framgång för utnyttjandet av maskinresurserna.

Tiden 1961-1964:

Skyddsmekanismer

Antalet datoranvändare ökade nu i allt snabbare takt. Det blev därför nödvändigt med säkrare skyddsanordningar inom systemen, både mellan olika användare och mellan användarna och operativsystemen.

En lösning på problemet med "oändligt loopande" program uppenbarade sig i och med införandet av den programmerbara avbrottsklockan. Denna bestod av ett internt register, vars innehåll, utgående från ett inplacerat startvärde, ökades med en tidsenhet per verkligt förfluten dito, tills s k "spill" erhölls. (Alternativt fanns klockor som minskades ner mot noll.) Vid spill genererades en avbrottssignal, som sändes till centralenheten, där sekvenskontrollen överfördes till övervakaren. Klockan kunde både initialiseras och avläsas från användarprogramnivå. Även denna servicefunktion kunde emellertid till att börja med missbrukas. Själva klockmekanismen behövde skydd mot otillåten access, och det var först när sådant (via styrprogrammet) tillhandahölls som dess existens nådde väsentlig betydelse för systemen.

I avsikt att gardera mot otillåten skrivning i operativsystemens residenta delar lanserades under denna tid maskinvarumässigt minnesskydd. Denna viktiga funktion arbetade på olika system enligt skilda principer:

- register innehållande övre och undre gränser för tillåtna minnesadresser för användarprogram

- register med information om skyddsstatus för vart och ett av minnesblock av fix storlek (för de fall minnet var blockindelat).

- en "flagga" lagrad i varje tillåtet ord eller teckenposition.

Dessutom blev det så småningom möjligt att för vissa system konstruera sådana faciliteter som möjliggjorde specifikation om skydd av vilken kombination som helst av läs/skriv/exekvera avseende ett minnesutrymme. Ett försök att nå access till otillåtet minnesutrymme resulterade i att kontrollen via brytsignalsystemet lämnades till operativsystemets residenta styrprogram.

Fullständig säkerhet kunde emellertid ännu inte uppnås. En användare kunde fortfarande modifiera skyddsregistren. Dessutom fanns för vissa system möjlighet att undvika de i operativsystemet tillhandahållna standardmässiga rutinerna för in/utmatning.

Direktminnen och systemdecentralisering

Några år in på 60-talet hade så mycket erfarenhet av tidigare operativsystemsprinciper samlats att den fortsatta utvecklingen i flera avseenden kunde förutspås. Man väntade dock på maskinvarumässig möjlighet att med acceptabel driftsekonomi kunna realisera sina idéer. Det var framför allt billigare sekundärminnen som erfordrades. De använda snabba trumminnena kunde, med få undantag, normalt inte tillhandahållas med önskat förhållande lagringskapacitet/pris.

Man torde kunna påstå att skivminnena kom att innebära en vändpunkt i detta avseende. Dessa kunde till acceptabelt pris användas för lagring av stora datamängder med rimliga accesstider. I detta sammanhang bortser vi från den ökade flexibilitet som detta kom att innebära för användarna, och koncentrerar oss på skivminnenas samspel med operativsystemen. På skivminne kunde lagras stora systemdatamängder, med god accessekonomi som ett avgörande kriterium. Antalet systemprogrammoduler kunde ökas i tillräcklig grad, utan att väntetiderna för att läsa in dem för exekvering blev tillnärmelsevis av samma storleksordning som för bandorienterade system.

Ytterligare kompilatorer, interpretatorer, serviceprogram m m inkorporerades i systemen. Även översättare som knappt dagligen användes, av typen SNOBOL, SLIP, IPL etc, lagrades i den till omfånget alltmer ökande systemfilen på direktminne. Operativsystemens struktur förändrades även. Den residenta primärminnesdelen kunde anropa styrprogrammoduler i skivminnet, som efter laddning till primärminnet kunde anropa ytterligare transienta moduler osv. Även subsystemen började sålunda konstrueras och uppbyggas enligt modulär filosofi.

I enlighet med de tidigare erfarenheterna från utnyttjandet av primärminnet, avsattes en "systemets slask-fil" (system scratch file) på direktminne, via vilken kommunikationen mellan de externt belägna subsystemen styrdes. Denna fil kom även till användning som arbetsutrymme för viktiga hjälpprogram, kompilatorer etc. Givetvis erfordrades fortfarande en kommunikationsarea även i primärminnet, men denna kunde nu minskas något i omfång.

Direktminnets lagrings- och åtkomstmöjligheter lämpliggjorde som nämnts en modulär systemuppbyggnad. Det förefaller dock sannolikt att man vid systemkonstruktion ofta bortsåg från att även direktminnesaccesser kräver tid. Den administrativa "spilltid"[1] som dagens operativsystem kräver härrör ofta från omfattande sökningar i och manupuleringar med datamängder på direktminne. Uppmärksamhet mot detta har riktats först på senare år.

Tillgången till direktminne ledde bl a till att debiteringstekniken kunde förbättras. De reformer som genomfördes med direktminnet som bas innebar bl a att, utan alltför allvarliga spilltider som följd, varje identifikationsstyrkort kunde kontrolleras mot en tabell över tillåtna kundidentifikationer, samt att begärda resurser inte skulle innebära kostnader som översteg de för kunden tillgängliga reserverade penningmedlen.

Direktminnen kom således att få stor betydelse för systemens utbyggnad. Vi bör emellertid komma ihåg att det till att börja med, i många fall, endast var en praktisk möjlighet att förverkliga tänkta systemidéer som därmed erbjöds. Den principiella och teoretiska utvecklingen av operativsystemen hade redan vid denna tid nått längre än vad de förverkligade systemen praktiskt kunde uppvisa. Detta är ett förhållande som fortsatt under hela 60-talet. De praktiskt tillhandahållna systemen kan med avseende på flexibilitet, användarvänlighet och effektivitet sägas ligga klart efter dagens teoretiska betraktelser.

---

1) spilltid = administrationstid (overhead).

Systemuppdatering

Komplexiteten hos de operativsystem som vid denna tid utvecklades var sådan att erforderlig tid för underhåll och uppdatering av systemen blev betydande. Denna tid blev en faktor att räkna med vid betraktande av systemtillgänglighet. Det var inte längre praktiskt lämpligt att företa en komplett systemrekonstruktion, ens från objektprogramnivå, vid inkörning av en ny systemkomponent. I stället uppdaterades/reviderades "masterversionen" av systemet, sparad på band eller skiva, i en separat process kallad systemredigering. Trots att endast en begränsad felsökning/korrigering rörande systemet kunde äga rum under normal daglig körbelastning var det i alla händelser nödvändigt med grundlig utprovning av varje ny systemkomponent i sin slutgiltiga användningsmiljö innan den kunde frisläppas generellt. Den större vikt som successivt kom att fästas vid försiktighet och vaksamhet rörande nyinförda systemkomponenter hade samband med de omfattande och besvärliga konsekvenserna av systemkollaps, när detta råkade inträffa.

Hos vissa användare blev det vanligt att avsätta standardmässig tid för datorsystemen inte bara för maskinvarukontroller, utan även för kontrollåtgärder rörande operativsystemen. Tid på dygnet då inga användarjobb tilläts belasta systemet användes för systemtester o dyl.

En metod som prövades för att hålla nere servicetiderna innebar tillhandahållande av en temporär editeringsfacilitet som kunde användas av systemprogrammerarna i den normala jobbströmmen. Denna (dynamiska) editeringsteknik använde sig av metoden att upprätta temporära systemkopior, vilkas livslängd endast täckte det under bearbetning varande användarjobbet. Härefter återgicks direkt och automatiskt till den tidigare, och uttestade, systemkomponentversionen. På detta sätt undgick man många gånger total systemkollaps (där sådan egentligen borde inträffat) och förstörde eventuellt endast det jobb som var under behandling, kanske ett speciellt användarjobb av testtyp. Tekniken visade sig speciellt värdefull för direktminnesorienterade system. Den kunde dock inte utan implementeringskomplikationer tillämpas på de djupast liggande systemdelarna, av typen de primärminnesresidenta rutinerna, vilkas funktion var av avgörande betydelse för systemet som helhet. Inte heller kunde man med framgång använda densamma i multiprogrammeringsmiljö, då flera samtidigt under bearbetning varande program kunde förstöras av ett temporärt systemfel. Icke desto mindre var tekniken intressant, och till sin konstruktion avspeglar den den allt större betydelse som fästes vid systempålitlighetsproblemen och flexibilitetskraven.

Multiprogrammering

Ett principiellt avgörande skede i utvecklingen inträdde i och med implementering av multiprogrammeringssystemen. Detta kunde visas upp i produktionsmässig form på några system redan omkring 1961.

Ett datorsystems användning av multiprogrammeringsteknik innebär att fler än ett användarprogram vid en given tidpunkt befinner sig under bearbetning i systemet. Observera betoningen på användarprogram vid denna "definition". Redan tidigare kunde exekvering av ett användarprogram, styrt av ett operativsystem (vilket senare då även kan sägas befinna sig "under bearbetning"), med en generellare definition än den ovan givna sägas vara exempel på multiprogrammering.

