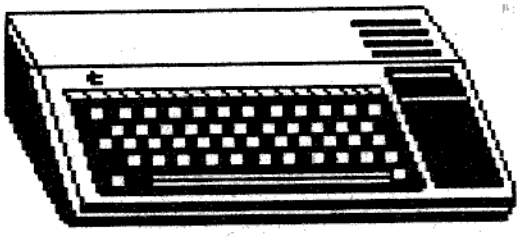
PROGRAM-BITEN 85-4

nittinian

SCREEN DUMP



MATRIKEL NUMMER

Innehåll

ISSN 0281-1146

Redaktören . . .

Det här numret är det andra jag ansvarar för som redaktör. En sak man funderar över är varför så pass få medlemmar hör av sig till tidningen. I det här numret ha vi en rättelse av ett program publicerat i nr 84-4. Av en händelse upptäcktes felet. Varför har ingen reagerat tidigare? Beror det på att de läsare som knappade in programmet upptäckte det snabbt och tyckte att felet var så självklart att alla andra också skulle hitta det. Eller beror det (hemska tanke) på att få besvärar sig med att knappa in programmen i tidningen? Ett annat program som kräver rättelse är *Sverige Runt* som publicerades i förra numret. Raderna 2440–2630 blev ordentligt tillrörda, detta beroende på att kontrolltecken omdefinierats i programmet. En rättelse kommer i nästa nummer.

I förra numret hade Anders Persson med en utmaning om att göra ett program för att beräkna checksummor för programrader i BASIC. Syftet var att publicera dessa i tidningen för att underlätta inknappande av programmen. Ett svar på detta har också inkommit. Vi kommer att presentera det närmare i nästa nummer, men det vore kul att höra ifrån dem som brukar knappa in program från tidningen om de tycker att det vore en hjälp.

Tävlingen *bra nyttoprogram* kan nu betraktas som avslutad. Så värst många bidrag har det inte blivit. Det är ännu för tidigt att kora en segrare. I nästa nummer ska vi nog kunna återkomma till det.

Det här numret upptas till en tredjdel av föreningens matrikel. Den är uppdaterad till mitten av september. Ta en titt i den och se efter vilka som bor i närheten. Har Du tagit kontakt med dem för att höra vad de gör med sina 99-or? Om inte så är det ett bra tillfälle nu.

Något material om TI-59 har vi inte med i det här numret. Det betyder inte att den grenen är helt avsågad, bara att det inte kommer in särskilt mycket material från det hållet. Till nästa nummer återkommer vi med TI-59-sidor igen. I det numret kommer vi också att publicera flera program i BASIC och XBASIC.

Här och var i tidningen finns små bilder utspridda. På första sidan finns också ett par rubriker gjorda i lite ovanliga stilar. Allt detta är framställt med hjälp av ett grafikprogram som heter *GRAPHX*. Det kommer från Australien. Enligt min mening är det ett fantastiskt program. På medlemsmötet i Stockholm i september väckte det också en hel del uppmärksamhet. Jag hoppas återkomma med en recension i ett senare nummer.

I förra redaktörsspalten hade jag ett upprop om någon som ville åta sig att hålla i en hårdvaruspalt i tidningen. Något direkt svar har inte kommit. Vi har dock en hel del godbitar på gång. Bl a har vi fått ritningar till ett bygge av ett 32K-minne som kan byggas in i konsolen.

Peter Odelryd

Peter Odelryd
Järnåldersringen 340
136 65 Handen
tel 08/777 12 16

Medlemsträffar

Staffanstorp. Lördagen den 22 november kl 9–17. Kastanjevägen 2. Upplysningar genom Sven Lundgren, tel 046/256421.

Stockholm. Lördagen den 29 november kl 14–18. Militärhögskolan, Valhallavägen 117. Mer information i ett separat utskick.

I redaktionen:

Redaktör	Peter Odelryd
Utmaningsredaktör	Anders Persson
Forth-redaktör	Lars-Erik Svahn
Programförmedlare	Börje Häll
Allt-i-allo	Claes Schibler

Föreningens och redaktionens adress:
Föreningen Programbiten
c/o Schibler
Wahlbergsgatan 6, 1 tr ned
12146 Johanneshov

Datainspektionens licensnummer: 82100488

Postgiro: 198300-6
Medlemsavgiften för 1986 är 120:–

Annonser, insatta av enskild medlem (ej företag), som gäller försäljning av moduler eller andra tillbehör i enstaka exemplar är gratis.

Övriga annonser kostar 2000 SEK per helsida, förutom baksidan som kostar 3000 SEK.

För kommersiellt bruk gäller följande:
Mångfaldigande av innehållet i denna skrift, helt eller delvis, är enligt lag om upphovsrätt av den 30 december 1960 förbjudet utan medgivande av Föreningen Programbiten. Förbudet gäller varje form av mångfaldigande genom tryckning, duplicering, stencilering, bandinspelning, diskettinspelning etc.

Föreningens tillbehörsförsäljning

Följande finns att köpa för medlemmar genom att motsvarande belopp insätts på postgiro 198300-6.

För TI-99/4A:	
Användartips med Mini Memory	60:–
PB-Forth kassett (föreningens Forth)	250:–
PB-Forth diskett (föreningens Forth)	250:–
PB- och TI-Forth+TI-Forth manual vid *samtidig* beställning (600 för icke medlem)!!	400:–
TI-Forth (diskett)	60:–
TI-Forth (manual)	190:–
PB-Forth källkod (3 disketter) [1]	120:–
TI-Forth källkod+demo (3 disketter) [2]	120:–
Game-Forth diskett	60:–
Forth&99 (3 disketter) [1]	120:–
Disassembler (diskett)	60:–
Multiplan/TI-Writer uppgraderingar diskett	60:–
Super Debugger diskett	60:–
Datavision modul	565:–
16 K RAM CMOS modul	875:–
Nittinian T-tröja	40:–
99'er magazine nr 12/82, 1–5, 7–9/83 (per styck)	20:–
Programbiten/Nittinian 1985 (TI 59+99)	100:–
Programbiten/Nittinian 1984 (TI 59+99)	80:–
Nittinian, årgång 1983 (TI-99/4A)	60:–
För TI-59 mm	
Programbiten, årgång 1983 (kalkylatorer)	40:–
Programbiten, årgång 1982 (kalkylatorer)	40:–
Programbiten, årgång 1981 (kalkylatorer)	40:–
Programbiten, årgång 1980 (kalkylatorer)	40:–
Programbiten, årgång 1978/79 (kalkylatorer)	40:–
Programbiten, fem årgångar 1978–83	160:–
Programbiten, sex årgångar 1978–84	220:–
Programbiten, sju årgångar 1978–85	300:–
Katalog med belgiska och engelska program för räknare TI-57, TI-58, TI-59	20:–
Föreningens programmeringsblanketter (TI-59), olika typer, block om 50 blanketter (se PB 83-1 sid 30) per block	11:–
Patenthandlingar TI-59	25:–
40 st tomma magnetkort med plånbok	SLUT
Tom magnetkortsplånbok	10:–

[1] Säljes endast till den som tidigare köpt PB-Forth
[2] Säljes endast till den som tidigare köpt TI-Forth

Ordföranden . . .

Det är nu dags att ta nya tag under hösten. Med hjälp av tidningen och tips från dig som medlem kan vi hålla liv i 99:an länge till. Skriv eller ring gärna till någon av oss i styrelsen och ställ frågor och kom med tips.

Jag själv skaffade min 99:a i början på 1983. Det var min första dator så jag har lärt mig allt från grunden på 99:an. Jag skaffade expansionboxen i oktober 1983 och kom sedan med i föreningen i början av 1984 efter det att TEXAS lagt ned tillverkningen.

TI-99/4A lever vidare med många nya tillbehör. Vi har beställt ytterligare *freeware* från utlandet. Många av dessa program är mycket avancerade och minst lika bra som många kommersiella program.

Nästan alla dessa program är skrivna i assembler för disk. Vi skall göra ett försök att översätta dessa till kassett för Mini Memory eller Extended Basic + 32 K RAM. Detta är en sak som jag själv är mycket intreserad av. Hör gärna av dig om du gjort någon egen översättning.

Jag har upplevt att TI-99/4A är en mycket bra dator att lära sig på. Det är få saker som kan haka upp sig och det är svårt att krascha ett program. Även spelprogram blir bra med mycket färger och många flera sprites än VIC 64 och Atari.

Andra saker som är speciellt bra på TI-99/4A är att den räknar med 14 decimaler och klarar av potenser upp till 127. Den lagrar även talen decimalt (RADIX 100) vilket gör att den alltid räknar rätt så att 0.1+0.1=0.2 vilket inte är sant på de flesta hemdatorer.

Tangentbordet kanske ser litet ut men tangenterna känns mycket skönare än de flesta andra datorer. ALPHA LOCK påverker endast bokstäverna och ej den övre sifferraden som t.ex. VIC-64 gör.

99:an är välförsedd med kommandon och det är nog mycket enklare att översätta från TI-99/4A till andra datorer än tvärtom. Om du har sett programlistningar för VIC 64 i *Compute!* och jämfört med motsvarande för TI-99/4A så känns det bra att slippa VIC 64 med dessa hjärter och spader och massor med konstiga tecken som ej har någon direkt motsvarighet på tangenterna. VIC 64 har heller inga mellanrum mellan kommandon och variabler vilket gör att IF THEN och FOR NEXT flyter ihop. 99:an är mycket bättre på alla desa punkter och dessutom slipper man PEEK och POKE i normala program. 99:an har även långa variabelnamn och RESEQUENCE som standard.

Lagringen av programfiler på kassett går bra men lagringen av datafiler går mycket långsamt. Lösningen på detta är antingen Personal Record Keeping eller Expansionsminne 32K.

PRK har lågt pris och har alla viktiga kommandon för att skriva och läsa text var som helst på skärmen motsvarande DISPLAY och ACCEPT i Extended Basic. Dessutom finns en snabb lagring av data på kassett liknande normala programfiler.

Expansionsminne 32K finns även att köpa fristående så att du inte behöver köpa expansionsboxen. I Tyskland finns mycket billiga minnen kombinerade med Centronics parallellgränssnitt för skrivare. RAM 32K+Centronics kostar DEM 290 och RAM 128K+Centronics kostar 400 DEM. Även den senare är helt fristående och verkar mycket prisvärd även om jag inte sett den. Skriv gärna till tidningen och berätta om vilka fristående minnen du har och hur de fungerar.

Om du känner för att ställa upp göra något inom föreningen så hör gärna av dig och berätta vad du skulle vilja göra. Detta gäller speciellt om du tycker att något område är försummat som du skulle kunna sköta bättre. Vi inom styrelsen försöker hålla ett så brett område som möjligt men även vi i första hand vill jobba med det vi känner mest för.

Även du som inte själv vet vad du kan göra inom föreningen men ändå tycker att du har tid bör höra av dig. Vi skall nog kunna hitta på något tillsammans.

I PB 86-2 hade jag ett förslag en träff om grund-BASIC. Det var endast en medlem som hörde av sig och ville komma. Från dig som inte bodde i närheten av Stockholm blev det större intresse för kassett med BASIC-program. Jag har skickat ut 15 sådana kassetter.

Jag har skaffat många billiga program från USA t ex LOGO II för USD 20 som även fungerar med kassett och 32K RAM. Hela programmet finns i modulen så du behöver inte ladda in några hjälpfiler från kassett utan endast spara dina färdiga program som i BASIC.

Samma dag som detta skrivs kom ett brev från *Millers Graphics* med DISkASSEMBLER som jag fick 24 dagar efter det jag skickat brevet med beställning och bankcheck (moneyorder). Det är snabba leveranser från MG, något som även gäller *MICRO-pendium*.

Mycket sämre är det med *Home Computer Magazine* som tycks ha slutat komma ut. Av en notis i den engelska *TI*MES* framgår att man kommer med ett stort *slutnummer* med ett värde av USD 25 så att man ordnar skulden till prenumeranterna. Jag betalade prenumerationen i början av mars och har ännu inte fått några nya nummer utan endast fyra gamla nummer som beställdes samtidigt.

Jan Alexandersson

Jan Alexandersson
Springarvägen 5, 3 tr
142 00 TRÅNGSUND
Tel. 08-771 05 69

Från programbankiren

Det har skett en del ändringar i programbanken, from den lista som infördes i Programbiten 86-3. Några nya kategorier har tillkommit. Det är 87 assemblerrutiner, 88 assemblerprogram och 89 subrutiner, subprogram. Alla dessa fanns förut under 90 praktiska programhjälpmedel vilket jag tycker är missvisande.

Kategorierna 87 och 89 innehåller små programsnuttar som inte gör något "vettigt" om de används för sig själva, de bör alltså användas i andra program. En del av rutinerna går inte att köra fristående utan måste ingå i ett program. Observera att under kategori 88 kommer 15883001 Disassembler att utgå och ersättas av en diskett med källkoder och objektkod.

I fortsättningen kommer vi inte att publicera den kompletta listan över program i Programbanken i varje nummer.

Följande tillägg, rättningar och borttagningar har gjorts sedan föregående fullständiga lista över program i programbanken.

	14	Ekonomi
17143015P	—K	Kontoprogram. Upp till 22 konton.
	89	Subrutiner, suprogram
17893019	—-	Shellsort. 99'an 83-4/5
17893020	—X	Quicksort. 99'an 83-4/5
18893003	—-	Riktiga små bokstäver. 99'an 83-4/5
18893005	—-	"DISPLAY AT" i TI-Basic. 99'an 83-1
18893006	—-	Högerjusterade tal. 99'an 83-3
	91	Spel
17913016	—-	Earth Attack. 99'an 83-1
17913017	—J	Dominion. 99'an 83-2
17913018	-JX	Jagad. 99'an 83-4/4
18913001	—J	Pucman. 99'an 83-3
18913002	-JX	Stockholm Competition. 99'an 83-3
18913004E	—-	One-Check. 99'an 83-2
18913011	-JX	SHARK. Ta dig till bryggan utan att ätas av hajarna
	92	Utbildning
18923009	-JX	Svenska städer. Placera städerna rätt på kartan. PB 86-3
18923010	—-	W-ORD-S1. Övning av engelska glosor. Endast på kassett
18923012	—-	W-ORD-S2. Övning av engelska glosor. Något svårare än W-ORD-S1. Endast på kassett
	97	Demo
17973013	-DX	SORTDEMO. Demo av Quick-, Fast-, och Bubblesort. PB 86-3
17973014	—X	SORTDEMO. Demo av Quick-, Fast-, och Bubblesort. Kassettversion. PB 86-3
18973007	—-	Att använda grafiken. 99'an 83-2
18973008	—-	Sammansatta ledtexter med INPUT.99'an 83-4/5
	99	Övrigt
18993013	—X	QUETELETINDEX. Visar om du har under-, normal- eller övervikt.

Advanced diagnostics

av Anders Persson

Millers Graphics har en del intressanta program att erbjuda, däribland Advanced Diagnostics. Det här programmet har i någon mån karaktären av universalverktyg. Det går nämligen att använda både till att testa en del av maskinvaran och till att härja på disketter.

I motsats till mycket av den programvara som finns till 99/4A är den här inte menydriven. Det kommer alltså inte hela tiden fram listor med alla alternativ som finns tillgängliga. I stället får man skriva in det kommando man vill ha direkt. Anledningen till detta är naturligtvis att det finns ganska många olika saker som man kan göra, och alla är ungefär lika tänkbara i varje ögonblick. Ett menysystem, där man först ska tala om ungefär vad det är man vill göra, sedan exakt vad det är och till sist vilka parametrar (filnamn, sektornummmer etc.) som behövs, ja, det blir helt enkelt för tungrott. Personer med vana vid Disk Manager modulen förstår nog vad jag menar.

Varför arbetar inte alla system med kommandon då? Jo, därför att menyer är lättare att lära sig. Allt som kan göras står ju på skärmen. Med ett kommandodrivet program räcker det inte att användaren vet vad han vill, han måste veta hur man kommenderar det också. Att det kan fungera alls överhuvudtaget beror på att man ganska snabbt lär sig de vanligaste instruktionerna. För att det ska fungera även med mindre ofta använda program krävs att programmet kan tala om vilka kommandon det förstår, om man så önskar. Det kan Advanced Diagnostics.

Test av maskinen

Advanced Diagnostics kan kolla om minnet fungerar på datorn. Detta gäller både konsolens RAM och minnesexpansionen. Dessutom kollas Mini Memory, om den är ansluten.

Det går också att se om diskdriven snurrar med rätt varvtal, hur snabb stegtid mellan spåren den klarar och om den kan lokalisera alla spåren på disketten.

Disketter

Förutom en katalog med filnamnen kan man få reda på om någon sektor på disketten befanns vara dålig vid formateringen och om någon fil är uppdelad i flera smådelar på disketten. Om man vill editera inuti en fil kan man också få reda på vilka sektorer som en fil upptar.

För ändring av filer finns en fullskärmseditor att tillgå. Efter ändringarna kan man skriva tillbaks sektorn var som helst på disketten. Det går även att läsa ett helt spår, inklusive ID, CRC osv.

Disketter kan formateras också. Advanced Diagnostics kan initiera en hel diskett på vanligt sätt, eller också bara formatera enskilda spår.

För kopiering finns kommando som buffrar upp till 36 sektorer innan man måste skriva ut informationen. Även här kan man kopiera till något annat ställe på samma diskett, om man så vill.

Kommandofiler

För att underlätta utförandet av vissa uppgifter kan Advanced Diagnostics utföra sina funktioner även om orderna kommer från en fil på en diskett. En sådan fil kan man skriva exempelvis med TI-Writer, eller något annat program som ger en DIS/VAR 80 fil. Alla kommandon kan programmeras på det här sättet. Det går alltså utmärkt att skriva en kommandofil som kopierar en hel diskett helt automatiskt. En del sådana här filer följer med Advanced Diagnostics, bland annat en demonstration av Advanced Diagnostics möjligheter.

Diverse

Några kommandon används huvudsakligen i kommandofiler. Beep och pause är sådana.

Det finns också möjlighet att välja färgerna på skärmen, omvandla tal mellan hexadecimalt och decimalt samt dumpa skärmen på skrivare. Filnamn och utskriftsbredd kan ställas in valfritt.

I slutet av den här artikeln finns ett exempel på hur det ser ut när Advanced Diagnostics just har konstaterat vilka filer som finns på en diskett.

Bruksanvisningen är mycket detaljerad, och innehåller även lite information om kontrollkretsar för disketter.

Genom att trycka på AID kan man också få snabbhjälp "on-line", som teknikerna säger när de ska låta avancerade. Detta är alltså den kommandolista jag nämnde tidigare.

Summering

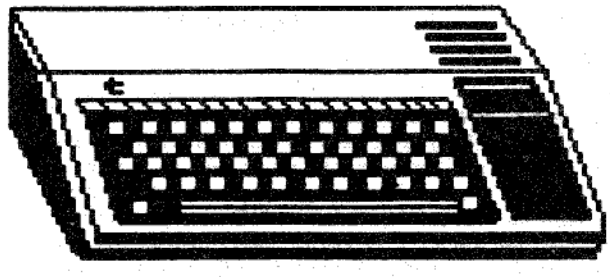
För att köra Advanced Diagnostics krävs 32K RAM, diskettenhet och någon av X-BASIC, E/A eller Mini Memory. Programmet kan också laddas med CorComp's diskkontrollkort. Däremot är det inte kompatibelt med Myarc's sådana. Programmet kostar USD 19,95 och kan beställas från de flesta firmorna i USA som handlar med program till 99/4A.

En nackdel är att det är ordentligt kopieringsskyddat. Du kan alltså inte flytta det till någon av dina egna disketter för att ha det nära till hands. Vad värre är, det går inte att göra någon säkerhetskopia heller.

Orsaken till det här otyget är naturligtvis det utbredda piratkopierandet. Notabelt är att det är en tämligen dubbelmoralisk inställning Craig Miller håller sig med. Han har ju nyligen visat sin GRAM Cracker i USA. Det är en grunka som just är tänkt till att norpa andras program med...

Så här ser det ut när Advanced Diagnostics gjort en filkatalog. Utskriften är gjord med programmets egen skärmdump. (Utskriften är lätt redigerad. Vissa linjer osv på skärmen dumpas inte till skrivare.)

```
     ADVANCED DIAGNOSTICS
            Status
Command : Disk Directory
-------------------------------------
Drive    : 2     Track  : 0
Side     : 1     Sector : 10
Free     : 680   H/S    : 3
Used     : 758   Bad    : 0
            File List
  AREG2/0       AREG2/1
  AREG2INSTR    CASSCOPYL
  CASSCOPYO     CASSCOPYS
  DIAGS-REV     DISK*TEKN
  DISK-TEKN     DISK-TEKN2
  KASS*TEKN     KASS-TEKN
  LTKOMMA/1     LTKOMMA/2
  LTKOMMBAS     MAPGROM0
-------------------------------------
(Disk Directory              )
(                            )
-------------------------------------
Press Any Key to Continue
```



Intresse(grupp) för Pascal?

av Anders Persson

I nummer 85-1 skrev jag ett upprop angående intresse för Pascal. Reaktion från medlemmarna: Ingen.

Senare skickades det ut en enkät till er där det bland annat fanns en fråga om intresse för en SIG för Pascal.

Väntat resultat: De personer som jag kände till redan anmälde sig.

Oväntat resultat: Några som inte hört av sig till mig kryssade också för.

Ger jag möjligen ett så skrämmande intryck att ni inte vågade ringa eller skriva till mig direkt?

Nyheter

Nåväl, min aktivitet har ju inte heller gjort så stort väsen av sig när det gäller det här. Det har kanske fått någon att tro att aktivitet har saknats helt och hållet. Så är dock inte fallet, tvärtom. Men vad sker stort, sker tyst, eller vad det heter.

Program

För egna behov har jag utvecklat en del programvara av allmänt intresse. Det rör sig om inläsning från tangentbordet, utmatning på skärmen och utnyttjande av minnet. Dessutom har jag skrivit en del mindre program som kan vara roliga eller nyttiga. Ett exempel är det formateringsprogram som medföljer på Utility disketten. Det påstår sig klara även dubbelsidiga disketter, men det gör det inte. Ett dformat program som klarar allt från SS/SD till DS/DD har jag nu utvecklat. Till skillnad från min tidigare variant fungerar det här verkligen – tror jag...

Information

P-systemet håller sig med ganska mycket tabeller i RAM. Dessa behövs för att hålla ordning på vad som händer och sker. Tabellerna finns huvudsakligen i 8K-delen av minnesexpansionen. En önskan att utnyttja speciella funktioner i operativsystemet ledde till en kartläggning av en del av detta minnesinnehåll.

Tillgänglighet

För att du ska få tillgång till detta har jag ordnat några disketter. Dessa hyser ett antal filer med de program och den text jag beskrivit ovan, plus en del annat. Vissa personer, som jag tidigare haft kontakt med, har redan fått ett erbjudande om dessa disketter. Om du inte fått något sådant, men är intresserad, kan du kontakta mig.

Såvitt jag kan se måste en del av programmen vara idealiska för er som kommit en bit på väg med Pascal, men som ännu inte bemästrat mer avancerade grepp. Assemblerprogram utnyttjas flitigt för att möjliggöra speciella rutiner eller helt enkelt för att få upp farten tillräckligt.

Diverse nu och senare, såsom

Recension

Den amerikanska tidningen MICROpendium hade i februari 1986 en artikel om olika programspråk till 99:an. Pascal var naturligtvis med. Finessen med språkets stöd för strukturerad programmering och implementationens stöd för portabla program togs upp. Nackdelar som nämndes är den höga kostnaden på 99:an (jämfört med BASIC och annat) och svårigheten att komma åt maskinberoende funktioner. Naturligtvis (?) fanns det fel i artikeln också, antagligen beroende på dålig erfarenhet av Pascal. Visst finns det "goto" i Pascal, även om ingen vettig människa använder det.

Time-sharing

Själv har jag försökt övertyga systemet om att det ska klara av riktig multiprogrammering. Det finns ju redan rutiner för parallella processer, men problemet är att systemet inte tillåter riktig time-sharing. Eftersom ATTACH inte är implementerat kan man inte heller skriva egna rutiner för detta i Pascal. Men med hjälp av en hel del assembler gick det faktiskt att lösa. Mer om det i en annan artikel. (Ingen garanti för att den finns i detta nummer av tidningen.)

Grafik

Den som vill roa sig med grafik med hjälp av Pascal har säkert saknat bit-map. Liksom BASIC kan Pascal nämligen inte utnyttja denna finess. Det kan kanske tyckas märkligt, eftersom de andra tre varianterna, text, graphics 1 och multicolor, finns. Orsaken är säkert att TI utvecklade p-systemet redan på TI 99/4, föregångaren till 4A. Bland diverse skillnader märks avsaknaden av bit-map på den första maskinen. Det berodde på att den första versionen av den krets som genererar bilden inte hade bit-map.

Ett problem är att VDP RAM används av p-systemet till diverse olika saker. Bland annat finns inmatningsbufferten här, liksom kod när man kör Pascalprogram. Men eftersom systemet är tämligen flexibelt uppbyggt, med pekare till det mesta av intresse, borde det gå att flytta informationen och ändra på pekarna.

Det visade sig också vara riktigt. Jag har nu en assemblerrutin som ordnar övergången till bit-map, en annan som ritar streck och en tredje som går tillbaka till textmode igen. Jag har även lyckats rädda teckendefinitionerna och textskärmens innehåll, varför den ursprungliga texten på skärmen finns kvar när man kommer tillbaka till textmode. Det som återstår nu är diverse rutiner som behövs för att det ska bli bra att använda. Jag har planer på att göra en Turtlegraphics unit av lite enklare slag än den som följer med vissa IV.0 system. Tiden får utvisa när den blir färdig.

Brist på fantasi brukar jag inte lida av. Fler projekt finns i tankarna, men de är ännu så dimmiga att jag väljer att hålla tyst om dem så länge.

Register för databashantering

av Jan Alexandersson

I Programbiten 1986-1 fanns en förteckning över artiklar i tidigare nummer. Denna lista har matas in med hjälp av Personal Record Keeping och mitt BASIC-program SNABBFIL från Programbiten 1985-2.