Med multiprocessing avses normalt att flera användarprogram eller tidsmässigt parallella delar av ett program samtidigt skall befinna sig i bearbetning av två eller flera processorenheter. För realisering av multiprocessing inom ett datorsystem krävs alltså fler än en central processorenhet i systemet. Av dels systemkomplexitetsskäl och dels ekonomiska skäl har multiprocessor-system ännu inte kommit till användning i större skala: operativsystemen har först på ett relativt sent stadium i utvecklingen kunnat levereras med funktioner för styrning av flera processorer. I vidaste bemärkse skulle aktiviteten i en eller flera datakanal(er) samtidigt som en central programexekvering kunna sägas vara exempel på multiprocessing, men normalt syftas med begreppet att flera användarprogram i primärminnet (eller i flera primärminnen inom ett och samma datorsystem) skall befinna sig under exekvering samtidigt, med hjälp av två eller flera processorer, kopplade till primärminnessystemet. Via datakanaler hopkopplade skilda datorer (se nedan) brukar normalt inte betecknas som multiprocessingsystem. (Vi kan konstatera att begreppet multiprogrammering kan sägas innefatta begreppet multiprocessing, multiprogrammering kräver dock inte multiprocessing.) Ett "normalt" multiprogrammeringssystem är idag konfigurerat med endast en processor, och växlingen mellan program i minnet administreras av det residenta styrprogrammet.

Ett ursprungligt skäl för införande av multiprogrammering var att möjliggöra simultant utnyttjande av så många perifera datorenheter som möjligt sålunda undvikande situationer där den dyra processorenheten befinner sig stillastående i avvaktan på avslutandet av en datatransport. I en sådan situation skulle processorn i stället fortsätta bearbetningen med ett annat i primärminnet befintligt program.

Införandet av multiprogrammering kom vid satsvis bearbetning att medföra ett bättre, mera jämnt fördelat maskinutnyttjande än tidigare, och kortare omloppstider för en sats av användarprogram erhölls.

Ett av hjälpmedlen för att erhålla korta omloppstider, framför allt för korta program med hög angelägenhetsgrad, är användning av s k checkpoints. Redan vid 60-talets början hade dessa använts i konventionella uniprogrammeringssystem (motsats till multiprogrammering, endast ett användarprogram bearbetas åt gången), men de kom nu till vidare bruk. En check-point är ett läge i ett användarprogram, inplacerad antingen av programförfattaren eller av operativsystemet. När exekveringen når fram till denna punkt avbryts exekveringen och speciella åtgärder kan vidtagas, manuellt (användarprogrammerarstyrt) eller operativsystemstyrt. På "tidiga" datorer (slutet av 50-talet i detta fall) erfordrades åtgärder av operatören vid vissa av dessa check-points. En åtgärd att vidtaga vid en check-point är t ex att undanlagra dittills erhållna delresultat och att ge kontrollen åt något annat program i systemet. I fullständiga fall kan undanlagring av hela programmet, inklusive delresultat, hjälptabeller, statusinformation osv avses. En checkpoint funktion är oftast värdelös utan möjlighet att vid lämplig senare tidpunkt kunna återuppta verksamheten där denna avbröts. Man talar därför oftast om begreppet check-point/restart.

Check-point/restart kom till användning i multiprogrammeringssammanhang bl a genom att operativsystemet kunde avbryta exekveringen av långa program, för att ge plats för till omfånget mindre program som krävde korta omloppstider. - En på sätt och vis egendomlig demokrati har dominerat och dominerar än datorsystemens styrning (från operativsystemen och driftcentralerna): långa, tidskrävande jobb anses ofrånkomligt kunna vänta på kortare. Denna värdering innebär att kortare jobb betraktas som "viktigare" än långa. Ett flertal aspekter kan läggas på denna uppfattning. Varje programs "betydelse" bör ses i samband med det problem det löser, den manuella kapacitet och kostnad som är inkopplad, datorn är för det mesta endast ett hjälpmedel vid problemlösningen.

Perifera funktioner och kopplade system

En typ av multiprogrammering som nådde vid spridning, innebär perifert in/utmatningsarbete samtidigt som sekundärminnesorienterad bearbetning. Den normalt stora mängd mer eller mindre rent mediakonverteringsarbete, på sin höjd inkluderande dataredigering o d, som är vanlig vid de flesta installationer, har motiverat framtagande av speciella,

standardiserade mediakonverteringsprogram, med varierande benämningar (SPOOL-program, symbionts, parasitrutiner osv). Dessa program, som på grund av sina speciella och begränsade funktioner normalt belägger endast en liten mängd primärminne, behandlas i vissa fall speciellt av operativsystemet (ges hög prioritet), eller ingår i andra fall som "normala" jobb i jobb-kön.

Typisk verksamhet för mediakonverteringsprogram (MK) är konvertering av data från t ex kort till band, från kort till skiva, från band till radskrivare osv, över huvud taget konvertering mellan två perifera maskinenheter i datorsystemet. Multiprogrammering i ett datorsystem ledde bl a till att intresse knöts till prioriteringsproblem. En konvertering till/från en låghastighetsenhet, t ex kortläsare, radskrivare, bör ges hög processprioritet för att inte innebära alltför lågt utnyttjande av den perifera enheten. Emedan normalt varje för en delkonvertering erforderlig processtidsdel är mycket kort, kan denna verksamhet prioriteras utan alltför kraftig förlängning av samtidigt pågående processbundna jobbs körningstider. Vid varje avbrottssignal från MK-rutinen, t ex när en buffert har fyllts och behöver tömmas, tilldelas MK-rutinen erforderlig processtid varefter kontrollen återlämnas till ett annat lägre prioriterat jobb i systemet (via styrsystemet). Bl a hanterande av prioriteringsproblem i operativsystemen innebar ytterligare tids- och andra resurskrav för systemen själva. Hantering av MK-program simultant med ett användarprogram innebar dock bara begynnelsen till realisering av mera komplicerade prioriteringsalgoritmer.

En konfigurering av datorsystem som medförde ett multiprogrammeringsliknande arbetssätt var sammankopplingen av två datorer, varvid den ena bl a ombesörjde all in/utmatningsadministration åt den andra. Sammankopplingen skedde vanligen med någon fast höghastighetsförbindelse mellan de båda datorernas primärminnen. Denna typ av systemkonfigurering, vanlig främst för IBM-system (t ex IBM 7094/40 och, senare, IBM 360 ASP (Attached Support Processor) med 360/50-40 eller 360/65-40 etc), kan ses som en direkt utvidgning och effektivisering av IBM 7090-1401-systemen. Vid dessa sistnämnda hade den separata 1401-an skött all in/utmatning, och via manuell bandförflyttning (eller bandstationsnummerbyte mellan de två maskinerna) kommunicerat med 7090. Så länge som 1401-an kunde "ligga före" 7090 och mata denna sekvensiellt med väntande bandjobb kunde ett ökat utnyttjande av den större datorn uppnås.

ASP-systemen integrerade denna 7090-1401 teknik, bl a genom att inte längre kräva manuella handgrepp för kommunikationen mellan de båda systemen. I ett ASP-system handhar den långsammare datorn förutom in/utmatning även system- och vissa användarfiler samt är ansvarig för den huvudsakliga styrningen för det kopplade systemet som helhet. Om vi jämför med de tidiga satellitprocessorsystemen ersätter den långsammare datorn i ett ASP-system väsentligen in/utkanalerna, satellitdatorn och en del av den större datorns operativsystem. En typisk ASP-konfiguration innebär att den långsammare datorn förutom primärminne är utrustad med t ex

- en snabb kortläsare
- remsutrustning
- en eller flera radskrivare
- direktminne
- ett flertal bandstationer
- en operatörskonsol

medan den snabba datorn förutom primärminne-primärminne-förbindelsen endast har sekundärminnen för större användarfiler anslutna. Denna teknik visade sig kunna öka vissa systems arbetskapacitet i sådan väsentlig grad att det på senare större IBM-360-system av en-processor-typ förekommit att man inom systemet programvarumässigt simulerat den långsamma datorn, för att få möjlighet tillämpa av leverantören tillhandahållen standardiserad ASP-teknik.

ASP-systemen ansågs av IBM möjliggöra ett bättre systemutnyttjande i stort än en-dator-system, men var ekonomiskt svårare att motivera då en extra processor inklusive primärminne normalt krävdes. Dessa system kan ses som början till utveckling av datornät, där flera processenheter samt varierande perifer utrustning delar upp arbetet sinsemellan för att öka den totala effektiviteten.

Tiden 1964-1969:

Residenta filer

Som tidigare nämnts har 60-talets senare hälft inneburit lansering av relativt få nya huvudprinciper. I stället har tyngdpunkten lagts på förfining och utvidgning av tidigare existerande sådana. Däremot har de senare åren praktiskt kunnat visa upp systemprinciper som tidigare existe-

rat endast i dokumentform. Dessutom kan sägas att kring 1970 sådana operativsystemsfunktioner kan erhållas för styrning av små till medelstora datorer, som tidigare endast fanns realiserade på de större datorsystemen. Maskinvarans utveckling, t ex rörande avbrottssystem, in/utkanalsystem m m har varit en faktor som legat bakom denna sistnämnda utveckling.

Ett viktigt steg framåt har tagits i och med ökade möjligheter för data- och programfiler att residera permanent i ett datorsystems sekundärminnen. Detta innebär att användare kan bygga upp filer för rutinanvändning med endast begränsade krav på egna filkataloger m m. Filhanteringssystem som är speciellt viktiga i reelltids- och time-sharing-miljö kan med fördel nyttjas även för mera konventionella användningsområden.

Ett system för manipulering av residenta filer skall naturligen tillhandahålla en associerings- eller kopplingsmöjlighet mellan användarprogrammets filbeteckningar och de tidigare i systemet lagrade fysiska filerna. Access skall kunna medges såväl till användarens egna filer som till filer av extern natur (t ex systemfiler, andra användares filer osv). Olika metoder används härför:

1) Styrspråket kan innehålla möjlighet till associering av programmets filbeteckningar till de fysiska filbeteckningarna.

2) Om programfilbeteckningar är symboliska, i motsats till numeriska som i t ex FORTRAN, kan dessa i vissa fall referera direkt till de fysiska filerna.