Jag använde följande filstruktur:

	fält	typ	bredd	dec
1	ARTIKEL	CHAR	15	0
2	KÄLLA	CHAR	13	0
3	NUMMER	CHAR	7	0
4	SIDOR	DEC	4	1
5	SPRÅK	CHAR	5	0
6	TYP	CHAR	3	0
7	LISTNING	CHAR	1	0

Jag gjorde tre olika filer, en för varje årgång. När de var färdigrättade lagrade jag de på flexskiva även om det gått lika bra med kassett. Därefter gick jag tillbaka till menyn och nr 2 Personal Record Keeping. Där tog jag in filerna på nytt och sorterade efter två olika fält, först nummer och sedan språk. Sorteringen är mycket långsam. C:a 80 poster sorteras på 20 minuter.

När det sedan var dags för utskrift valde jag Personal Report Generator. Det går visserligen att använda PRK men den skriver mycket långsamt. Datorn tar dubbelt så lång tid på sig som printern. Den andra modulen är mycket snabbare.

Personal Report Generator kan förutom till utskrift även användas för MERGE av flera filer, DELETE av många poster i ett svep, DELETE eller ADD av fält.

Forth-spalten

Redaktörens adress:
Lars-Erik Svahn
Mellanbergsvägen 23, 1 tr
135 45 Tyresö
Tel: 08/742 61 07

Iteration och rekursion

En intressant och användbar möjlighet i FORTH (som inte finns i tex Basic), är rekursion. Med rekursion menas att en rutin anropar sig själv! Observera, att det inte är fråga om anrop av GOTO-typ, utan mera av GOSUB-typ eller CALL-typ.

Rekursion är ett mer avancerat sätt att åstadkomma upprepning i ett program. Det vanliga sättet är iteration (exempelvis med DO-LOOP, BEGIN-UNTIL, BEGIN-WHILE-REPEAT, etc).

Visserligen är det matematiskt bevisat att varje rekursivt program kan omvandlas till ett iterativt program (och omvänt!), men många programmeringsproblem kan lösas synnerligen enkelt och elegant med hjälp av rekursion (se nedan). Till vissa uppgifter passar iteration bäst, och till vissa rekursion. Det är en klar fördel att kunna välja. Särskilt i FORTH! I en del andra språk är det nämligen svårt att implementera rekursion på ett effektivt sätt. Men i FORTH är iteration och rekursion praktiskt taget lika effektiva, mycket beroende på FORTHs arrangemang med de två stackarna, direkt åtkomliga med parametrar och returadresser.

Det vanligaste sättet att göra rekursion i FORTH är via ordet MYSELF. Den definition som innehåller MYSELF anropar sig själv. Därmed lagras en returadress i returstacken, och när MYSELF har "exekverat färdigt" (kan den inte det så är det något fel på den rekursiva rutinen!), så återvänder programmet till ordet efter MYSELF.

MYSELF är konstruerat som ett immediateord som kompilerar kodfältsadressen till det senast definierade ordet i ordlistan:

```
: MYSELF LATEST PFA CFA , ; IMMEDIATE
```

Ö LATEST ger nfa som omvandlas till cfa via pfa.

Ett enkelt (men i princip dåligt) exempel på rekursion är programmet:

```
: FAKULTET ( n -- n! ) -DUP
IF DUP 1- MYSELF * ELSE 1 THEN ;
```

Detta är bara den matematiska definitionen på fakulteten, formulerad i FORTH. Den vanliga definitionen lyder:

n!=n*(n–1)! om n>0 och 0!=1.

Det principiellt dåliga med rutinen FAKULTET är att den först lagrar n+1 st returadresser på returstacken, samt n+1 parametrar (n,(n–1),...,2,1,0) på parameterstacken, och *sedan* börjar multiplicera!

FAKULTET gör sig därför mycket bättre i en iterativ version:

```
: FAKULTET ( n -- n! ) 1 SWAP -DUP
IF 1+ 1 DO I * LOOP THEN ;
```

I andra fall, när parametrarna och returadresserna på stackarna inte är "död last" kommer rekursion mer till sin rätt.

Principen för rekursiva lösningar av programmeringsproblem kan kanske sammanfattas:

1. Förenkling av problemet (beräkna (n–1)! i stället för n!)
2. Konstruera lösningen för hela problemet med hjälp av lösningarna för de förenklade problemen (n!=(n–1)!*n)
3. Ange de enklaste fallen explicit (0!=1).

Det är viktigt att de enklaste fallen är angivna, för det är via dessa angivelser som rekursionen avslutas och returstacken börjar poppas (*eng.* terminal cases).

Det är lika viktigt att förenklingarna är sådana, att de enklaste fallen uppnås efter ett ändligt antal upprepningar (det är förstås stackarnas djup som sätter gränsen).

Program med artificiell intelligens utnyttjar ofta den rekursiva programmerings-strategin. I AI-språket LISP används nästan enbart rekursion för upprepning.

Dynamiska datastrukturer och mera rekursion

I många sammanhang är det lämpligt att använda *dynamiska* datastrukturer, som tex listor eller träd. Till skillnad från *statiska* datastrukturer (som fält av DIM-typ och vanliga ascii-strängar) behöver de dynamiska inte vara av bestämd maximal storlek. Storleken kan istället tillåtas att växa under programkörningen, allt efter behov.

Detta är möjligt därför att dynamiska datastrukturer är uppbyggda av enheter av en viss typ. De kallas i allmänhet celler. De utgörs av minnesblock med datafält och länkfält.

För att undvika sammanblandning med FORTH-termen cell kallar jag dessa enheter *fack* här.

Datafälten innehåller det som ska lagras (eller pekare till dessa data) och länkfälten innehåller adresser till andra fack. Facken behöver inte ligga samlade på någon särskild plats i minnet, utan kan skapas (det kan göras programstyrt med en särskild teknik för att administrera sådana fack. Jag avser att skriva om det i en kommande spalt).

Statiska datastrukturer hanteras ofta bäst av iterativa rutiner. Dynamiska datastrukturer däremot hanteras ofta bäst av rekursiva rutiner.

För att bedöma vad rekursion har att erbjuda som programmeringshjälpmedel kan vi titta på följande exempel (se även figur):

Du har ett *binärt träd* av strängar och vill skriva ut alla strängarna. Varje fack består av 3 ord (dvs 6 bytes):

- 1:a ordet innehåller adressen till det första facket i den "vänstra" förgreningen (eller 0 om ingen sådan förgrening finns)
- 2:a ordet innehåller adressen till strängen (dimensionerad sträng i cpu-ram)
- 3:e ordet innehåller adressen till motsvarande "högra" förgrening (eller 0)

Det "översta" facket (som inget annat fack pekar på) kallas för trädets *rot*. Man anger ett träd genom att ange adressen till dess rot.

De "nedersta" facken, vilka inte har några förgreningar, kallas *löv*.

Följande rekursiva program löser problemet:

```
.TREE ( rot -- )
  DUP @ –DUP IF MYSELF THEN    ö ord 1
  2+ DUP @ CR COUNT TYPE       ö ord 2
  2+  @ –DUP IF MYSELF THEN ;  ö ord 3
```

Märk att strängarna skrivs ut från "vänster" till "höger".

Det vore intressant att se en iterativ motsvarighet till denna rutin, om någon känner sig manad . . .?

Programmets idé är att varje förgrening i sin tur är ett binärt träd, och att samma program kan användas för att genomsöka (traversera) detta träd – förenklingsmomentet! Löven svarar mot de enklaste fallen (binära träd utan grenar!).

I detta exempel utgör de data som lagrats på stackarna ingen "död vikt". Här är det nödvändigt att lagra adresserna till alla förgreningar, så att sökandet kan återupptas i närmaste ogenomsökta förgrening sedan ett löv har hittats.

En iteration är alltid lätt att överblicka i sin helhet (även om själva programmet skulle vara svårt att överblicka).

En rekursion *kan i regel ej* överblickas! De dynamiska datastrukturerna tex styr programflödet olika från fall till

fall. Däremot blir själva programmen ofta lättare att överblicka.

När Du konstruerar rekursiva rutiner måste Du tänka lite annorlunda än vid de vanligare iterativa. Istället för total överblick över rekursionen får Du nöja dig med en lokal överblick:

När Du skriver MYSELF, ska Du låtsas att den rutin Du just definierar redan är färdig att användas!

Därför är det viktigt att ha klart för sig *exakt vad* rutinen skall åstadkomma och vad den har för stackdiagram. Detta ska ju också MYSELF åstadkomma och detta stackdiagram måste ju också MYSELF ha.

För alla eventuella tvivlare på rekursionens lungnande inverkan på nerverna visar jag ännu ett exempel.

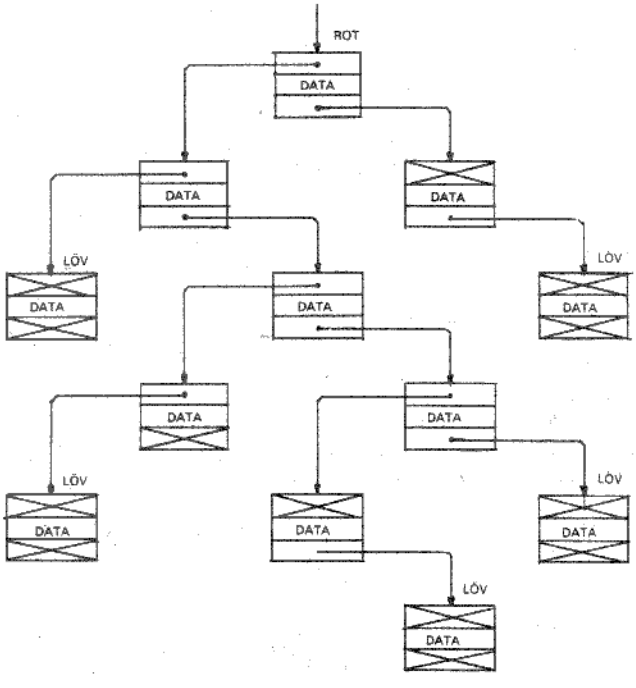
Vi tar rutinen FAKTORER som faktoriserar ett heltal>0 i sina primfaktorer. Strategin är den ovan beskrivna:

- Genom att leta efter faktoruppdelningar förenklas problemet (27=3*9). Faktorisera sedan 9. Den minsta faktorn är alltid ett primtal!).
- Faktorer som visas vara primtal blir "terminal cases".

```
: FAKTORER ( u --  p1..pn) >R 2
BEGIN DUP R 0 ROT U/ SWAP
WHILE DROP 1+
REPEAT OVER R =
IF DROP DROP R>
ELSE R> DROP MYSELF
THEN ;
```

REKURSION + FORTH = SANT

BINÄRT TRÄD



Modemkontakt önskas

Sten-Inge Gunnarsson i Göteborg vill kommunicera med andra 99-ägare via modem. Han har Terminal Emulator II och kan nås via telefon 031/917004. Adressen hans är: Box 8897, 402 72 Göteborg.

RGB-Modulator

Av Lennart Thelander

När jag nyligen köpte en ny TV passade jag på att köpa en TV med RGB-ingång (en Philips Matchline). Jag köpte också en RGB-Modulator. RGB står för Röd Grön Blå, de färger en TV-bild är sammansatt av. En teknisk beskrivning kommer sist i denna lilla artikel. Med denna ingång och RGB-Modulatorn får man en mycket skarpare bild än vanligt. Visserligen blev färgerna litet blekare, men det är ingen större nackdel. Nu kan jag se varje pixel tydligt även om det är blått och rött intill vartannat. Pröva röd text på blå bakgrund på era TV-apparater och se hur det grötar ihop sig. En RGB-Modulator ser ut som den vanliga antennmodulatorn, men med annan kontakt på utgången (och givetvis annat innehåll i burken). Kontakten är en s.k. SCART- eller EURO-kontakt som finns på de flesta moderna TV-apparater. Men se upp!!! Långt ifrån alla TV-apparater med SCART-kontakt har RGB-ingång. De flesta har bara videoingång. Modulatorn kostar ungefär 400 kronor, men det är väl använda pengar. Med en enkel justering i TV:n kan jag se alla 32 kolumnerna. Som väl alla har sett kan det vara problem med just detta. Egentligen är det ingen modulator utan en signalomvandlare. Mitt betyg om RGB-modulatorn blir alltså: *bra* köp för en som har videoingång. De som var på medlemsmötet i Staffanstorp den 8/2 kan intyga detta. De många som inte var där gick miste om en hel del. (Se där, nu fick jag sagt det också).