3) Olika översättningsmöjligheter kan realiseras för att kunna ge nya namn åt de fysiska filerna, passande till programmets logiska filbeteckningar.

Styrningen av filbeteckningarnas betydelse vid varje tidpunkt kan ombesörjas av speciella systemrutiner som även identifierar de sökta filernas lägen i systemet. Samtliga metoder ovan kan vara av intresse även inom ett och samma datorsystem.

Ett viktigt, i många fall avgörande, filhanteringsproblem har samband med filskydd. Å ena sidan måste systemet kunna garantera skydd mot otillåten läsning eller skrivning i användarfiler (privata filer måste kunna existera), å den andra bör en användares filer vara lättåtkomliga för de personer som verkligen är auktoriserade. Sålunda måste speciella instruktioner kunna ges rörande

- endast läsning tillåten
- endast skrivning tillåten
- total otillgänglighet
- fullständig tillgänglighet (t ex slask-filer, arbetsfiler)
- fleranvändartillgänglighet (simultant)
- access endast tillåten till speciella filgenerationer osv.

För att lösa dessa problem konstrueras nyckel-system, dvs system kring de koder som styr filers, posters eller termers tillgänglighet. Tillförlitligheten hos dessa nyckelsystem är dock långt ifrån 100-procentig.

Problemen kring filskydd tillmäts numera en ökande betydelse, speciellt med tanke på de centrala databankar som kan väntas realiseras inom en snar framtid. T ex måste persondata ovillkorligen kunna skyddas, bl a för att bevara samhällsmedborgarnas integritet. Varje individ måste dessutom ha tillgång till merparten av de data som berör henne själv.

Systemskydd

Den fasta styrningen av flödet av användarprogram genom datorsystemen sammanhänger med skyddsåtgärder inom systemen själva. Strikta "förhållningsorder" finns numera inbyggda i de flesta större system. Ett exempel härpå är införandet av s k priviligierad instruktionsnivå (master mode instruction level) i motsats till användarprogramnivå (slave mode). Datorsystemets instruktionsrepertoir indelas härvid i två klasser, master-instruktioner och slav-instruktioner. Distinktionen mellan dessa två ligger däri att från ett användarprogram endast slav-instruktioner kan initieras. För att kunna utföra även en master-instruktion måste datorsystemet först försättas i "master-mode", priviligierad arbetsnivå, den nivå på vilken operativsystemet arbetar. De flesta instruktioner är slav-instruktioner. Typiska masterinstruktioner kan vara:

- initiera en in/utmatningsoperation
- modifiera minnesskyddsregistret
- modifiera debiteringsregistren osv.

Från slav-programnivå kan inga master-instruktioner utföras utan "anhållan" om hjälp med att utföra t ex en in/ut-operation måste först ges från användarprogrammet, "slaven", till operativsystemet, "mästaren".

Vi har i tidigare avsnitt diskuterat skäl bakom detta in/ut-förfarande (bl a standardiserade in/ut-rutiner). Vid begäran om in/utmatning eller någon annan fundamental systemfunktion anropar användarprogrammet styrsystemet (en brytsignalsgenererande instruktion, s k supervisor call, utförs), användaren "mister kontrollen", systemet försätts i master mode, och utförande av systemfunktionen initieras (eller köas). Efter utfört arbete övergår systemet åter i slave mode, användartillstånd, varvid exekveringen fortsätter, ofta med något annat användarprogram (multiprogrammering). När t ex den begärda in/utmatningen utförts, genereras alltså åter en avbrotssignal, och det aktuella programmet läggs i kö för att erhålla processorkapacitet och fortsätta exekveringen i enlighet med internt prioriteringsschema.

Växlingarna mellan privilegierat och användar-tillstånd sker normalt snabbt, på en eller ett fåtal cykeltider.

Det är först med maskinvarumässig hjälp principer av ovanstående komplexitetsgrad har kunnat realiseras utan alltför stort tidsspill. Master/slave-tillstånd existerar således med hjälp av speciell maskinvara, vilken utvecklats och realiserats först kring mitten av 60-talet (i något undantagsfall tidigare).

Fördelning av processortid

Programavbrottsteknik har kommit till fundamental användning i samband med fördelning av processortid mellan till primärminnet inladdade program under simultan behandling. I syfte att åstadkomma så hög belastning på olika systemresurser som möjligt genereras i många system avbrott ingalunda blott vid begäran om in/utmatning eller andra privilegierade instruktioner. Hos mer avancerade system existerar en mängd skilda avbrottstyper. Man kan skilja på flera olika huvudprinciper för att fördela processortid mellan under behandling varande program i primärminnet:

a) ett program på given prioritetsnivå tillåts konsumera processortid till dess att det frivilligt via styrsystemet överlåter kontrollen till annat program (på vid aktuell tidpunkt högsta prioritetsnivå).

b) ett program på given prioritetsnivå avbryts av ett annat högre prioriterat program om detta vid tillfället erfordrar processortid.

c) processortid tilldelas de under behandling varande programmen i mindre, fixerade "doser", s k tidskvanta, varvid varje program för att utföra avsett arbete behöver ett eller (normalt) ett flertal kvanta.

Variationer av ovanstående principer är givetvis vanliga. Här närmast kommer vi att något diskutera metodik som ansluter sig till fördelningsprincipen c) ovan. Normal storleksordning på ett kvantum (som är att betrakta som en systemparameter) kan röra sig om decisekunder till några sekunder.

Denna spridning av resursen processortid mellan olika köande program används sålunda för att undvika att ett visst program under bearbetning "belägger" processorn för en orimligt lång tidsperiod. I time-sharing-miljö och i viss mån även i reelltidsmiljö är en dylik utjämning av processortid av fundamental betydelse för att hålla responstiderna för korta jobb (transaktioner) vid rimlig nivå. Vid system av detta slag kan högprioriterade inkommande jobb eller transaktioner medföra att något eller några andra vid tillfället i primärminnet varande program tvingas överföras (swappas) till ett sekundärminne för att bereda plats för de program som utgör/skall ta hand om de inkommande jobben/transaktionerna. Vi talar då om begreppet Roll out/Roll in, lägre prioriterade program "rullas" automatiskt ut på sekundärminne för att, när primärminnesutrymme åter finns tillgängligt samt prioritetssystemet så anfordrar, åter "rullas in" för fortsatt exekvering.

Metoderna för fördelning av processortid genom användning av kvantumbegreppet varierar som nämnts mellan olika system. I enklare system existerar endast en typ av avbrottsgenerering för uppnått tidskvantum: vid initiering av exekvering av ett program (vid programstart eller vid återupptagande av verksamheten efter ett tidigare avbrott) sätts en "äggklocka", tidsräknare, på kvantumtid, som sedan via koppling till datorsystemets interna klocka nedräknas mot noll. Vid noll genereras programavbrott (master mode entry), kontrollen lämnas till lämplig del av operativsystemet för beslut om vilket (normalt annat) program som då skall tilldelas processortid. Härvid läggs det nyss till en del exekverade programmet i väntestatus, tills det nästa gång kommer i tur att erhålla processortid.

Val av lämplig storleksordning på tidsmängden ett kvantum utgör oftast en kompromiss. Ett långt kvantum, på ett flertal sekunder, kanske, medför fördel (i och med korta omloppstider) för program som kräver lång processortid totalt. Å andra sidan förlängs därmed omloppstiderna för korta program, i och med att de får vänta proportionellt sett längre på att erhålla processortid. En kort kvantum-tidsenhet ger bättre omloppstider för korta program, medan spilltiden för operativsystemets programväxlingar och annan administration ökar. Det existerar system där detta problem hjälpts upp genom indelande av användarprogrammen i två skilda köer:

1) en för korta men frekventa program

2) en för mycket processortidskrävande program.

Program i den sistnämnda kön tilldelas långa processortidskvanta, av storleksordningen 1/2 - 1 minut, som kan avbrytas endast av program från den första kön. Dessa senare tilldelas kvanta om blott delar av en sekund. Med ett dylikt förfarande kan drägliga omloppstider erhållas för båda typerna av program och den "administrativa" over-head-tiden hållas nere. Då emellertid jobbprofilerna varierar avsevärt mellan olika installationer, kan generella lösningar på fördelningsproblemen knappast ges, utan anpassning av kvanta etc får oftast ske individuellt.

Medan det existerar system som inte över huvud taget använder sig av kvantumteknik[1)], finns å andra sidan system som har utvecklat denna teknik ytterligare. Ett exempel gavs ovan, två skilda programköer med olika kvanta. Ett annat innebär att ett in/utmatningsbundet program, som erfordrar processortid frekvent, men endast i mycket små doser kan begära startande av varje in/utmatningsaktion med speciell prioritering vad gäller processortilldelning. Efter varje avslutad in/utoperation ges förtur till erhållande av processortid under förutsättning att endast mycket begränsad sådan erfordras innan nästkommande in/utmatning. Härvid begränsas processortiden per tilldelning hårt, storleksordning ett par millisekunder; skulle programmet missbruka förturen och inte vara processor-färdigt efter denna korta tids utgång, bryts exekveringen omedelbart och programmet placeras i normal prioriteringskö (möjligen med låg prioritet som straff). Användning av dylik teknik styrs via in/utmatningsinstruktionerna. Anhållan om "master mode entry" för utförande av in/utmatning sker härvid med explicit angivande av att efter avslutad in/utmatning endast kort processortid kommer att erfordras före nästa avbrott. Ett system som tillämpar denna teknik är General Electric's GECOS II: den kallas där "Courtesy Call".