Hur kommer det en bild ur TV:n? Här kommer nu en teknisk beskrivning över olika sorters videosignaler för den som orkar läsa det. Ut ur datorn kommer tre signaler: Y, R-Y och B-Y. Y är den svartvita signalen med bildsynk och linjesynk. Har ni en svartvit (monokrom) monitor kan ni ta ut denna signal till monitorn. De två andra signalerna är färgsignaler. R-Y betyder Röd minus "Yellow" (egentligen SV/V). B-Y är då alltså Blå minus SV/V. Dessa tre signaler bestämmer alltså fullständigt hur en TV-bild ser ut. I RGB-burken omvandlas dessa tre signaler till fyra stycken: synk, röd, grön och blå. De tre färgerna körs sedan mer eller mindre direkt in i TV:ns färgkanoner och synk ser till att bilden inte flimmrar eller hoppar. Det här är det bästa sättet att få en färgbild på. Om man vill kan de bägge färgsignalerna slås ihop till en. Den signalen med Y-signalen bildar då de signaler som t.ex. Vic-20 och 64 (och 128) lämnar ifrån sig. De signalerna kallas för luminans repektive kroma-signal för Y resp. färgsignalen. Det finns monitorer som tar in de här signalerna, men de är ganska ovanliga. Om man sedan blandar kroma med Y får man en komplett videosignal. (För den insatte: färgsignalen moduleras på den svartvita med 4.43 Mhz. Det är den frekvensen televerkets tv-pejling är ute efter. Signalen finns i alla färgtv-apparater. Det bekymrar inte televerket att en hel massa andra apparater också använder denna frekvens.) Denna kompletta videosignal kan kopplas in i de flesta tv-apparater (i alla fall de som är yngre än fem år. Det står i bruksanvisningen hur denna kontakt ser ut, för det varierar ganska mycket.) Denna kompletta videosignal finns i den vanliga modulatorn till 99:an. Det är bara att plocka ut den om man vet varifrån. Nu har tre signaler bakats ihop till en enda. Men det är inte slut ännu. Nu kommer också ljudet in. Ljudet moduleras med 5.5 Mhz och läggs på videosignalen. Nu har vi alltså fyra signaler. Dessa moduleras sedan till kanal 36 för tv. Det motsvarar ungefär 600 till 700 Mhz. Jämför detta med FM-radio på knappt 100 Mhz. Denna komplicerade signal går sedan till TV:n där den sedan plockas isär till sina beståndsdelar igen. Först ljud, luminans och kroma och sedan till ljud, synk och RGB. Undra på att RGB-ingången ger en bättre bild . . .

(RGB-modulatorn är tillverkad av TI och har numret PHA 2037. Vid en telefonkontroll i slutet av april fanns den till salu hos *Digitalservice* i Stockholm, tel 08/41 60 52. Priset var 350 kr med moms. – *Red.)*

Produktkostnads-beräkning – en FORTHifikatorisk lustifikation på allvar

av Lennart Lindberg

Den här lilla rutinen tycker jag är toppen faktiskt – om nu den sortens entusiastiska språk kan tillåtas. Jag har hittat den i *Forth Dimensions** – som så mycket annat – nummer 4 av volym V, sidan 9. Artikeln är skriven av Marc Perkel från Springfield i Missouri. Alltihop finns på tre skärmar.

Den första skärmen innehåller själva programmet, som är ett par ord som räknar ut summor och adderar till en total-ackumulator (+TOTAL och DUSSIN) samt några definierande ord som skapar klasser av andra beräkningsord (APRIS och KOSTNAD) samt en klass som laddar Forth-skärmar (LADDA/KÖR). Ordet SUMMERA styr slutfasen av hela jobbet. Beräkningarna görs i dubbelstals-aritmetik.

Den andra skärmen skapar ett antal ord av två av de nämnda klasserna – APRIS, som gör ord för de varuslag som skall ingå i produkten och KOSTNAD, som gör ord för de kostnader som inte är direkt styckberoende utan knutna till leveransen. APRIS behöver på stacken just à-priset medan kostnad behöver värdet av just den insatsfaktorn per leverans. På skärm nummer ett skapas ett ord som laddar skärm två – PRISER.

Den tredje skärmen utnyttjar de ord som skapades på skärm två. De som bildas av APRIS hämtar ett antal från stacken, multiplicerar med à-priset från skärm två och adderar till totalen. De som bildas av KOSTNAD lägger värdet/kostnaden från skärm två för det aktuella kostnadsslaget till totalen. Ordet SUMMERA, som skapades på första skärmen beordras från tangentbordet och styr hela förloppet med den tredje skärmen inklusive laddningen av skärm tre med ordet FRUKTKORG samt avslutar med att skriva ut totalen för produkten/leveransen.

Det fiffiga med det här tycket jag är enkelheten. Den andra och den tredje skärmen fungerar dels som en del av programmet, dels också som en prislista, läsbar och möjlig att rätta och underhålla för en användare, och en produktbeskrivning, också den läsbar och möjlig att underhålla för användaren. Han sätter in priserna på skärm två och hittar själv på namnen på delarna – frukterna i exemplet. På skärm tre måste han sedan använda samma namn, vilket är det enda villkoret utöver att han måste – för dem som har ett à-pris – ge ett antal. Sedan är det bara att ladda skärm ett, beordra PRISER och beordra SUMMERA FRUKTKORG. Han kan fö ge skärm tre vilket namn han vill genom att byta namn på rad 13 i skärm ett. Ordet FRUKTPRIS kan vara vilket som helst om det skulle kännas fånigt.

Perkel påstår att rutinen används av en specerihandlare någon stans som har en hemdator i butiken.

Ordet som skriver ut ett dubbeltal som ett krontal – KR. – tror jag kan ha en allmän användning.

* Forth Dimensions är en tidskrift som ges ut av Forth Interest Group i USA.

```
SCR #30
  0 ( Prissättning. FD V/4/9. Fruktkorgen )
  1 : 2VARIABLE VARIABLE 2 ALLOT  ;
  2 0. 2VARIABLE TOTAL
  3 : .KR  <# ASCII R HOLD ASCII K HOLD 32 HOLD
  4        # # ASCII : HOLD #S #>  TYPE SPACE  ;
  5 : +TOTAL TOTAL 2@ D+ TOTAL 2!  ;
  6 : APRIS    <BUILDS , , DOES> 2@ >R OVER U* ROT R> * + +TOTAL ;
  7 : KOSTNAD  <BUILDS , , DOES> 2@ +TOTAL  ;
  8 : LADDA/KÖR  <BUILDS , DOES> @ LOAD ;
  9 : DUSSIN   12 *  ;
 10 : SUMMERA  0. TOTAL 2! ACOMPILEA ' >R R CFA EXECUTE
 11            CR R> NFA ID. ." KOSTAR "  TOTAL 2@ .KR  ;
 12 1 FH LADDA/KÖR PRISER
 13 2 FH LADDA/KÖR FRUKTKORG   : FRUKTPRIS ;    PAGE
 14 ." Kommando PRISER exekverar pris-skärm " CR
 15 ." SUMMERA  FRUKTKORG laddar mtrl-skärm" CR ." o ger summan" CR

SCR #31
  0 (    Priser i Fruktkorgen )
  1 FORGET FRUKTPRIS         : FRUKTPRIS  ;
  2 7.50  APRIS APPLE
  3 4.80  APRIS BANAN
  4 8.30  APRIS APELSIN
  5 0.25  APRIS KOERSBAER
  6 12.70 APRIS GRAPE
  7 8.90  APRIS PAERON
  8 5.30  APRIS PERSIKA
  9 .10   APRIS DRUVA
 10
 11 20.00 KOSTNAD KORG
 12 12.30 KOSTNAD PACKNING
 13 42.50 KOSTNAD FRAKT
 14 10.00 KOSTNAD HANTERING
 15

SCR #32
  0 (    Material i Fruktkorgen )
  1
  2 5 PAERON
  3 8 APELSIN
  4 3 GRAPE
  5 2 DUSSIN KOERSBAER
  6 3 DUSSIN DRUVA
  7 6 PERSIKA
  8 4 APPLE
  9 7 BANAN
 10
 11 KORG
 12 PACKNING
 13 HANTERING
 14 8 FRAKT
 15

TI FORTH --- a fig-FORTH extension       21jul86LL
```


THE SMART PROGRAMMER

De läsare som känner igen tidningen *The Smart Programmer* blir nog glada över meddelandet att tidningen nu ska utkomma regelbundet igen. Den gavs ju från början ut av *Millers Graphics*, en välkänd firma inom 99-kretsar. De hade dock stora problem med att hålla utgivningstakten. Nu har den slagits ihop med en annan amerikansk tidning, *Super 99 Monthly*. Denna har givits ut regelbundet i ett par år och har samma inriktning som The Smart Programmer, tekniskt djupgående artiklar och mycket programlistningar i Forth och assembler.

Tidningen behåller namnet The Smart Programmer och ska komma ut varje månad. När detta skrives har juni- och juli-numren nått Sverige. Om resten håller samma klass blir det säkert intressant.

En prenumeration kostar 20 eller 32 dollar per år, beroende på om man vill ha den levererad per båt eller flyg.

Adressen är:
Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663
USA

Från pixel till punkt

av Sören Bernle

Ett screen-dump-program i x-basic, som jag tidigare snickrat ihop, krävde 1/4 timme i tid för att printa ut en skärm. Att jag hade önskemål om ett snabbare program förstår Du nog, och att detta skulle innebära ett program skrivet i assembler förstod jag. Jag förstod även att mina kunskaper i detta ädla språk, var allt för klena för att klara av en sådan uppgift.

Men så en dag, fick jag av Arne Wennberg i L-krona, ett artikelutdrag ur *The Best of 99'er,* med listning och beskrivning av ett screen-dump-program skrivet i line by line assembler för minimemory och avsett att kunna dumpa basicskärmar.

Efter att ha läst och helt tagit för sanning vad jag läst, gav jag mig i kast med att skriva ett program i assembler (äger ej minimemory).

Som vanligt, begrep jag inte allt vad som stod i artikeln och listningen uppfattade jag som rörig och körtidskrävande. Så med hjälp av artikel-listningen och ett idogt manualbläddrande, lyckades jag åstadkomma ett program som dumpade assembler-skärmen.

Mina försök att modifiera programmet, till att dumpa x-basic skärmar, stupade på att det inte finns någon färdig DSRLNK-rutin att tillgå. Jaha vad gör man då?

Jo, man tar kontakt med någon av sina mer kunniga TI-99 vänner och ber om *HJÄLP!!!!!*

Någon dag senare, efter detta nödrop, ligger det ett gyllenbrunt kuvert i min brevlåda, innehållande en diskett med mycket behövliga subrutiner.

Vem var då denna hjälpare i nödens stund? Jo, Anders Persson ifrån Lund!

Och si, efter ett tags knåpande (jag är rätt bra på att röra till det för mig "ibland" nämligen), spottade printern ut x-basic skärmar som bara den!

Nåväl, drygt ett halvår senare, funderade jag på att försöka få programmet publicerat i PB.

Men publicering kräver programkommentering, vilket helt saknades.

För att kunna kommentera, fick jag tänka igenom programmet ånyo. Jag tyckte då att det verkade milt sagt rörigt, men det var inte det värsta. Det värsta var att jag inte begrep de beräkningar som ledde fram till var karaktärskoden var lagrad! Jag mindes nu att det begrep jag inte heller när jag skrev programmet! Jag läste artikeln om och om igen, och fick till sist den hädiska tanken, att jag kanske var klok som inte begrep.

För att bli klok på om jag var något klok, ringde jag ovan nämnde Anders och fick beskedet att jag var något klok.

Stärkt av det ovanstående, började jag genast skriva ett helt nytt program. Se listningen.

Angående DSRLNK-rutinen, så har Anders Persson gett mig sitt muntliga tillstånd till att den får publiceras i PB.

Hoppas Du på att den nedanstående programbeskrivningen även skall förklara hur DSRLNK-rutinen fungerar?

Känn Dig då, som jag, *BLÅST!*

Jag begriper inte ett dugg!

Programbeskrivning:

Denna beskrivning, riktar sig främst till Dig som har mycket måttliga (dock några) kunskaper i assemblerprogrammering.

Beskrivning och resonemang, följer listningen rakt uppifrån och ned.

Alla hänvisningar avser E/A-manualen, om ej annat angivits.

DEF START, tilldelar programmet anropsnamnet START (se 14.4.1).

Programmet läser skärmen från övre vänstra hörnet till nedre högra hörnet, en karaktär i taget, och överför varje tänd pixel till en punkt på papperet.

Informationen om vad som finns på skärmen är lagrad i VDP-RAM. För att nå denna minnesarea, krävs subrutiner typ VSBR etc. Men eftersom x-basic laddaren inte tillåter några REF:s, måste dessa istället equaliseras (se appendix). GPLWS (se *Utmaningen* PB 85-4 sid 8.)

DSRLNK-subrutinen, kräver en PAB samt en PABBUF och i detta program dessutom WRITE och CLOSE värden (se kapitel 18).

För att få utprintningen helt skärmliknande och för att undvika tomma punktlinjer mellan var rad, måste printern ställas i 8/72 tums radhöjd. Koderna för detta finns vid label'n INGR.

När programmet har slutfört sin uppgift, återställes printern i normal-mode. Vid label'n ENDUP finns koderna för detta.

Därefter följer 2 data-satser, med koderna för vagnretur och radframmatning. Den senare av dessa, med labeln NEXROW, utgör det första word'et av den 3 word långa printer-kommando-satsen, som printern måste ha innan utprintning av ny rad kan ske.

Eftersom programmet skall överföra varje tänd pixel till en punkt på papperet, måste printern ställas i 8 punktersgrafik-mode. Varje byte printern matas med i detta mode, innehåller information om en punkts radbredd och 8 punkters radhöjd, vilket överensstämmer med skärmkaraktärens pixelhöjd. Andra datasatsen efter NEWROW, innehåller denna kod.

Enligt ovanstående, framgår att det krävs 8 bytes för att printa ut en karaktär. En skärmrad är 32 karaktärer lång, alltså krävs det (32x8) = 256 bytes för att printa ut en hel skärmrad. Printern måste få information om hur många bytes den skall tolka som grafik-bytes och inte som ASCII-värden. Tredje datasatsen efter NEWROW, innehåller värdet för hur många bytes som skall tolkas som grafik-bytes.

BUFF är det minnes-område, där de ovan nämnda 256 grafik-bytes'en successivt lagras.

OUT är det 8 byte stora minnes-området, där karaktärskodningen för printern byggs upp. Att jag här har valt direktivet DATA istället för BSS, beror på att dessa 8 bytes måste vara tomma i starten av körningen.

PDATA (se kapitel 18).

IN är det 8 byte stora minnes-området, som innehåller den aktuella skärmkaraktären under avkodningssekvensen.

BSCSTO är det minnes-område, där under körningen, eventuella delar av x-basic programmet i VDP-RAM lagras.

MYREG (se START, här nedan).

START. Här börjar programkörningen med att skifta över till ett eget workspace. (se Utmaningen i PB 85-4 s. 8)

Därefter sparas innehållet på VDP-adresserna >F80 tom >1083 i BSCSTO. *Skäl:* Om någon del av x-basic programmet ligger på dessa adresser, blir det överskrivet och förlorat.

Vidare följer instruktionsblocket, som öppnar filen. Nästa block ställer filen i skriv-mode. (se kapitel 18)

Nästa block åter igen, ställer printern i så tät radframmatning, att karaktärerna nuddar varandra. *Skäl:* Se ovan!

Därefter ökas PAB:ens Character counter, från 3 till 131 bytes. *Skäl:* För var rad skall det till printern överföras 262 bytes (CR + LF + 4 grafik-mode-bytes + 256 grafikbytes). Detta är 7 bytes fler än vad PAB:en klarar av. Därför får Character counter sättas till halva bytesmängds-värdet, och utprintningen får istället ske i två omgångar.

Skärmens innehåll är lagrat i Screen Image Table (i forts förkortat SIT). Denna tabell börjar på adressen >0000 i VDP-RAM och innehåller 768 bytes, som vart och ett är bundet till sin unika skärmposition. Dvs att byte >0000 står för karaktären längst till vänster på översta raden, >001F = 31 står för karaktären längst ut till höger på översta raden, >02FF = 767 står för karaktären längst ut till höger på den nedersta raden.

Alltså! Att läsa i denna tabell från >0000 till >02FF är det samma som att läsa skärmen från det övre vänstra hörnet till det nedre högra. Dock med det stora undantaget att i denna tabell får man endast veta värdet på den faktor som multiplicerad med 8 ger adressen till det första bytet av de 8 på varandra följande karaktärs-beskrivande-bytes'en.

Den ovan nämnda faktorn, består av ASCII-värdet + 96.

Eftersom x-basic endast tillåter skärmkaraktärer inom ASCII-värdeområdet 30−143, betyder detta att Pattern Descriptor Table (i forts förkortat PDT), skall starta på (30+96)x8 = 1008 = >03F0 och sluta på (143+96)x8+7 = 1919 = >77F. Det är just vad den gör! (Se sid 10 i PB 84-5.)

R4 är det register som pekar på vilket byte i SIT som står i tur att läsas och eftersom tabellen skall läsas från 0 − 767, nollställes R4.

R8 är det register som innehåller det aktuella mönster-beskrivnings-bytet under avkodningen. Detta registers högra byte måste vara 0-ställt i starten.

NEXROW är startpunkten för ny rad, varför R5 laddas med start adressen till BUFF.

NEXCHR. Från SIT i VDP-RAM hämtas det byte, som R4 pekar på. SRL-instruktionen flyttar bytet från MSByte till LSByte samtidigt som MSByte blir 0-ställt.

Enligt tidigare resonemang, skall R1 nu multipliceras med 8, detta utförs med instruktionen SLA.

R1 innehåller nu adressen till det första av de 8 sökta mönster-beskrivnings-bytes'en som är lagrade i PDT.

De följande fyra instruktionerna, kopierar över dessa 8 bytes till IN-bufferten

När vi nu äntligen fått tag i karaktärsmönstret, så visar det sig, att det är lagrat, så att säga på fel håll!

I datorn är karaktärerna kodade horisontellt, medan printern kräver att de skall vara kodade vertikalt.

Koden är lagrad så, att det första bytet (= lägsta adressen) innehåller hexa-koden för karaktärens översta del osv. ned till det sista bytet i gruppen (= högsta adressen) som innehåller hexa-koden för karaktärens nedersta del.

Alltså, om någon bit är tänd i det första bytet, "vill" printern att det skall läsas som >80 = 128, osv. ned till det sista bytet, där printern "vill" läsa var tänd bit som >01 = 1.

SE UPP! Nu kommer jag in på liknelser. Man kanske virrar bort sig!

Tänk Dig nu, att karaktärsmatrisen (8x8), är förvandlad till 8 långa, på varandra staplade lådor, och att var låda har 8 fack i rad. Dessa facks eventuella innehåll, har i den understa lådan värdet >1 = 1. För vart steg upp i lådstapeln är detta värde fördubblat. Dvs att fackens eventuella innehåll i den översta lådan har värdet >80 = 128.

Tänk Dig vidare, att var låda är skjutbar åt vänster och att så snart ett fack kommit ut i det fria, faller fackets eventuella innehåll ut.

Tänk Dig nu ännu vidare, att jag rakt inunder denna lådstapel, på en i sida skjutbar slid, ställer upp 8 hinkar i rad.

Tänk Dig nu, om möjligt ännu vidare, att det är så finurligt anordnat, så att när Du kommer fram till denna remarkabla anordning, att hinksliden och de 8 lådorna står i ultima höger position, samt att första gången Du för hinksliden till ultima vänster, följer den nedersta lådan med och tömmer sina facks eventuella innehåll i respektive hink. Så snart Du nått ultima vänster, drar Du sliden ultima höger, varpå anordningen fattar tag i låda §2 nedifrån osv. Dessa ultimativa vänster-höger rörelser upprepas i allt 8 ggr.

Och hokus pokus, i dessa 8 hinkar ligger nu printerkoden prydligt samlad, ett byte i var hink.

I programmet, är dessa lådor med sina fack, jämförbara med IN-bufferten och hinkarna med OUT-bufferten. Slut på liknelsen!

Det vänstra bytet i R6 innehåller det aktuella bit värdet i avkodnings-loopen (R6 motsvarar alltså i liknelsen här ovan, det enskilda värdet i vart lådfack). Eftersom avkodningen börjar med bytet som har den högsta adressen (nedersta bit-raden i karaktären), så laddas detta registers vänstra byte med värdet >01.

NEXBYT. R2 är pekaren för vilket byte i IN bufferten, som skall avkodas. Det skall i starten av avkodningen, som tidigare sagts, peka på bytet på högsta adressen (lägsta bit-värdet) vilket är lika med att R2 har värdet >7. Eftersom R2 tidigare blivit laddat med >8, tre instruktioner bakåt i programmet, och här via instruktionen DEC blir minskat med 1, så får R2 rätt startvärde.

Det byte som skall avkodas, flyttas nu över till det vänstra bytet, i det tidigare 0-ställda R8. Om detta byte skulle visa sig vara tomt, är det helt onödigt att avkoda det, det skulle bara kosta tid.

Vid ett tomt R8, hoppar därför programmet över, bit för bit avkodningen, till BITUP.

Men om bytet i R8 ej är tomt, så laddas R7 med OUT som är adressen till det första bytet i OUT-bufferten.

NEXBIT. Enligt liknelsen ovan, sköts var låda åt vänster och dess eventuella innehåll föll ut och samlades upp i hinkar. Det är just vad som så att säga sker här. Första hinken på sliden, är förvisso redan utskjuten på sin plats (LI R7,OUT). Instruktionen SLA R8,1 flyttar samtliga bitar i R8 ett steg åt vänster, och om MSB var satt innan operationen så blir carry-biten satt. Men å andra sidan, om MSB ej var satt innan operationen så blir carry-biten ej satt.

Är carry-biten satt, så hoppar programmet till BITON där värdet i R6 adderas till det aktuella OUT-bytet och återhopp sker till NEXBIT.

Är carry-biten ej satt, så checkas det om R8 är tomt.

Är R8 ej tomt, så fortsätter avkodningen. För att R7 skall hålla takten (rätt hink till rätt fack) ökas det med 1 via INC R7-instruktionen. Därefter görs återhopp till NEXBIT.

Är R8 tomt, så betyder det antingen, att de resterande bitarna av de ursprungliga 8 ej är satta. Eller att samtliga bitar har blivit checkade. Förklaring till det senare alternativet: R8 blev tidigare i programmet 0-ställt och därefter har endast värden tillförts till det vänstra bytet. Eftersom SLA operationen flyttar alla bitarna en position vänster och fyller på med nollor från höger, blir effekten den, att när det gjorts 8 SLA operationer är hela R8 0-ställt.

Om nu R8 är 0-ställt, hoppar programmet till BITUP.

BITUP. Efter att värdet i R6, via SLA-instruktionen, blivit fördubblat, göres här ett återhopp till NEXBYT. För såvida R6 inte innehöll värdet >8000 innan SLA, för om det gjorde det blir värdet i R6=>0 och då har också samtliga byte's i IN bufferten blivit avkodade och något återhopp blir inte aktuellt.

När samtliga karaktärsbyte's är avkodade, (R6=0), och ligger snyggt uppsamlade i OUT-bufferten, så skall dessa överföras till radbufferten BUFF.

R7, tilldelas adressen till det första bytet i OUT-bufferten och GOBUFF-loopen överför därefter samtliga 8 bytes till BUFF.

När denna överföring är slutförd, skall också samtliga bytes, i OUT-bufferten, vara 0-ställda. (Om inte OUT-bufferten var rensad inför var karaktärsavkodnings-omgång, skulle den efter var sådan omgång, innehålla helt galna värden.)

INC R4, får karaktärspekaren att peka på nästa byte i SIT som står i tur att avkodas.

När GOBUFF-loopen ovan snurrat färdigt, pekar värdet i R5 (*R5+) på nästa byte i BUFFen, som står i tur att ta emot avkodade värden. R5 får alltså här, sitt rätta värde inför nästa GOBUFF start.

Som tidigare sagts, innehåller en rad 256 bytes. I BUFFen ligger det första bytet på adressen BUFF och det sista bytet på adressen BUFF+255. Om nu 256 bytes blivit avkodade och lagrade i BUFF, så skall värdet i R5 vara lika med (BUFF+256) = BUFF + >100.

CI-instruktionen checkar om radbufferten BUFF är full.

Är den ej full, hämtas ny karaktär via JNE NEXCHR-instruktionen. Men är den full, dvs en skärmrad är avkodad, så skall den printas ut. Denna utprintning måste göras i en två varvs-loop (*skäl:* se ovan).

R5 är loop-räknare och laddas med värdet 2.

De 6 viktiga printer-kommando-bytes'en, har i programmet lagts på adresserna direkt före BUFF-adressen.

Genom att sätta PABBUF:ens startadress till NEWROW, så slinker de lätt och enkelt med, vid var utprintning.

Eftersom värdet i R2, vid vart varv loopen gör, adderas till R1, måste R1 subtraheras med samma värde, för att få rätt startvärde i loopen.

Efter att PRIOUT-loopen snurrat färdigt, undersöks via CI R4,768 om alla karaktärer har blivit avkodade.

R4 ökas med ett efter var karaktärs-avkodning, dvs att om R4 = 768 så har karaktärerna 0 − 767 blivit avkodade.

Vilket i sin tur betyder att hela skärmen blivit avkodad, mao uppgiften är slutförd.

Om R4 ej är lika med 768, fortsätter avkodningen via återhopp till NEXROW.

När dumpningen är slutförd, ändras PABens character counter värde till 6. Så att endast 6 bytes av PABBUFens innehåll sändes till printern.

I instruktionsblocket därefter tillföres printern de 6 bytes, som ligger på start adressen ENDUP. Dessa 6 bytes, ställer printern i normal-mode och ger order om 2 fullhöjds radframmatningar.

I de följande 2 instruktionsblocken, stänges filen (se kapitel 18), och det sparade innehållet i BSCSTO återföres till VDP-RAM.

Till sist, göres återhopp till x-basic-programmet (se 24.11.3).

Saknar Du kunskaper i hur man laddar och kör igång en assembler-subrutin från x-basic? Se då sid 8 i PB 85-4 och/eller E/A-manualen kapitel 17.

Printer-koder:

Printer-koderna i det listade programmet, är anpassade till en parallell-kopplad Taxan Kaga KP-810 printer. (Denna skall vara Epson-kompatibel.)

För att anpassa programmet till Din printer, måste kanske en hel del av dessa koder ändras, ja kanske rentutav alla.

Om Du behöver ändra dessa koder. Checka med Din printer-manual vilka värden Din printer vill ha.

Checka även med E/A-manualen, angående värdena i PDATA och dess Character counter.

För att underlätta dessa eventuella ändringar i programmet har alla instruktioner, som är direkt eller indirekt printer-påverkande, markerats med (!!!), i kommenteringsfältet.

Men seriellt kopplad skrivare då?

Då kan Du ändra i programmet på följande sätt. Ändra TEXT-direktivet efter labeln PDATA ifrån 'PIO' till 'RS232.BA=9600.DA=8'. Klarar inte din printer så hög baudrate så ändra till något lämpligare. När namnlängden nu har ökat från 3 till 18 tecken så måste Du också ändra från >003 till >0012 i DATA-satsen på raden ovanför. (Se kapitel 18.) Den ökade namnlängden medför att PABen har utökats med 15 bytes. Därför måste förstås instruktionen LI R2,13 i i OPEN FILE-blocket ändras till LI R2,28. (Se kap 18).