Prioritering och yttre resurstilldelning

Förfinad teknik har kommit till användning vid ökad automatisering av alla de faser ett program genomlöper inom ett datorsystem. Vi kommer nedan att beröra tilldelning av vissa systemresurser till ett program. Härvid lämnas resursen processortid åt sidan, då denna behandlats i föregående avsnitt. Vi betraktar i stället primärminne och yttre system-

1) Detta gäller många datorsystem för "administrativ databehandling", bl a beroende på vilken datorleverantör som levererat utrustningen.

enheter.

Resurstilldelning styrs i nyare system huvudsakligen med hjälp av programprioritetsbegreppet. Vi har redan berört detta begrepp. Olika typer av prioritet kan ges åt ett program:

- extern prioritet; anges oftast på styrkort som föregår jobbet (på eller omedelbart efter jobb-identifikationsstyrkortet)

- intern prioritet; framräknas av systemet med hänsyn till de resurser som programmet visar sig kräva. Denna interna prioritet kan (oftast) ej explicit påverkas av programmeraren.

Den enklaste externa prioritetsprincipen innebär möjlig angivelse av en enda prioritetsgrad, som komplement till avsaknad av prioritetsangivelse. Härvid kan användaren belastas av en ökad penningtaxa för bearbetningen. Andra system medger användning av en vid skala av externa prioriteter, i förekommande fall från 1 till 10, från 1 till 64 eller från 1 till 99. Debiteringen kan i dessa fall bli betingad av vald prioritet; hög prioritet innebär företräde framför andra program vid fördelning av systemresurser, vilket leder till kortare omloppstider. Dessa högre krav på systemet kompenseras vanligen med högre utnyttjandetaxor.

Intern prioritet beräknas ej sällan med hänsyn dels till den externa och dels till omfattningen av de resurskrav som aktuellt jobb ställer på systemet. Ett jobb som anger låga systemresurskrav, t ex en ren kompilering, kan ges ett förhållandevis högt prioritetsbidrag, medan ett jobb som (på styrkort) anger lång erforderlig processortid samt t ex omfattande utmatning kan ges lägre prioritetsbidrag. Denna metod kan bl a användas för att (via omloppstiden) styra användarna till att icke onödigtvis belägga systemresurser. Vi kan tänka oss den slutliga inre prioriteten som en funktion, t ex summan av den externa och det från resurskraven härrörande interna bidraget. Mer komplicerat beräkningssätt förekommer för flera system. Vissa system har mekanismer som kontinuerligt kan ändra jobbs interna prioritet. Ett jobb som t ex under lång tid fått stå tillbaka för högre prioriterande jobb kan "flyttas upp" till högre prioritetsnivå för att ej få alltför lång omloppstid.

Prioriteten används ibland även i samband med tilldelning av primärminne omedelbart före exekveringsstart. Som ovan nämnts kan prioritet ha väsentlig inverkan på den omloppstid som erhålls.

Tilldelning av resurser till ett jobb (resursallokering) sker teoretiskt sett bl a med hänsyn tagen till existerande resursers "vikt" för systemets

totala bearbetningsförmåga. Man söker styra bearbetningen så att användarna i ökad utsträckning belägger enheter som "normalt" har lägre beläggning. Det är således fråga om beläggningsutjämning, i syfte att få ut så mycket som möjligt av datorsystemet. Prisfrågor kan här spela in. Då primärminnet är dyrt i inköp, ofta ca en tredjedel av hela konfigurationens kostnad, anskaffas detta vanligen endast i begränsad omfattning. Man "tar sällan till" vad gäller primärminnesstorlek, medan detta är vanligare rörande sekundärminne. Det är normalt viktigare för datorsystemets funktion i stort att primärminnet utnyttjas effektivt än att t ex magnetband beläggs till fullo. Magnetbandsutrymme finns normalt tillgängligt i större omfattning än primärminne. Tilldelning av yttre maskinenheter sker därför oftast före tilldelning av primärminne. Situationer kan också uppkomma då ett program under exekvering framställer krav på utvidgade resurser. Dessa krav kan t ex vara beroende av inmatade data eller andra på förhand okända parametrar. Endast få system idag kan emellertid effektuera dylika "beställningar". Normalt är att samtliga resurskrav måste kunna anges före jobbstart.

Ett dylikt "prioritetstänkande" har även för enstaka system slagit igenom vad gäller tidsmässig turordning mellan tilldelning av yttre systemresurser. Tilldelning sker då generellt enligt principen att för systemets totalfunktion väsentligare datorenheter tilldelas efter tilldelning av mindre fundamentala resurser. För en installation med både trum- och skivminne samt magnetband kan tilldelning av systemresurser till ett jobb som kräver någon del av samtliga dessa, enligt ovan lämpligen ske enligt turordningen

1) magnetband

2) skivminnesutrymme

3) trumminnesutrymme

4) primärminnesutrymme

Emedan det successivt visat sig allt viktigare att inte onödigtvis belägga systemresurser har intresse även knutits till realisering av möjlighet att före programexekveringsslut kunna friställa ej längre erforderliga systemresurser. Ledigförklarande t ex av en magnetbandsenhet före avslutad exekvering är exempel härpå. Härvid är en kopplingsmöjlighet önskvärd mellan använt programmeringsspråk och aktuellt datorsystems styrspråk för att friställning av systemenheter skall kunna styras från källprogramnivå. Enklaste princip, varvid detta senaste problem lämnas därhän, är att endast medge möjlighet till frikoppling av enheter i samband med byte från ett jobbsteg till ett annat (detta byte sker i nära nog samtliga fall dirigerat från styrspråksnivå).

Primärminnestilldelning

På grund av de höga krav på effektivast möjliga primärminnesutnyttjande vid multiprogrammering, som antytts i föregående avsnitt, har stort intresse knutits till olika metoder härför. Vi kan här urskilja ett antal avgränsade principer:

1) Metoder utgående från fysisk begränsning av minnet

   a) Statisk, sammanhängande regionindelning.
   b) Dynamisk, sammanhängande regionindelning.
   c) Statisk fragmentering.
   d) Dynamisk fragmentering.

2) Virtuell minnesteknik, dynamisk blockutbytesteknik (paging)

   a) Kravstyrd.
   b) Prognosstyrd.

Om vi betraktar existerande system idag är metoderna under 1) använda i större utsträckning än de under 2).

Sålunda arbetar de flesta systemen med en fix övre minnesgräns (-adress), bestämd av den fysiska minnesstorleken.

Enklaste tilldelningsteknik, 1a), baserar sig på indelning av minnet i ett fixt antal sammanhängande regioner (partitions), till antalet maximerat av det antal program som operativsystemet förmår behandla simultant. Detta maximala antal varierar mellan 5 och ca 15 i aktuella system. I praktiken är en indelning i 3-4 regioner vanlig. Endast ett program[1] kan laddas in i varje region, någon kommunikation över regiongränserna är normalt icke möjlig. Regiongränser fastslås en gång för alla vid ett givet bearbetningsskift, men kan ändras antingen genom ny systemgenerering eller genom viss insats från operatören så att olika regionindelningar kan användas t ex vid olika tider på dagen, beroende på tidsvarierande jobbprofiler och -frekvenser. Enär minnesbehoven för ankommande jobb i de flesta fall inte är kända på förhand för driftsledningen, får regionindelningar väljas som approximerar "normala" behov. Detta innebär ett ej okomplicerat optimeringsproblem. Man torde kunna påstå att ett mindre gott minnesutnyttjande blir följden av använd-

---

1) Ett program kan givetvis vara uppbyggt av en mängd delprogram. Det som här avses med ett program är sett ur multiprogrammeringssynvinkel. Det kan formuleras att vid användning av metoden 1a) multiprogrammering inom en region normalt ej är möjlig.

ning av denna princip. Små jobb får ofta inladdas till regioner anpassade för större jobb, med outnyttjade regiondelar som följd, och stora jobb kan ofta bli utsatta för långa omloppstider i väntan på tillgängligt minnesutrymme. En tänkbar fördel hos denna minnestilldelningsprincip är de jämförelsevis enkla mekanismer som krävs för såväl processortids- som minnestilldelning och planering. Det skulle kunna förmodas att primärminnesanspråken hos operativsystem med denna minnestilldelningsprincip kunde hållas låga. Tyvärr har så inte alltid i verkligheten blivit fallet. Dessutom borde operativsystemets administrativa belastning på datorsystemet i detta fall kunna hållas inom rimliga gränser.

Beträffande program med rörligt minnesbehov, dvs program som expanderar eller imploderar i minnet under exekvering, är principen 1a) mindre lämplig. Expansion kan ej ske längre än till aktuell regions storleksgräns och återlämnat minne (implodering) kan ej utnyttjas av program i andra regioner. Metoden 1a) används bl a av IBM 360 OS: MFT I, MFT II, EMFT.

Metoden 1b) torde innebära ett bättre minnesutnyttjande. Här tilldelas de ankommande jobben sammanhängande minnesutrymmen i "regioner" som (direkt) motsvarar deras uppskattade behov.

Om programmen tilldelas utrymme statiskt, dvs de behåller angiven plats i minnet under hela exekveringen, blir följden att minnet efter viss tid riskerar att "fragmenteras", och lämna en mängd mindre och osammanhängande icke utnyttjade minnesdelar. För att (i varje ögonblick) bereda största möjliga sammanhängande utrymme måste någon typ av "packning och omflyttning" företas. Vid dynamisk tilldelning sker således relokering (flyttning av program i minnet) allteftersom lediga utrymmen uppkommer, från avslutade jobb. Härvid uppkommer, om programmen ständigt söks flyttas "nedåt" i minnet för att fylla igen luckor, sammanhängande ledigt utrymme vid minnets "slut". Metoden 1b) förutsätter att ankommande program i in-kön såväl som i minnet är relokerbara, då ju aktuell placering i minnet för hela exekveringstiden inte är känd vid inladdningen. Denna metods och de följandes nackdel kan sägas vara tids- och utrymmesbehovet för de mekanismer i styrprogrammet som ombesörjer relokeringen.