Snabbhet:

Taxan printern, är utrustad med en 3kb buffert. En screendump, består av över 6kb. Alltså klarar inte bufferten av att sluka hela bytes-svärmen på en gång, men arbetstiden för datorn blir ändock väsentligen förkortad. Jag dumpar nu ut en normal skärm på 7.5 sek datortid, alltså 120 ggr snabbare än tidigare. Printern spottar ut en skärm på 24 sek.

Har Du förstått programmet? Då förstår Du också, att den effektiva dator-kör-tiden har samband med skärminnehållet.

Med ovanstående printer-uppsättning, dumpas en helt tom skärm på 6,5 sek och en helt fylld på 8,5 sek effektiv datortid.

Ökad snabbhet:

Om instruktionerna, CI R1,>0080 och JEQ EMPTY, placeras mellan instruktionerna, SRL och SLA i NEXCHR blocket, samt instruktionen innan GOBUFF blocket, dvs LI R7,OUT, tillföres label'n EMPTY. Så kommer programmet, vid en space-karaktär, att hoppa över bit för bit avkodningen.

Detta medför att programmet blir något snabbare (ca 0,5 sek på en normal skärm). Men det förutsätter också att karaktären med ASCII-värdet 32, verkligen är en tom karaktär. Den kan ju i det anropande programmet, av skäl "Gud vet vad", blivit tilldelad annat än nollor.

Eftersom programmet ej skall kräva några andra förutsättningar, än att det är en x-basic-skärm, så har jag avstått från det nyss beskrivna programtillägget.

Laddningstid:

X-basic laddaren laddar programmet på 14 sek.

Andefattigt!

Programmet printar *inte* ut sprite-karaktärer!

Deras eteriska existens är, i vart fall än så länge, allt för svårfångad för mig.

Vrångbild?

Ett av kraven på programmet var, att utprintningen skulle vara helt skärmliknande. Det blir den också, utom i ett stort väsentligt avseende. Den blir nämligen utdragen på höjden med ca 19%. Detta pga att våra TV-skärmar arbetar med 625 linjer och att datorn är anpassad till 525 linjer, vilket medför att skärmbilden blir hoptryckt till endast 84% av full höjd.

Mao, våra skärmar visar fel proportioner, medan utprintningen visar rätt.

Den vidstående screen-dumpen av sverigekartan visar med all tydlighet vad effekten kan bli. Kartan har rätt proportioner på skärmen, men vid utprintningen blir den näst intill en karikatyr på vårt avlånga land.

Dessa proportionfel är inte mycket att göra åt, utan man måste bestämma sig under programmeringen, om grafiken skall vara korrekt på skärmen eller på papperet.

Men vanliga BASIC-skärmar då?

Det går givetvis alldeles utmärkt att dumpa dessa också. Enda kravet är, att basic programmet går att ladda och köra i x-basic!

Det var allt!

Phu!

Redaktörens kommentar

Jag testade ändringen för seriekopplad printer på min Epson-kompatibla Gemini-printer och det fungerade bra. Dock måste jag lägga till .PA=N på slutet i printerbeskrivningen för att det skulle fungera. Då måste jag förstås också ändra i värdena som beskrivits ovan.

```
*      SCREENDUMP ROUTINE TO BE USED IN EXTENDED BASIC PROGRAMS.       *
*                                                                      *
************************************************************************
*                                                                      *
*      An article about a Mini-Memory Screendump-program               *
*      encouraged and inspired me to write this program.               *
*                                                                      *
*      Thank's Arne Wennberg for the article!                          *
*                                                                      *
*      Thank's Anders Persson for the DSRLNK subroutine!               *
*                                                                      *
*                                                                      *
*      860429                                                          *
*                                                                      *
*      Sïren Bernle                                                    *
*      P1 280                                                          *
*      S-260 38 Kattarp                                                *
*      Sweden                                                          *
*      Tel. 042-93305                                                  *
*                                                                      *
************************************************************************
       DEF  START
*
*  NMLPNT EQU  >8356     IS IN THE DSRLNK ROUTINE!
*  STATUS EQU  >837C              -"-
*  VSBW   EQU  >2020              -"-
*  VMBW   EQU  >2024              -"-
*  VSBR   EQU  >2028              -"-
*  VMBR   EQU  >202C              -"-
*  GPLWS  EQU  >83E0              -"-
*
```

```
PABBUF EQU  >1000
PAB    EQU  >FB0
WRITE  BYTE >03
CLOSE  BYTE >01

*  PRINTER CODES  ***************************************
*                                                        *
INGR   BYTE >1B,>41,>08  8/72 INCH LINE SPACING CODE     *
ENDUP  DATA >1B40        RESET PRINTER CODE              *
       DATA >0D0A        CARRIAGE RETURN AND LINE FEED CODE *
NEWROW DATA >0D0A                -"-                     *
       DATA >1B4B        SINGLE DENSITY GRAPHIC CODE     *
       DATA >0001        BYTES/LINE IN GRAPHIC-MODE      *
*                                                        *
*  FURTHER PRINTER-DEPENDANT-VALUES IN THE PROGRAM,      *
*  IS MARKED WITH '!!!', IN THE COMMENT FIELD!           *
*                                                        *
**********************************************************

BUFF   BSS  >100         PRINTCODE-BUFFER FOR ONE SCREENROW
OUT    DATA 0,0,0,0      PRINTCODE-BUFFER FOR ONE CHARACTER
PDATA  DATA >0002,PABBUF
       DATA >0003        !!!
       DATA >0000
       DATA >0003        !!!
       TEXT 'PIO'        !!!
       EVEN
IN     BSS  8            SCREENCODE-BUFFER FOR ONE CHARACTER
BSCSTO BSS  >104         AREA TO STORE BASIC PROGRAM PARTS IN
MYREG  BSS  >20

START  LWPI MYREG

       LI   R0,PAB       * SAVE PARTS OF BASICPROGRAM STORED
       LI   R1,BSCSTO    * IN THE PAB AND PABBUF AREA
       LI   R2,>104
       BLWP @VMBR

       LI   R0,PAB       * OPEN FILE
       LI   R1,PDATA
       LI   R2,13        !!!
       BLWP @VMBW
       LI   R3,PAB+9
       MOV  R3,@NMLPNT
       BLWP @DSRLNK
       DATA 8

       MOVB @WRITE,R1    * CHANGE I/O OP-CODE TO WRITE
       LI   R0,PAB
       BLWP @VSBW

       LI   R0,PABBUF    * SET PRINTER TO 8/72 INCH LINE SPACING
       LI   R1,INGR
       LI   R2,3         !!!
       BLWP @VMBW
       MOV  R3,@NMLPNT
       BLWP @DSRLNK
       DATA 8

       LI   R0,PAB+5     * CHANGE CHARACTER COUNT TO >83
       LI   R1,>8300
       BLWP @VSBW

       CLR  R4           CHARACTER POINTER
       CLR  R8           BYTE-BEING-DECODED REGISTER

NEXROW LI   R5,BUFF

NEXCHR MOV  R4,R0        * GET CHARACTER (ASCII VALUE+96)
       BLWP @VSBR
       SRL  R1,8         SHIFT FROM MSByte TO LSByte
       SLA  R1,3         TO GET PATTERNADDRESS, MAKE R1 8 TIMES GREATER

       MOV  R1,R0        * MOVE CHARACTER-PATTERN-CODE
       LI   R1,IN        * TO IN-BUFFER
       LI   R2,8
       BLWP @VMBR

       LI   R6,>0100     BIT VALUE REGISTER
NEXBYT DEC  R2           R2, IS THE BYTE-POINTER
       MOVB @IN(R2),R8
       JEQ  BITUP        IF BYTE EMPTY, NO DECODING

       LI   R7,OUT
NEXBIT SLA  R8,1         SHIFTS MSB TO CARRY-BIT
       JOC  BITON        IF CARRY-BIT SET, ADD BIT-VALUE TO BYTE
       JEQ  BITUP        IF BYTE EMPTY, END DECODING
       INC  R7           TO KEEP R7 IN SEQUENCE
       JMP  NEXBIT
BITON  AB   R6,*R7+
       JMP  NEXBIT

BITUP  SLA  R6,1         MAKE BIT-VALUE, 2 TIMES GREATER
       JNE  NEXBYT       IF NOT EQUAL, 8 BYTES HAS NOT BEEN TESTED

       LI   R7,OUT       * MOVE THE 8 DECODED BYTES TO THE ONE ROW-BUFFER
GOBUFF MOV  *R7,*R5+     * AND CLEAR THE OUT-BUFFER
       CLR  *R7+
       CI   R7,OUT+8
       JL   GOBUFF

       INC  R4           POINTS TO THE NEXT CHARACTER
       CI   R5,BUFF+>100 AT THE END OF THE ROW?
       JNE  NEXCHR       NO!

       LI   R5,2         YES!
       LI   R0,PABBUF    * PREPARE FOR ONE ROW PRINT-OUT
       LI   R1,NEWROW->83
       LI   R2,>83

PRIOUT MOV  R3,@NMLPNT   * 2 TIMES PRINT-OUT LOOP
       A    R2,R1
       BLWP @VMBW
       BLWP @DSRLNK
       DATA 8
       DEC  R5
       JNE  PRIOUT

       CI   R4,768       AT THE END OF THE LAST ROW?
       JNE  NEXROW       NO!

       LI   R0,PAB+5     YES! * CHANGE CHARACTER COUNT TO 6
       LI   R1,>0600     !!!
       BLWP @VSBW

       LI   R0,PABBUF    * RESET PRINTER AND LINESPACE 2 TIMES
       LI   R1,ENDUP
       LI   R2,6         !!!
       BLWP @VMBW
       MOV  R3,@NMLPNT
       BLWP @DSRLNK
       DATA 8

       MOVB @CLOSE,R1    * CLOSE FILE
       LI   R0,PAB
       BLWP @VSBW
       MOV  R3,@NMLPNT
       BLWP @DSRLNK
       DATA 8

       LI   R0,PAB       * MOVE BASIC-PROGRAM-PARTS BACK TO VDP-RAM
       LI   R1,BSCSTO
       LI   R2,>104
       BLWP @VMBW .

       CLR  R0           * SO NO ERROR IS REPORTED
       MOVB R0,@STATUS
       LWPI GPLWS        * RETURN TO CALLING PROGRAM
       B    @>0070

**************************
*
* DSRLNK ROUTINE TO BE USED IN ASSEMBLY PROGRAMS CALLED BY EXTENDED BASIC.
* APPEARS EXACTLY LIKE THE ED/AS DSRLNK TO THE CALLING PROGRAM.
* BLWP @DSRLNK
* DATA >8          (FOR NORMAL DSR CALL)
* IN CASE OF AN ERROR, THE EQUAL BIT IS SET UPON RETURN.
*
* NOTE: THIS ROUTINE AUTOMATICALLY CHECKS IF THE FILENAME IS CS1 OR CS2, WHICH
* MEANS A GPL DSR, OR ANY OTHER, WHICH MEANS ML DSR.
* THIS IS AN IMPROVEMENT OVER THE ED/AS DSRLNK, AND RELIEVES THE CALLING PROG-
* RAM FROM THAT TASK.
*
* THE NAME LENGTH POINTER MUST BE STORED @>8356 BY THE CALLING PROGRAM.
* ALSO, IN CASE OF AN ERROR THAT IS SUPPOSED TO BE REPORTED BY THE ERR ROUTINE,
* THE CALLING PROGRAM MUST STORE THE PAB POINTER @>831C.
*
* A-DATA 841011
*
**************************

PAD    EQU  >8300
NMLPNT EQU  >8356
DSRMOD EQU  >836D
SP     EQU  >8373
STATUS EQU  >837C
GPLWS  EQU  >83E0
GPL11  EQU  >83F6

GRMWA  EQU  >9C02
GRMRA  EQU  >9802
GRMRD  EQU  >9800
*                        DSRLNK HAS NO DEF, SINCE IT IS SUPPOSED TO BE CALLED
VSBW   EQU  >2020
VMBW   EQU  >2024
VSBR   EQU  >2028        BY AN ASSEMBLY PROGRAM ONLY.
VMBR   EQU  >202C

DSRLNK DATA DSRWS,DSRPGM
SAVGRM DATA 0            TO SAVE GROM ADDRESS IN
DSRWS  BSS  >20
HIGROM BYTE >00
LOGROM BYTE >10
EQUMSK DATA >2000        POSITION OF EQUAL BIT IN ST AND COND BIT IN GPLST
IOERMS DATA >E000        MASK FOR I/O ERROR CODE IN PAB STATUS BYTE

*********************

DSRPGM MOVB @GRMRA,@SAVGRM    SAVE GROM ADDRESS
       MOV  *R14+,R0          GET DSR MODE
       MOVB @GRMRA,@SAVGRM+1
       DEC  @SAVGRM           CORRECT GROM ADDRESS
       MOVB @HIGROM,@GRMWA    SET UP NEW GROM ADDRESS
       MOV  @GPL11,R10      SAVE GPL RETURN ADDRESS
       MOVB @LOGROM,@GRMWA

       SWPB R0           MAKE DSR MODE A BYTE
       MOVB R0,@DSRMOD
       MOV  @NMLPNT,R3
       MOVB @GRMRD,R0    CALCULATE GROM ADDRESS OF ROUTINE
       ANDI R0,>1F00
       SWPB R0
       MOVB @GRMRD,R0
       SWPB R0
       INCT R0

       MOVB @SP,R2       PREPARE PUSH ON GPL RETURN STACK
       SRL  R2,8
       AI   R2,PAD
       INCT R2

       LI   R1,>AA0C     PUSH >AA0C ON RETURN STACK
       MOV  R1,*R2
       SWPB R2
       MOVB R2,@SP       WRITE STACK POINTER BACK

       MOVB R0,@GRMWA    SET UP NEW GROM ADDRESS
       SWPB R0
       MOVB R0,@GRMWA

       MOV  @>3FE4,R2    >3FE4 HAPPENS TO BE A GOOD ADDRESS...
       LI   R1,ENDIO
       MOV  R1,@>3FE4
       LWPI GPLWS
       LI   R11,>0070
       B    *R11         BRANCH TO GPL INTERPRETER

ENDIO  LWPI DSRWS
       MOV  R2,@>3FE4    RESTORE @>3FE4
       AI   R3,-8
       MOV  R3,R0
       BLWP @VSBR        READ PAB STATUS BYTE
       SZC  @EQUMSK,R15  ASSUME NO ERRORS

       CZC  @IOERMS,R1   CHECK ERROR CODE IN PAB STATUS
       JNE  ERROR        IF NOT ZERO THEN THERE IS AN ERROR

       MOVB @STATUS,R2   ZERO. MUST CHECK STATUS BYTE
       CZC  @EQUMSK,R2
       JEQ  NOERR        IF ZERO THEN NOERROR

ERROR  SRL  R1,5         STORE ERROR CODE IN MSB OF R0 (CALLING WS)
       CLR  *R13
       MOVB R1,*R13
       SOC  @EQUMSK,R15  SET EQUAL BIT IN ST

NOERR  MOVB @SAVGRM,@GRMWA    RESTORE GROM ADDRESS
       MOV  R10,@GPL11        RESTORE GPL RETURN ADDRESS
       MOVB @SAVGRM+1,@GRMWA
       RTWP
       END
```