En stor hjälp vid programrelokering är s k basadressregister, vars innehåll automatiskt adderas till operandadressen före en instruktions utförande. Vid relokering behöver operandadresserna ej ändras, endast innehållet i basadressregistret. - Denna princip för minnesutnyttjande lämpar sig i vissa fall även för dynamiskt rörligt minnesbehov från programmens sida under exekvering, då relokeringsfaciliteter finns

inbyggda. Metoden 1b) används av ett flertal operativsystem, såsom t ex ICL's George, General Electric's GECOS III, Data SAAB's MK-Dirigent, IBM OS/MVT, m fl.

Fragmentering, metod 1c), innebär att tilldelningen sker i form av minnesblock av fix längd, fragment, av storleksordningen 1 K till 4 K tecken. Ett program tilldelas ett erforderligt antal dylika fragment, vanligen fysiskt belägna på olika platser i minnet (de bildar sålunda ett icke sammanhängande minnesområde). Dessa fragment sammanknyts/länkas ihop med hjälp av länktabeller, en för varje aktivt program. De relativa programadresserna modifieras med aktuellt länktabellinnehåll under exekveringen. Ett gott minnesutnyttjande erhålls, outnyttjade utrymmen blir förhållandevis små. Ett program som erfordrar t ex 11 200 ord tilldelas, om blockstorleken är 1 000 ord, 12 block, varav uppenbarligen endast 800 ord ej används. Spillutrymmen kan givetvis uppkomma då antalet lediga block tillsammans ej täcker behovet för det minsta köande jobbet, men allmänt bör metoden 1c) utnyttja minnet bättre än 1b). Det statiska i metoden 1c) innebär att ett program ej kan tilldelas ytterligare block under exekveringens gång. Dynamiken i 1d) innebär att ett program under exekvering kan tilldelas resp återlämna minnesblock.

Virtuell minnesteknik, eller dynamisk blockutbytesteknik, paging, baserar sig på ett dynamiskt utbyte av programblock mellan primär- och sekundärminnena. Aktuella användarprogram indelas i block av motsvarande storleksordning som vid fragmentering, vilka i relativ form samtliga placeras på ett sekundärminne. Då prioriteringsalgoritmen i operativsystemet anvisar klartecken för laddning av ett program A (föregånget av bl a allokering av A:s erforderliga yttre enheter), laddas så många block av A som erfordras för att A skall kunna börja exekveras (ej nödvändigtvis hela A). Vid fallet 2a), kravstyrd blockutbytesteknik, fortsätter exekveringen till den punkt i A där för vidare exekvering av A ett icke till primärminnet laddat block erfordras. Härvid genereras en avbrottssignal och kontrollen lämnas till en systemrutin av karaktären "blockbytesadministratör". Under det att A därefter befinner sig i vila (och processtiden nyttjas av annat program) hämtas de begärda sekundärminnesresidenta A-blocken in till primärminnet. De placeras antingen i ledigt minnesutrymme eller skrivs över delar av A som redan exekverats[1]. När kontrollen småningom återgår till A fortsätts exekveringen med de nyss laddade A-blocken. Emedan A:s programkontrollförlopp givetvis inte är känd för systemet från början, kan t ex begäran om utförande av upprepade hopp-instruktioner leda till ideliga avbrott för framhämtande av icke-residenta delar av A. Trots att denna "swapping" kan överlappas

---

1) Åtskilligt finns publicerat avseende "optimala" metoder att administrera denna utbytesteknik.

med andra programs exekvering är risken ej ringa för mindre gott totalt systemutnyttjande på grund av tidsförluster för programadministration, swapping etc. Realisering av blockutbytesteknik rent programvarumässigt har visat sig alltför ineffektivt, maskinvarumässiga hjälpmedel i form av utökade adresseringsmekanismer har visat sig erforderliga.

Algoritmen i metod 2b) för utväljning av vilka block som från sekundärminne i förväg skall hämtas in till primärminnet kan vara utformad med hjälp av statistiska metoder. Vi kan då tala om prognosstyrd blockutbytesteknik. Skilda statistiska principer kan komma i fråga. Som exempel kan nämnas att laddning av särskilt frekventa programblock företrädesvis kan utväljas, som komplement till aktuellt erforderliga programblock. Beroende på programstruktur kan även motsatt förfarande tänkas: block som redan exekverats kan synas löpa relativt liten risk att åter komma i fråga. Andra metoder kan användas. T ex kan "god" blockindelning utföras vid kompilering av programmen.

Exempel på system med maskinvarumässig adresseringsfacilitet som underlättar blockutbytesteknik är bl a operativsystemet Master för datorn CD 3300, samt datorsystemet IBM 360/67.

Debitering i multiprogrammeringsmiljö

Då flera program simultant befinner sig under behandling i primärminnet, mellan vilka kontrollen växlar enligt på förhand ej känt förhållande, är debitering med x kronor per verkligt förfluten timme orealistisk. Detta skulle innebära att en kund riskerat få betala för sådan från gång till gång varierande tid då hans program befunnit sig i vila på grund av att kontrollen legat hos annat program.

I stället måste de tidsintervall då hans program verkligen självt befunnit sig i exekvering ackumuleras. Denna information sparas i speciella register. Vid ett programs slut avläses i aktuellt register under hur lång tid programmet exekverats, och denna tidsangivelse matas ut och delges användaren. Han noterar därvid sin använda processortid, vilken i multiprogrammeringsmiljö alltid är avsevärt kortare än den verkligt förflutna tiden. Andra program hur ju bearbetats simultant, och dessutom har viss icke försumbar tid åtgått för operativsystemets styrande verksamhet. Denna sistnämnda tid betraktar vi som administrationstid (overhead time). Administrationstiden innefattar, som framgått av tidigare avsnitt, bl a brytsignalshantering, in/utmatningsadministration, programinläsning/byte, tidtagning osv. Detta innebär m a o arbete som an-

vändarprogrammen i mindre automatiserade system själva skulle få utföra. Det är sålunda icke helt rättvisande att betrakta den totala administrationstiden som spilltid eller icke produktiv tid. Icke desto mindre bör det vara ett av huvudintressena för operativsystemskonstruktörer att söka hålla denna administrationstid inom rimliga gränser. Stora skillnader har emellertid noterats mellan existerande system i detta avseende.

Användaren belastas sålunda med en taxa för använd processortid. Vissa system debiterar emellertid inte endast för denna systemresurs. Dessa debiteringssystem kompletterar processortidsdebitering med debitering för primärminnesresidens, mera avancerade system debiterar dessutom för nära nog varje annan utnyttjad systemresurs.

Residenstid i primärminnet som separat debiteringsfaktor speglar den alltmer ökande vikt som primärminnesutnyttjandet tillmäts. Debitering sker dels med hänsyn till verkligt förfluten tid i minnet och dels beroende av hur stort minnesutrymme som belagts. För ett system som tilldelar primärminne i block av viss fix längd anges debiteringsgrund normalt som produkten av antalet belagda minnesblock och den förflutna tiden. I den mån antalet belagda minnesblock varierat under programexekveringen ges då debiteringsgrund som en summa av deltermer härrörande från beläggningen av de olika minnesregionerna var för sig. Med en speciell taxa för minnesresidens kan dessutom möjligen den synpunkten anläggas att det avses kompenseras mot användare som genom rak kodning (mera minneskrävande) söker hålla erforderlig processortid nere.

Vissa system medger standardmässig möjlighet att för en driftsledning söka erhålla så internt jämn systembeläggning som möjligt genom anpassning av de speciella debiteringstaxorna även för yttre enheter. För sekundärminnen av direktminnestyp debiteras enligt samma princip som för primärminnet, kompletterat med en taxa för tidsmässigt belagd höghastighetskanal. För sekvensminnen debiteras för "låsta" bandstationer, varvid residenstid används, kompletterat med kanalkostnad.

För konventionell in/utmatning lämnas av nära nog samtliga system standardmässiga uppgifter om antal lästa hålkort, antal lästa hålremstecken, antal på radskrivare utmatade rader och sidor osv. Dessa uppgifter kompletteras oftast med hyreskostnader av mera fast typ, för egna band, skivpackar, speciellt radskrivarpapper osv. Det framgår att debitering på nyare multiprogrammerade datorsystem i rättvisans tecken är uppdelad på en mängd skilda poster. Systemet lämnar standardmässigt vid varje jobbs slut uppgifter på samtliga belagda resurser, till grund för driftsledningens taxemultiplikationer och summeringar. Dessa sistnämnda beräkningar utförs vanligen via månatligen körda debiteringsprogram.

Ett principiellt problem vid debitering i multiprogrammeringsmiljö rör primärminnesresidensen. Beroende på var eller under vilka köromständigheter ett program exekverats i minnet kan användaren notera icke överensstämmande kostnader vid körning av samma program vid skilda tillfällen i ett system. Detta beror på att väntetiden för programmet medan andra program exekveras bredvid i minnet kan variera. Om samtliga bredvidliggande program är starkt in/utmatningsbundna, med ideliga in/utavbrott och överlämnande av kontroll till det förstnämnda programmet som följd, kan väntetiderna för detta program väntas bli korta, med kort residenstid som slutgiltig följd. Detta blir sålunda billigt för den förstnämnde användaren. Om däremot bredvidliggande program är processbundna och har högre prioritet, tvingas det förstnämnda programmet kanske belägga minnet längre i väntestatus, med högre kostnad som följd. Emedan karaktären för bredvidliggande program ej är känd för användarna, blir de principiellt utsatta för varierande körkostnader. I vettigt utformade system är dessa variationer måttliga, medan de i andra fall kan bli avsevärda, ibland upp till en faktor två. Någon metod som generellt kan klara detta principiella problem är för närvarande inte känd för författarna.

Administrationstid och systemresursbehov

Om vi tänker tillbaka på 50-talets operativsystemsfunktioner, jämfört med de enligt avsnitten ovan beskrivna från 60-talets senare del, kan vi huvudsakligen notera:

1) Dagens system är väsentligt mer komplexa till konstruktion och arbetsprinciper.

2) De nuvarande systemen kan sannolikt anses ta tillvara datorsystemens resurser på effektivare sätt.

3) Större insatser krävs av användarna (via styrspråk etc) vid konfrontation med de existerande systemen än tidigare.

Det bättre datorutnyttjandet med dagens system har emellertid inte uppnåtts gratis. De styrande uppgifter som operativsystemen övertagit från operatörer och programmerare utförs numera till viss del inom datorsystemen och detta tar maskinresurser i anspråk. Dessa är ofta ingalunda av försumbar storleksordning. Begreppet over-head-time, ovan försvenskat till administrationstid, är i dagligt tal inte ofta definitionsmässigt klarlagt. Rosin anger (schablonartat) i Computing Surveys, vol 1 nr 1 1969, följande:

"the amount of time required by the supervisor to maintain the job flow".

En något striktare definition är önskvärd. Tänkbart vore att från total processortidsåtgång för ett givet antal jobb subtrahera jobbens summerade processortider, för att erhålla systemets administrationstid. Detta blir emellertid inte rättvisande, då administrationstiden är intimt knuten till löpande jobexekvering. Förutom tidsåtgång för laddning av jobb, växling mellan jobb och avslutning av jobb måste vi ta hänsyn till tidsåtgång i samband med administration av avbrott under exekvering. Fördelarna med t ex den decentraliserade och överlappade in/utmatningen kompenseras alltså av visst "tidsspill" för hantering av de avbrottssignaler som styr densamma. Speciellt märkbara blir administrationstider i multiprogrammeringsmiljö, med ideliga programkontrollväxlingar. Dessutom varierar givetvis spilltiderna avsevärt beroende på jobbflödets utseende, programtyper osv. Genomsnittsangivelser för olika jobbmiljöer blir därför nödvändiga.

Som synes kan knappast generella spilltidsangivelser ges för betraktade operativsystem. Anvisade värden, mellan 5 % och 50 % av totaltiden, hänsyftar vanligen till speciella installationer, och låter sig knappast exakt jämföras. Vi får därför dessvärre nöja oss med att konstatera att administrationstider

- är icke försumbara till storleksordning
- kan meningsfullt jämföras endast för väl definierade typiska jobbmängder och jämförbara system.

Klarare begrepp kan fås angående operativsystemens övriga systemresursbehov. Här kommer minnesbehov i blickfånget. Dagens operativsystem kräver ofta en kraftig del av en datorinstallations primärminne såväl som sekundärminnen. Icke sällan har primärminnesbehov på flera hundra kilobytes noterats, kompletterat med omfattande direktminnesutrymme (ofta 5-10 Megabytes). Dylik storleksordning på behov är naturligtvis mindre populär hos såväl köpare som användare. Program som utan vidare kunde köras på tidigare datorer, ryms ofta inte i dagens primärminnen, trots att dessa ofta är väsentligt större än vad som tidigare var brukligt. Segmentering av existerande användarprogram blir därför ofta en nödvändighet, med ty åtföljande ökad exekveringstid för programmen (användarstyrd "swapping" erforderlig). Vid konfigurering av ett aktuellt datorsystem idag är det sålunda av mindre intresse att studera totalt primärminnesutrymme, vikt bör i stället huvudsakligen fästas vid storleken av tillgängligt primärminne efter det att operativsystemet "tagit sitt". Resonemanget gäller även sekundärminnesutrymme.

Vi kan med fog fråga oss varför utvecklingen gått hithän, med alltför dominerande systemresurskrav för operativsystemen själva. Utvecklingsmässigt skulle vi kunna formulera trenden att "systemen (programvara + maskinvara) Parkinsonskt sett alltmer äter upp sig själva". Tre tänkbara skäl för denna utveckling kan nämnas:

1) Alltför liten vikt har av leverantörerna lagts vid totalsystemsbetraktelser och målanalys. Man har koncentrerat sig på optimering av delsystemens uppförande och underlåtit att tillräckligt studera delsystemens sammanlagda effekt.

2) Operativsystemen har oftast framställts under starkt decentraliserade arbetsformer. Olika personer har konstruerat de olika delsystem helt separat, med konkreta svårigheter till samordning vid arbetets slut som följd (jfr Parkinsons lag).

3) Systemen har konstruerats med tanke på att kunna klara nära nog alla typer av tillämpningar i stället för viss specialisering. En tänkbar jämförelse pekar mot det ineffektiva härmed: biologiska system är ofta huvudsakligen utformade med tanke på speciella funktionsförhållanden, och synes klara sig bra härmed.

Vi återkommer med ytterligare synpunkter på dagens systemsituation (i förhållande till morgondagens förväntade) i detta kapitels sista avsnitt.

Time-sharing- och reelltidssystem

Utvecklingsmässigt kan, i anknytning till vad som berörts i kapitel 1, time-sharing-system och multipel-access system sägas ha existerat i viss omfattning sedan mitten av 60-talet. En markant strävan mot reelltidssystem har kunnat noteras under de senaste åren även för applikationer av administrativ karaktär och ej enbart för militära-strategiska system.

Båda dessa typer av system, vilka har funktionella likheter i flera avseenden, baserar sig på direktbearbetningsprinciper som berörts i detta kapitels tidigare avsnitt. En verifiering av tesen att ett time-sharing system inte nödvändigtvis behöver eller ens bör vara komplicerat uppbyggt (en tes som numera framstår allt klarare) är principerna och funktionsdugligheten för det tidiga (1963) BBN-systemet från Cambridge, USA (BBN = Bolt, Beranek, Newman). Detta system realiserades på en PDP-1-dator med 8 K ords primärminne, varav 4 K var permanent

upptagna av operativsystemets residenta del. Som back-up minne fungerade en trumma med användarutrymmet 22 "sektorer" om 4 K ord vardera. Några speciella maskinvarumodifikationer hade gjorts till datorn:

- möjlighet fanns att "dumpa" primärminnets användarutrymme (4 K) till en trumsektor samtidigt som en annan sektor lästes in till primärminnet.

- master/slave-mode operationer hade införts

- en klocka hade inmonterats, som kunde generera avbrottssignaler var 20:e millisekund.

Med BBN-systemet kunde samtidigt 5 anslutna skrivmaskiner vara i arbete. Systemet arbetade med en okomplicerad teknik för programväxling mellan primär- och sekundärminne: endast ett program befann sig i primärminnets användarutrymme åt gången, de andra befann sig därvid i vila på trumminnet. Kvantumteknik tillämpades där ett och samma program tilläts exekveras högst 140 millisekunder i taget. När det i primärminnet aktiva programmet hade tillgodogjort sig sitt processortidskvantum, eller begärt in- eller utmatning, eller nått sitt programslut, genererades ett programavbrott, kontrollen gavs till operativsystemet och programmet i primärminnet ersattes av ett annat på trumman väntande program. Den programinplanerande algoritmen arbetade helt utan prioritetsnivåer, med en enda cirkulär kö på trumman. Responstiden för en användare kunde inte påverkas från skrivmaskinsterminalen.

BBN-systemet realiserade ett antal viktiga time-sharing-principer på ett enkelt och funktionsdugligt sätt, och kom sannolikt att utgöra en förebild för senare system.

Ett sätt att komma ifrån önskemålet om master/slave-teknik visades av det omtalade JOSS-systemet (nämnt tidigare). Här utfördes varje instruktion interpretativt, och operativsystemet befann sig "ständigt i master mode". En dylik princip är tillförlitlig, när operativsystemet blivit fullständigt inkört och befriat från programfel kan användarna inte åstadkomma någon som helst programstyrd systemskada. Effektivt maskinutnyttjande kan vid interpretering emellertid knappast uppnås med dagens maskinvaror, spilltiderna blir avsevärda. Man räknar med att 20-40 gånger mer processortid åtgår vid interpretering jämfört med exekvering av ett kompilerat program.

Det skulle föra för långt att i denna kortfattade utvecklingsmässiga översikt vidare belysa arbetsprinciperna för senare time-sharing system. Många har uppbyggts med hjälp av funktioner och principer, vilka utprö-

vats i normala multiprogrammeringssystem och av vilka de flesta berörts tidigare i detta kapitel. Vi återkommer i stället senare till funktionsprinciper för den betydelsefulla milstolpe som time-sharing visat sig innebära för access till datorsystem.

Ett reelltidssystems operativsystem omfattar givetvis funktioner av vilka många förekommer även vid satsvis arbetande system. Till skillnad från satsvis bearbetning och time-sharing är ett i drift varande reelltidssystem helt dedikerat för bestämda tillämpningsuppgifter. Endast de funktioner som av systemerarna infogats i applikationsprogrammen kan utföras. Det finns hos dagens system ingen möjlighet att från en reelltidsterminal överföra program till systemet (i varje fall ännu ej implementerat; vi skulle i så fall närma oss begreppet time-sharing i stället för reelltid). Bildligt sett kan ett reelltidssystem sägas vara "transaktionsstyrt", till skillnad från ett satsvis arbetande system och i viss mån även ett time-sharing-system, som kan sägas vara "programstyrt". Denna syn på reelltidssystem i förhållande till andra system baserar sig huvudsakligen på skillnader ur användningssynpunkt snarare än principskillnader för bearbetning av program. Studerar vi operativsystemens funktion i de olika fallen, finner vi sålunda väsentliga likheter. Det är hela tiden fråga om att ta emot transaktioner, hårt kodifierade eller ej, vilkas behandling kräver initiering av aktivering av en kedja av residenta eller transienta programdelar. De för reelltidssystem normalt starka önskemålen om snabba responstider och hög total systemtillförlitlighet medför att kraven på motsvarande operativsystemfunktioner är högt ställda. Hittills har dessa önskemål endast i begränsad omfattning infriats av existerande operativsystem.