Samlingsdisk för Programbiten -85

av Jan Alexandersson

För att göra det litet lättare för dig som medlem att ta del av alla bra program i tidningen från föregående år har jag samlat ihop det bästa på en fullproppad flexskiva. Det som inte finns med är i första hand LT-Writer och alla FORTH-program tex FORIT.

Du kan beställa LT-Writer hos Lennart Thelander. Det saknas också två bra assemblerprogram från Hans Wickström och Sören Bernle. Som en extra bonus har vi tagit med det kompletta CSAVE från Anders Person som inte har publicerats i sin helhet tidigare. Priset är 60:−. Innehåll:

Namn	*Sekt*	*Typ*
CATALOG	3	PROGRAM
CHARFILTER	3	DIS/VAR163
CSAVE1O	10	DIS/FIX 80 (startas med CSAVE)
CSAVE1S	34	DIS/VAR 80
DART	19	PROGRAM
INDEXSORT	3	PROGRAM
JOYST	2	DIS/VAR163
LANDER	12	PROGRAM
LIST28	7	PROGRAM
LISTAORD	5	PROGRAM
LTWRITMINI	18	PROGRAM
MINIASSEM	23	PROGRAM
MINICHIME	33	PROGRAM
MULTISAY	3	PROGRAM
NUC	24	PROGRAM
OHMSLAG	16	PROGRAM
ORMAR	13	PROGRAM
PEEKV/XB	5	PROGRAM
PROGLOAD	9	PROGRAM
RITAKURVA	8	PROGRAM
RITAKURVAK	6	PROGRAM
SNABBFIL	18	PROGRAM
STRSORT	3	PROGRAM
STRSORTXB	3	PROGRAM
SVENSKABA	2	DIS/VAR163
SVENSKAXB	2	DIS/VAR163
TAPETESTO	3	DIS/FIX 80 (startas med MAG)
TAPETESTS	8	DIS/VAR 80
TIPSRAD	11	PROGRAM
TOTO-G	15	PROGRAM
TRAIN	8	PROGRAM
TRIPPELPKT	24	PROGRAM
WIND	4	PROGRAM

Jag har även sammanställt en 20 min kassett med nästan alla program som finns på flexskivan. CSAVE och TAPETEST har lagrats för Mini Memory med kassett. Priset är 50:−. Innehåll:

BASIC
Ohms lag
Ormar
Rita Kurva
Strängsortering

Personal Record Keeping-BASIC
Snabbfil

Extended BASIC
Dart
Lander
LT-miniwriter
NUC-spelet
Tipsrad
Toto-G
Train
Trippelpunkten
Wind-data
Strängsortering-XB
Indexsortering

Extended BASIC+expansionsminne 32K
Mini-Chime (inkl. Mini-Assem)
PEEKV/XB

Extended BASIC + Speech synthesizer
Multi-Say

Mini Memory (Easy Bug L)
CSAVE + TAPETEST
(Startas från Mini Memory 3 RUN med CSAVE respektive MAG)

Bygg din egen RAM-disk!

av Peter Odelryd

En liten amerikansk firma, *Horizon Computer Ltd*, har tagit fram en RAM-disk för 99-an. Det är ett kort för expansionsboxen. Till skillnad från liknande produkter, tex *Myarcs* 128 och 512 K-kort, kräver detta kort att man redan har minst 32K minne i boxen. Flexskiveminne är också nödvändigt, eftersom den tillhörande programvaran ligger på diskett.

Vad är nu en RAM-disk och hur fungerar den? Det är helt enkelt ett kort som fungerar som en "elektronisk flexskiva". Man kan lagra och hämta filer på den precis som på en flexskiva. På RAM-disken går det 5−10 gånger snabbare. Tänkbara tillämpningar är såna där man ofta hämtar in filer på diskett, tex vid arbete med kompilerande språk som c99. Kommunicerar man via modem kan det också spara pengar om man kan lagra ner text snabbare än på diskett.

Man kan adressera RAM-disken som DSK1 tom DSK6. Det innebär att man kan ha tre flexskiveenheter och dessutom upp till tre RAM-diskar inkopplade samtidigt!

Horizon säljer sin produkt i olika versioner. Det finns färdigbyggda kort motsvarande både enkel- eller dubbelsidiga flexskiveenheter, bestyckade med 104 resp 192K RAM. De kostar 165 resp 210 dollar. Många tycker nog att den byggsats som de säljer är intressantare. Kortet säljs med programvara och byggbeskrivning för 53 dollar. Då får man själv skaffa de komponenter som behövs till kortet. Det gäller främst minneskretsarna. Det går åt 13 resp 24 st Hitachi HM 6264 LP-15 statiska CMOS-RAM för enkel- eller dubbelsidig RAM-disk. Det finns firmor i Sverige som säljer kretsarna för ca 30:−/st + moms. Priserna gäller då för 25 st. Köper man 100 eller fler blir det billigare. Dessutom behövs några IC-kretsar och dioder, motstånd mm. Kortet är avsett av byggas med batteribackup, varför batterier och hållare också tillkommer. Batterierna laddas upp när boxen är på och skall aldrig behöva bytas. Komponentsatser kan köpas från en amerikansk firma för 72 resp 105 dollar. På alla ovan nämnda priser tillkommer frakt.

Den här artikeln bygger på material från Horizon Computer och en recension i tidningen MICROpendium. Recensionen avsåg det färdigbyggda kortet. Det fick genomgående bra kritik, möjligen med reservation för priset. Bygger man det själv blir det förstås billigare. Som med alla elektronikbyggsatser krävs det förstås en viss vana för att lyckas.

Avsikten med den här artikeln är främst att kolla intresset för att bygga det här kortet. Finns det ett sådant kan vi gå vidare och kolla var man kan köpa komponenterna billigast. Det enklaste vore kanske att få ett paketpris på alla komponenter från någon elektronikfirma. Räkna med att lägga ut minst 500 kronor för inköp av det nakna kortet, frakt och tull mm. Om man köper 5 kort samtidigt kan man få dem till ett bättre pris.

Eftersom det här inbegriper en del hanterande av pengar kommer inte föreningen Programbiten att stå för det. Är Du intresserad så hör av dig inom 2 veckor efter tidningens utgivning. Skicka ett brev innehållande ett frankerat kuvert med din adress. Intresseanmälan betyder inte i det här läget att Du binder dig för något. Skicka brevet till min hemadress (står i slutet av Redaktörs-spalten).

Styr pratorn med CALL LOAD

Av Stephen Shaw

Artikeln ursprungligen publicerad i den engelska användartidningen *TI*MES* nummer 6 (hösten 1984). Översatt och bearbetad av Peter Odelryd.

För att kunna använda det här programmet behöver du Speech synthesizern och antingen Mini Memory, Editor/Assembler eller Extended BASIC *och* 32K minnesexpansion.

```
100 CALL INIT
110 S=-27648
120 FOR I=1 TO 1000
130 NEXT I
140 PRINT "START ..."
150 FOR X=1 TO 20
160 REM TEST RUTIN IN HÄR
170 FOR T=1 TO 30
180 PRINT ">";
190 NEXT T
200 NEXT X
210 PRINT "STOPP ..."
```

Det här programmet ger oss en bakgrund för de tester vi ska göra.

Att köra ovanstående program, med slingan på rad 170 upprepad 30 gånger (som i listningen) tar 18,6 sekunder från "START" till "STOPP". Ändra rad 170 så att slingan bara genomlöps 20 gånger och exekveringstiden blir 12,7 sekunder.

Nu kan vi lägga in det vi vill testa. Det första går bara ifrån Extended BASIC: Ersätt rad 160 med nedanstående:

```
160 CALL SAY("#THAT IS INCORRECT#")
```

Kör programmet igen. Om rad 170 genomlöps 20 gånger blir tiden 44 sekunder. 30 gånger ger tiden 50 sekunder. Tiden för talet är konstant, det lägger till ca 21 sekunder till tiden.

Nu ska vi pröva nästa alternativ, som också fungerar med Mini Memory! Ersätt rad 160 med denna rad:

```
160  CALL LOAD(S, 70, "", S, 65, "", S, 72, "", S, 70,
     "", S, 64, "", S, 80)
```

Om du nu kör programmet så säger det samma sak lika många gånger, men tiden blir en helt annan!

Med rad 140 genomlöpt 20 gånger: 26,3 sek, genomlöpt 30 gånger: 26,5 sek.

Vi vet att det tar 6 sek extra att genomlöpa slingan i rad 170 10 extra gånger. Men var har den tiden tagit vägen?

CALL SAY-rutinen stoppar allt annat tills den "pratat färdigt". Men om CALL LOAD används istället fortsätter datorn direkt med nästa rad utan uppehåll.

Det går alltså fortare att använda CALL LOAD, särskilt om programmet ger datorn annat att göra medan den pratar.

Det var demonstrationen. Nu lite teori.

Referens: Editor/Assembler-manualen sidorna 351, 355, 422 till 427. (Obs tryckfelet i första stycket på sid 355, borde vara sektion 22.1.4, inte som det står i manualen.

Adressen –27648 är SPEECH WRITE-adressen. Vi skickar dit värden och så småningom börjar pratorn tala.

Värdena som ska skrivas på adressen kan man räkna ut på följande sätt:

Bestäm först vad du vill att pratorn ska säga ur standardvokabulären. Titta sen i tabellen (sid 422–427 i E/A-manualen). Vårt exempel ovan "THAT IS INCORRECT" återfinns på adressen 6816. Detta är ett hexadecimalt värde, inte ett decimalt!

De fyra siffrorna är sedan omkastade och blir 6186. Sen måste vi ge värdena ett offset på hex 40. Eftersom vi anger värdena i CALL LOAD decimalt lägger vi till 64 till varje siffra.

(64+6)	(64+1)	(64+8)	(64+6)
70	65	72	70

(Om siffrorna här vore de hexidecimala bokstäverna A–F fick bokstäverna följande värden: A=10, B=11, C=12, D=13, E=14, F=15).

Vi måste också markera slutet på ordet genom att skriva en nolla, med ett offset på 64 blir det 64. Slutligen måste vi instruera pratorn att prata, genom att avsluta med hex 50, decimalt 80.

Att göra alla ovanstående beräkningar manuellt blir ju tidsödande i längden. Att göra upprepade beräkningar är ju som bekant en uppgift som datorer klarar av. Skriv ett BASIC-program som hanterar uppgiften!

Notera hur CALL LOAD använts i programmet, *ett* kommando för att skriva flera olika värden till samma adress.

Om Du inte har Editor/Assembler-manualen till hands så kommer nedan några ord att testa.

56B3 Ready to start	7DDB You win	17A5 Again
1913 Answer	1D82 Check	1DA2 Choice
1F1A Command	27B6 Else	3148 Goodbye
3571 Help	3757 Hurry	39BD Instructions
37CF I win	3AED Joystick	47CD Name
49A5 Nice try	5093 Please	2034 Computer
700F Try again	52AA Printer	68FE That is right
6696 Texas Instruments	77E9 What was that	

Rättelse PB 84-4 sid 15

Det första *Prator*-programmet skall ändras på rad 500:

```
500 CALL LOAD (SPWR, LOAD+BH-16*N)
```

Om Du även stoppar in printsats på raderna med CALL LOAD i detta program så slipper du slå i tabellen enligt ovan. Alla värden kommer i den ordning de skall vara vid CALL LOAD och med rätt offset.