Reelltidssystem kännetecknas även av viss maskinvarumässig utrustning som normalt inte förekommer i satsvis miljö. Vi tänker här på kommunikationsmultiplexorer, kommunikationsdatorer, bildskärmar eller annan direktansluten datagenererande eller mottagande terminalutrustning för samspel människa-maskin eller maskin-maskin. Denna orientering mot datakommunikation har introducerat viss tyngdpunktsförskjutning inom operativsystemen mot "message-switching" funktioner. Maskin och programfunktioner för att öka reelltidssystemens driftssäkerhet och tillgänglighet spelar här en betydelsefull roll.

Det kanske ändå mest kännetecknande för reelltidssystem är dels ordets magiska klang som utlovar lösning på alla företagsstyrningsproblem dels den oförbätterliga optimism varmed ADB-chefer gör upp tids- och kostnadsplaner för reelltidsbaserade "Management Information Systems".

Tiden 1969 -

En framtidsvy

Vi vill här inte göra anspråk på att förutsäga utvecklingens kommande förlopp. Dagens operativsystem synes dock besitta vissa brister, av vilka vi här vill påpeka några och i tillämpliga fall antyda vad man bör kunna vänta av 70-talets system.

Inledningsvis kan det naturligtvis ifrågasättas om dagens "general-purpose" system har kommit för att stanna. Det har, möjligen försäljningspolitiskt motiverat, ansetts att ett operativsystem bör kunna agera effektivt i nära nog samtliga tänkbara problemmiljöer, det må röra satsvisa kommersiella tillämpningar, satsvisa tekniskt orienterade tillämpningar, forskningsbetonad problemmiljö, time-sharing-miljö, reelltidsmiljö osv.

Även om det ännu knappast lyckats att realisera system som ens funktionsmässigt klarar allt detta, så har önskvärdheten såväl från användarnas som leverantörernas sida visat sig tydlig. Röster hörs som menar att utvecklingen redan syns ha anvisat väsentliga nackdelar med ett dylikt generellt tänkande. Alltför höga krav på människorna vid kommunikation med systemen (rigorösa styrspråk utan tillräckliga förenklingsmöjligheter, omfattande tekniska krav vid program- och datainmatning, mångfaldig redundans vid resultatutmatning, för att nämna några), tunga resurskrav för operativsystemen själva: "minnesätande" system, avsevärda administrationstider osv; ett nytänkande anses vara motiverat.

Två citat som understryker detta förhållande må ges. Saul Rosen säger i "Programming Systems and Languages": "Design of a system is a compromise between many design criteria. One of the most important is keeping the system overhead cost low on the many problems that do not require the use of very sophisticated system features".

Liknande tongångar spelas upp av Robert F. Rosin i "Supervisory and Monitor Systems", ACM Computer Surveys, Vol 1 nr 1, mars 1969: "The designers of supervisory systems must implement schemes which result in as little overhead as possible for jobs which don't require or use the full resources of a system, but which make extended functions available to those users for whom they are worth the added cost."

Den maskinvarumässiga decentralisering som gör sig märkbar för närvarande, med det ökande intresset för små datorer, understöds alltså

av operativsystemsutvecklingen härvidlag. Användare med en mångfaldigt skiftande användningsmiljö (skilda typer av jobbprofiler) kan komma att koncentrera sig på ett flertal skilda system, maskinvarumässigt som programvarumässigt, snarare än ett enda universellt kombinationssystem.

Skälen för denna utveckling är ekonomiska, bästa systemekonomi ej inkluderande de mänskliga kostnaderna har hittills varit ett dominerande kriterium vid systemplanering.

Vad som kanske förvånar läsaren är att både Rosen och Rosin (ovan) lägger tonvikten vid ett effektivt utnyttjande av maskinsystemet utan att samtidigt tala om människans plats i sammanhanget. Det kan ej förnekas att kostnader för systemerare och programmerare i dag överstiger maskinvarukostnader. En än större skillnad är att förvänta i framtiden. Det kan därför förefalla endast måttligt meningsfyllt att ensidigt jaga efter maskineffektivitetsökning.

En rimlig utveckling rörande datorsystem kan således vara mot:

- lättåtkomliga system (vid olika slag av terminaler)

- instruktiva, "frågvisa" system, som hjälper till med bortglömda konventioner.

- lättmodifierbara system, det bör vara möjligt att på ett enkelt sätt ändra systemets arbetssätt eller utöka/minska dess funktionsrepertoir.

De flesta maskinorienterade systemerare och programmerare är idag i hög grad medvetna om operativsystemens existens i så måtto att dessa system styr och ofta begränsar handlingsfriheten. Det förefaller rimligt att i framtiden önska sig system där användarproblemet - föremålet för bearbetning - står i centrum och ej operativsystemet. Operativsystemet bör i användarens ögon vara ett servicesystem snarare än ett styrande system.

En annan debattpunkt för närvarande rör allmänna systemkonstruktionsprinciper: skall ett operativsystem tillhandahållas i form av en enda "svart box" eller i form av en samling på olika sätt kombinerbara systemmoduler? Den svarta boxen kännetecknas av att den ur användarnas synpunkt ej låter sig modifieras; modulsystemprincipen medger anpassbarhet till eventuellt växlande användarmiljö. Det vill vid första åsyn synas uppenbart med dominans för de fördelar modulprincipen innebär. Ett närmare studium leder dock till ett något annorlunda slutintryck. Erfarenheter från installation och, framför allt, underhåll av operativsystem

pekar på den höga systemprogrammerarkompetens som är erforderlig för att modulsystemen skall fungera tillfredsställande. Tillräcklig kompetens kommer att finnas hos leverantörerna eller hos programvarutillverkarna. Kommer användarna att ha tillgång till lika god specialistkompetens? Även med hänsyn tagen till en expanderande utbildning i informationsbehandling, särskilt rörande systemprogrammering, är det rimligt att resa tvivel härom. Antalet datorinstallationer kommer sannolikt att expandera kraftigare än tillgången på kvalificerade systemprogrammerare. Man kan därför tänka sig fördelar med koncentration på mera box-liknande system, vilka levereras färdiginkörda (så nära som sig göra låter) från tillverkaren, och av vilka slutna uppdateringar sedan kontinuerligt tillhandahålls från samma källa. För att möjliggöra viss flexibilitet inom användarmiljön är det tänkbart att leverantörerna till samma maskinvarusystem, alternativt de fristående programvarutillverkarna, kommer att tillhandahålla ett mindre antal olika box-system, ett för satsvis bearbetning, ett för time-sharing, ett för vissa reelltidstillämpningar osv. Uppdatering av dessa boxar kommer likafullt leverantören av programvaran att få stå för.

Först när utvecklingen av operativsystemsprinciper stabiliserat sig i förhållande till dagens situation, är det rimligt att vänta att dessa boxar kommer att utvecklas från programvarukaraktär till maskinvarukaraktär (eller till ett samspel av båda dessa). Någon dylik stabilisering är dock ej i sikte - utvecklingen har tvärtom accelererat.

Den inte ringa konstruktionsmässiga komplikationsgraden hos dagens operativsystem kommer sannolikt att ytterligare ökas för systemen under 70-talets första hälft. Förvånansvärt sent i programvaruutvecklingen har allvarligt intresse börjat knytas till effektivitetsstudier rörande systemens samspel med människan och utnyttjande av maskinvarorna. Först de senare åren på 60-talet har i vissa system intern statistik över maskinvaruutnyttjande börjat insamlas: kanalbeläggningsgrader, minnesutnyttjande, accesskonflikter, spilltider, balans mellan behandlade jobbtyper osv. Några system insamlar kontinuerligt och standardmässigt dylik statistik med hjälp av en i operativsystemets centrum sig befinnande registreringskärnrutin (som t ex matar ut dessa beläggningsdata på ett reserverat systemband). Studier som syftar till att kartlägga systemerarnas-programmerarnas effektivitet i olika OS-miljöer har endast gjorts vid ett fåtal installationer.

För att söka kompensera för, eller åtminstone minimera, de administrationstider som mer eller mindre permanent residenta registreringsrutiner medför under själva beläggningsstudierna, innebärande skeva mätresultat, har på sista tiden intresse börjat knytas till inkoppling av mekaniska

maskinvarumässiga beläggningsmätare i systemen. Specialkonstruerade räkneanordningar av dylik typ bör på effektivare sätt än inbyggda programrutiner kunna ge besked om systemutnyttjande.

Först när resultat från dylika systemmätningar hunnit iterera sig fram till verklig och effektiv påverkan på själva operativsystemens konstruktion, kan tiden anses vara mogen för upptagande av diskussioner om möjlig standardisering av principer för vissa givna jobbearbetningstyper. För olika leverantörer gemensamma arbetsprinciper för specialiserade operativsystem av "svart box"-typ kommer först därefter att bli aktuella.