Annonser

Säljes:
Multiplan 395:–
Speech synthesizer 395:–
Manual till Editor/Assembler 100:–
(eventuellt porto tillkommer)

Jan Alexandersson
Springarvägen 5, 3 TR
142 00 TRÅNGSUND
Tel: 08/7710569

Säljes:
TI-99/4A med Extended BASIC-modul och manual.
Spelmoduler: Adventure, Munchman och Miner 2049er.
Pris: 1300:–

Jan Tengelin
Thermiavägen 4
671 00 Arvika
Tel: 0570/16187

Säljes:
Programmer's Reference Guide to the 99/4A (bok från Compute)
Compute's Beginner's Guide to Assembly Language on the TI-99/4A (bok från Compute)
Spy's Demise (spel för Mini Memory)

Mikael Jonsson
Örnvägen 56
222 31 LUND
Tel: 046/119700

Bug i DM 1000

Jan Alexandersson har upptäckt ett fel i Disk Manager 1000, ett freewareprogram som utför allt det som TI:s Disk Manager 2 gör (och mycket mer). Om man kopierar från en diskett till en annan och redan har en fil med samma namn på disketten man kopierar till kan man få problem. Om filen är större än 40 sektorer kopieras bara de första 40 rätt, resten fylls med skräp från sektor 0 och följande. En fullständig beskrivning av DM 1000 kommer i nästa nummer.

Parameterpassning mellan Extended BASIC-program

Följande artikel är hämtad ur *TI*MES*, den engelska användarklubbens för TI99 tidning, och är skriven av Mike Kabala. Översättning och viss bearbetning av Börje Häll.

En av de mer avancerade möjligheterna för Dig som programmerar i Extended Basic är möjligheten att länka ihop program med RUN-kommandot. Det medger möjligheten att skriva mjukvara som är så stor att den inte får plats i minnet på en gång. När Du är färdig med ett programsegment skriver Du bara RUN "DSK1.xxx" för att läsa in nästa segment.

Tyvärr finns det ett stort "fel". RUN nollställer alla variabler som finns i minnet. Följande program visar detta. Spar det första programmet som DEMO1 och det andra som DEMO1A.

```
100 ! DEMO1
110 CALL CLEAR
120 DISPLAY AT(1,1):"Knappa in någonting"
130 !
140 ! ACCEPT AT behövs för att få texten
150 ! på ett bestämt ställe
160 !
170 ACCEPT AT(2,1):A$
180 !
190 ! Alla variabler nollställs när nästa
200 ! kommando utförs
210 !
220 RUN "DSK1.DEMO1A"
230 END
100 ! DEMO1A
110 CALL CLEAR
120 !
130 ! Alla variabler har nollställts
140 ! av RUN-kommandot i föregående
150 ! program
160 !
170 PRINT A$
```

Någonstans mellan DEMO1 och DEMO1A har värdet i A$ försvunnit. Nu går det att komma runt detta problem på flera olika sätt. Du kan, om Du vill vill, skapa en file och lagra A$ i den och sedan läsa in A$ från filen i programmet DEMO1A. Det verkar lite onödigt att skapa en file bara för att skicka en eller två variabler mellan ett par program. I den här artikeln vill Mike visa Dig två andra metoder som kringgår problemet.

I den första metoden används skärmen som tillfälligt minne. Skriv bara det Du vill skicka mellan programmen innan Du utför RUN. Det inlästa programmet kan sedan läsa den texten från skärmen. Följande två program visar detta.

Läs in DEMO1 och ändra det så att det i stället läser in DEMO2A. Spar det sedan som DEMO2 och det andra som DEMO2A. Slutligen, läser Du in och kör DEMO2 och det Du knappar in bör skrivas ut rätt av DEMO2A.

```
100 ! DEMO2
110 CALL CLEAR
120 DISPLAY AT(1,1):"Knappa in någonting"
130 !
140 ! ACCEPT AT behövs för att få texten
150 ! på ett bestämt ställe på skärmen
160 !
170 ACCEPT AT(2,1):A$
180 !
190 ! Alla variabler nollställs när nästa
200 ! kommando utförs
210 !
220 RUN "DSK1.DEMO2A"
230 END
100 ! DEMO2A
110 !
120 ! Läs vad föregående program
130 ! skrev på skärmen
140 ! Offset av 2 behövs för skillnad
150 ! mellan ACCEPT och GCHAR
160 !
170 A$=" "
180 FOR I=3 TO 30
190 CALL GCHAR(2,I,A)
200 A$=A$&CHR$(A)
210 NEXT I
220 !
230 ! Värdet har skickats mellan
240 ! programmen med skärmen
250 ! som tillfälligt minne
260 !
270 CALL CLEAR
280 DISPLAY AT(5,1):"Du skrev: "
290 DISPLAY AT(6,1):A$
300 END
```

För den andra metoden måste Du ha expansionsminne. Knepet här är att använda expansionsminnet som tillfälligt minne för variabelvärdet. Fördelen med det är att Du inte behöver skriva något på skärmen för att skicka värden mellan programmen. Var noga med att inte använda lägre adress än 9984 och inte högre än 16184. (Mer om dessa adresser senare. *Övers anm.*).

Du måste också komma ihåg att göra CALL INIT innan Du skriver något i expansionsminnet. Knappa in de följande programmen och spar dem som DEMO3 respektive DEMO3A för en demonstration.

```
100 ! DEMO3
110 CALL CLEAR
120 PRINT "Knappa in någonting"
130 ACCEPT A$
140 !
150 ! Initialisera
160 ! expansionsminnet
170 !
180 CALL INIT
190 !
200 ! Spar strängens längd
210 !
220 A=LEN(A$)
230 CALL LOAD(9984,A)
240 !
250 ! Spar strängen
260 !
270 FOR I=1 TO A
280 CALL LOAD(I+9984,ASC(SEG$(A$,I,1)))
290 NEXT I
300 CALL CLEAR
310 RUN "DSK1.DEMO3A"
320 END
100 ! DEMO3A
110 CALL CLEAR
120 PRINT "Du knappade in: "
130 !
140 ! Hämta längden på strängen
150 !
160 CALL PEEK(9984,L)
170 !
180 ! Hämta strängen
190 !
200 X$=" "
210 FOR I=1 TO L
220 CALL PEEK(I+9984,X)
230 X$=X$&CHR$(X)
240 NEXT I
250 !
260 ! Värdet har skickats mellan
270 ! programmen med expansionsminnet
280 ! som tillfälligt minne
290 !
300 PRINT X$
310 END
```

Du kanske har märkt att Mike bara har använt strängvariabler i exemplen. Det beror på att det är lättare att hantera det här problemet med strängar i stället för numeriska variabler. Det betyder inte att Du inte kan skicka numeriska värden mellan programmen. Det är bara att använda STR$() för att konvertera numeriska variabler till strängar i det första programmet och sedan i det andra använda

VAL() för att konvertera tillbaka.

Slutligen, om Du inte har en diskdrive och vill försöka med den första metoden så knappa in följande program och spar det till CS1. Knappa sedan in DEMO2A och spar det på samma kassett *utan* att spola tillbaka bandet. Spola sedan tillbaka bandet och läs in det första programmet och kör det.

Efter det att Du har knappat in något och tryckt på ENTER så blir Du uppmanad att spola tillbaka bandet. *Gör inte* det utan tryck på ENTER och följ sedan anvisningarna som skrivs på skärmen.

Anledningen till att raderna 120 och 170 måste ändras är att datorn skriver ut 14 rader med instruktioner från kassettrutinen. Om inte ändringen görs kommer det andra programmet att läsa från fel ställe på skärmen.

```
100 ! DEMO1 för kassett
110 CALL CLEAR
120 DISPLAY AT(15,1): "Knappa in någonting"
130 !
140 ! ACCEPT AT behövs för att få texten
150 ! på ett bestämt ställe
160 !
170 ACCEPT AT(16,1):A$
180 !
190 ! Alla variabler nollställs när nästa
200 ! kommando utförs
210 !
220 RUN "CS1"
230 END
```

Därmed är det slut på Mike Kabalas artikel från *TI*MES*.

Nu något om adresserna vid användning av minnesexpansionen. När jag provade Mikes andra metod började jag fundera över vilka adresser i minnesexpansionen som skulle kunna användas för att skicka värden mellan olika program. Jag kom då fram till följande. Om inga assemblerrutiner används så kan adresserna från och med 8200 till och med 16383 användas. Om assemblerrutiner används har man utrymmet från 9460 och upp till början av REF/DEF-tabellen. Då frågar man sig: var börjar REF/DEF-tabellen?

En annan fråga är om man kan använda någon del av höga minnet i expansionsminnet. Jag letade då i olika tidningar och i *The Smart Programmer* March 1984 från Millers Graphics hittade jag en del intressanta saker. På adressen 8194 får man reda på den första lediga adressen i låga minnet. Adress 8196 ger den högsta lediga adressen+1. Motsvarande adresser för höga minnet är –31860 som ger lägsta fria adressen och –31866 som ger högsta fria adressen+1. X-Basic kan inte läsa in assemblerprogram i höga minnet men där ligger istället X-Basic-programmet. Med den här kunskapen kan man göra Mikes metod två generell enligt följande programrader för låga minnet

```
100 CALL INIT
110 !
120 ! Ta reda på lägsta fria adressen
130 ! i låga minnet
140 !
150 CALL PEEK(8194,A,B)
160 LLFRI=256*A+B
170 !
180 ! Ta reda på högsta fria adressen
190 ! i låga minnet
200 !
210 CALL PEEK(8196,A,B)
220 LHFRI=256*A+B
230 ! Resten av programmet
```

och följande för det höga minnet i expansionsminnet

```
100 CALL INIT
110 !
120 ! Ta reda på lägsta fria adressen
130 ! i låga minnet
140 !
150 CALL PEEK(8194,A,B)
160 HLFRI=256*A+B-65536
170 !
180 ! Ta reda på högsta fria adressen
190 ! i låga minnet
200 !
210 CALL PEEK(8196,A,B)
220 HHFRI=256*A+B-65536
230 ! Resten av programmet
```

Observera att den här metoden att ta reda på ledigt utrymme i låga respektive höga minnet går bara att använda om eventuellt laddad assemblerkod är relokerbar. Är den det, så uppdateras adresserna för att peka på de fria adresser som gäller efter laddningen av assemblerkoden. Är assemblerkoden inte relokerbar måste man veta var den läses in i minnet och därigenom veta vilket ledigt utrymme som finns för överföring av variabelvärden från ett program till ett annat.

Det går givetvis att spinna vidare på det här och ta reda på om det finns tillräckligt med plats för att lagra variabelvärden, om det går att göra med Editor/Assembler eller Mini Memory och Basic osv.

Metod 1 via skärmen bör gå bra från Basic om Du tar hänsyn till att skärmen skrollas när ett program slutar och när Du knappar in OLD DSK1.xxx och RUN.

Frågor och svar

Av Börje Häll

Eftersom det kan ta lite tid från det vi har fått frågorna tills tidningen kommer ut, så kan Ni som sänder in frågorna sända med ett frankerat och adresserat kuvert så får Ni svaren på posten.

Från *Anders Månsson* i Malmö har vi fått följande frågor.

Fråga: Vad håller datorn på med från det man knappat i RUN och tills själva programmet börjar?

Datorn reserverar utrymme och lägger upp symboltabeller för variabler, arrays och data. Detta kan Du läsa mer om i det 16-sidiga komplementet som följde med X-Basicmanualen.

Fråga: Om jag sätter 5 sprites på en "dotrow" varför blir då den femte osynlig?

Det beror på att videoprocessorn (9918A) bara har 4 spritesregister. De registren används bla för att hålla reda på platsen för den pixelrad som just ritas. (Från *The Smart Programmer* september 1984).

*Fråga:*Kommandot CALL COINC(#1,#2,16,C) kollar om sprite 1 och sprite 2 befinner sig mindre än 16 positioner från varandra. Om man vill ha reda på vilka godtyckliga sprites som ligger inom ett visst antal positioner från varandra måste man ha en loop, vilket tar tid i X-Basic. Finns det någon adress i minnet som man med CALL PEEK kan läsa i stället för att ha en loop?

Beträffande COINC så finns det inget register som är tillgängligt från X-Basic. Det är möjligt att man kan skriva assemblerrutiner som kan titta i Sprite Attribute List och därigenom ta reda på positionerna för spritsen. Att 2 sprites överlappar varandra kan man utläsa i VDP status byte på adress >837B (–31877), bit 2 satt, dock inte vilka sprites det gäller.

Fråga: Finns det någon minnesadress på vilken jag med CALL PEEK kan avgöra om en sprite överlappar fast grafik?

Nej, men det kanske kan lösas på liknande sätt som anges i föregående svar.

Fråga: Vilken kringutrustning behövs för programmering i FORTH och Pascal?

PB-FORTH kräver minnesexpansion, X-Basic eller Mini Memory eller Editor/Assembler, kassettspelare och programkassett med FORTH.

TI-FORTH kräver minnesexpansion, Editor-Assembler, diskdrive och programdiskett med FORTH.

Pascal kräver minnesexpansion, P-kod-kort, diskdrive och kompilator, editor, filhanteringssystem på diskett (Från *99-an* nr 2 1983).