Trender som kommer att ha avgörande inverkan på operativsystemens utveckling är

(1) trenden mot databankar samt kommunikation mellan databankssystem

(2) trenden mot "förenklat umgänge" mellan människa - dator - kanske kommunikation mellan dessa i ett språk nära "det naturliga"

(3) trenden mot att tillhandahålla för alla samhällets medborgare informationsbehandlingsservice - ungefär som telefon, TV eller elektricitet.

När det gäller databankar existerar det redan idag system med egenskaper som visserligen är primitiva men för området fundamentala. Dessa system accepterar vanligen strukturerade data och kan företa olika former av utsökning. En del system medger även att helt separera program- och filbeskrivningen. Även om många problem återstår att lösa avseende dessa "generella filhanteringssystem" bör de redan nu ses som en utökning av operativ-systemen. Deras betydelse väntas bli störst vid applikationssystemutformningen och kommer sannolikt att radikalt ändra principerna för systemarbete. Av tids- och utrymmesskäl har vi dock valt att ej närmare beröra detta komplex i denna bok. Dessutom finns det redan viss litteratur (även svensk) på området.

Trenden (2), förenklat människa-datorumgänge syftar dels på att på ett mer naturligt och flexibelt sätt än hittills kunna tala om för datorsystemet vad man vill ha utfört, och när, och dels på mer naturliga utmatnings--presentationsprinciper såsom visuellt på bildskärm, och vokalt. Det är t ex redan nu klart att "programmet" sådant som vi känner till det idag, innebär ett besvärligt, fel-vänligt och för människan generellt onaturligt sätt att definiera problemställningar och bearbetningar. Än så länge tvingas vi dock, mest beroende på bristande datorresurser men även på

grund av det invanda systemets tröghet, att arbeta med det konventionella begreppet "program".

(1) och (2) syftar således till att utöka datorerna med funktioner (mest programvarumässigt) som gör det lättare för människan att beskriva datastrukturer, söka och manipulera data. Detta bör innebära att den av användaren "programmerade" delen av ett applikationssystem minskar i förhållande till den totala programmängden. Det förenklade "umgänget" kommer också medföra effekten att "systemarbete" sådant som vi känner till det i dag kommer att ändra karaktär i och med att många av systemerarnas uppgifter kommer att övertas av datorn. Att vårt sätt att definiera problem och bearbetningar för ett datorsystem och "samspela" med detta måste radikalt ändras pekar även trenden (3) på.

Vad kan man då säga om framtida operativsystem? Sannolikt kommer dessas arbetsuppgifter och organisation att expandera.

Mindre och mindre arbete kommer att kvarstå för systemerarna åtminstone vad utrustningsanpassad konstruktion av systemet anbelangar. De kanske kommer att få möjlighet att helt ägna sig åt det "rena" problemorienterade systemarbetet - utan att behöva tänka på knepiga konventioner av datororienterad natur. Det bästa operativsystemet är ju i alla fall det som inte märks.

LITTERATUR

1. Rosen, S., Programming systems and languages, McGraw-Hill 1967

2. Rosin, R. F., Supervisory and Monitor Systems ACM Computing Surveys, vol 1 nr 1 mars 1969

3. Martin, J., Design of Real-Time Computer Systems, Prentice-Hall 1967

4. Wilkes, M. V., Time-sharing computer systems, MacDonald Computer Monographs 1968

5. Alt-Rubinoff, Advances in computers, Volume 3, Academic Press, New York/London 1962.

Till ovanstående publikationer tillkommer olika datortillverkares manualer avseende operativsystem.

Studentlitteraturs utgivning i informationsbehandling — ett urval

Titles marked E are published in English also
Die mit einem D bezeichneten Bücher sind auch in deutscher Auflage erschienen
S:llä merkipyt teokset ovat myös julkaistu suomenkielisinä

Allmän introduktion

	L A Bergström—M Jern	Inroduktion till grafisk informationsbehandling
	R Brandinger	ADB — Datorn
	K Cervin—K-E Hillander	Data för datorer
E	*O Dopping*	Datamaskiner och databehandling
S	*O Dopping*	Kort och brett om ADB
	T Ekman—C-E Fröberg	Data för skolan
	Ö Leringe	Programmering av datorer — en inledning
	S Persson	Elementär datamatik
	K Siro (red)	Vi lär oss ADB och BASIC
	H van Tongeren—G Lekberg	Databehandlingens och programmeringens grunder
	L Wettermark	ADB från början

Numerisk analys

B-E Bengtsson—T Ekman	Numerisk analys — allmän kurs
E Lundqvist	Introduktion till numeriska metoder
E Lundqvist—A Sjöberg	Numerisk analys
E Lundqvist—A Sjöberg	Numeriska metoder
A Lysegård	Formelsamling i numerisk analys
A Lysegård	Problemsamling i numerisk analys

Digital teknik och maskinvara

L Alm	Logisk systembyggnad. Lärobok och arbetsbok
A Andersson—S-E Wallin —A Westerdahl	Analogiteknik och analogimaskinen
L Bengtsson—O Eriksson —S-E Wallin	Logiska kretsar
P-E Danielsson m fl	Digital teknik
S *C Jennel—S-E Wallin*	Datorteknik och programmering
Y Sundblad	Boolesk algebra, grafer och ändliga automater
S-E Wallin	Datamaskinteknik

Datalogi med programmering och databehandlingsteknik

	D Belsnes—O-J Dahl	Algoritmer og datastrukturer
	B-E Bengtsson	ALGOL — övningsuppgifter
	G Birtwistle—O-J Dahl—B Myhrhaug—K Nygaard	SIMULA BEGIN
	O Björner	Avancerad COBOL
S	*O Björner—K Holm*	Grunderna i COBOL
	J Bubenko jr	Databehandlingsteknik
	J Bubenko jr—O Dopping	Dataehandlingens xyz
	J Bubenko jr—T Ohlin	Introduktion till operativsystem
	O-J Dahl	Syntaks og semantikk i programmeringsspråk
	T Ekman—G Eriksson	Programmering i Standard FORTRAN
E	*T Ekman—C-E Fröberg*	Lärobok i ALGOL
	L Ewald—E Roupé—B Wahlqvist	Lärobok i BASIC
	N Fredholm	Databasmetodik
	S Frenkel—S Persson	Elementär FORTRAN
	T Gilb	Reliable EDP Application Design
E	*E Gustafsson*	Handbok i optisk läsning
	E Gustafsson	HUPOL
E	*E Gustafsson*	Optisk handskrift — självstudiekurs
	H Haugland	Programutvikling
	H J Helms (red)	Vejledning i konstruktion af datamatiske systemer
	P Hoving—P Övernäs	Programmeringsövningar i COBOL
S	*S Kallin*	Lärobok i FORTRAN
	S Kallin—C Odén	Lärobok i PL/1
	R Kurki-Suonio	Computability and Formal Languages
	A Lysegård	Assembler för UNIVAC 1108
E	*A Lysegård*	Lärobok i COBOL
	P Naur	Concise Survey of Computer Methods
	J Palme	FORTRAN för den som kan ALGOL
	J Palme	Programmeringsspråk
	Regnecentralen, København	Datamatik — serie i 11 hæfter
	E Roupé	Programmeringsövningar i BASIC
	L Stenström	Programmeringsövningar i teknisk databehandling
	Y Sundblad	Algoritmteori
	L-E Thorelli	BIM och BAS — en orientering om maskinnära programmering
	L-E Thorelli	Listprogrammering
	P Östling (red)	Projektering av reeltidssystem — en introduktion

Administrativ utveckling med systemarbete

C Andersen—M Arentzen —A Petersen	Systembeskrivning
G Bark—R Brandinger	Arbete med informationssystem
R Brandinger—J Norrby	ADB — Systemarbete
J Bubenko jr— O Källhammar—B Langefors—M Lundeberg— A Sølvberg	Systemering 70
J Bubenko jr—B Langefors —A Sølvberg (Editors)	Computer-Aided Information Systems Analysis and Design
M Bäckström	Systemarbetets metodik
IAG	Management Information Systems
B Langefors	System för företagsstyrning
B Langefors	Theoretical Analysis of Information Systems
M Lindström— G Österberg-Svahn	Faktainsamling — informationsutbyte
M Lundeberg— E S Andersen	Systemering — informationsanalys
A Olerup	Introduktion till systemalgebra
S Persson	Beslutstabeller
L Wettermark	Systemarbete pågår

Övrigt, speciellt tillämpningar

H E Andersin—R Sulonen	Simuleringsteknik
K-G Björk—J Nordberg— T Johansson	Mikrofilmsystem — en grundläggande vägledning
P G Cassel	Statistisk databehandling
O Dopping	Sätt rätt lätt
Edb-rådet, København	Training Course for Programmers
Edb-rådet, København	Training Course for Datamaticians
Edb-rådet, København	Training Course for Operators
M Glader	ADB-lösningar på redovisningsproblem
O Karlqvis (red)	ADB-ordbok
N Lindecrantz	Dokumentation och datamaskiner
T Matre	Prosesstyring med datamaskin — en introduksjon
P Naur	Datamaskinerna och samhället
P Naur	Planer og idéer for datalogisk institut ved Københavns universitet
S Nordbäck—B Rystedt	Computer Cartography
I Westhagen	Prosesstyring

Janis Bubenko jr – Tomas Ohlin
Introduktion till operativsystem

Ett operativsystem är ett system av datormekanismer och program som har till syfte att dels underlätta konstruktion och implementering av applikationsrutiner dels styra dessa rutiners datorbearbetning.

Del 1 beskriver den maskin- och programvarumässiga utvecklingen fram till dagens operativsystem.

Del 2 belyser de viktigare funktionerna hos operativsystem och diskuterar även dessas roll i systemeringssammanhang.

Del 2 och vissa avsnitt i del 1 förutsätter kunskaper om datorers uppbyggnadsprinciper och programmering.