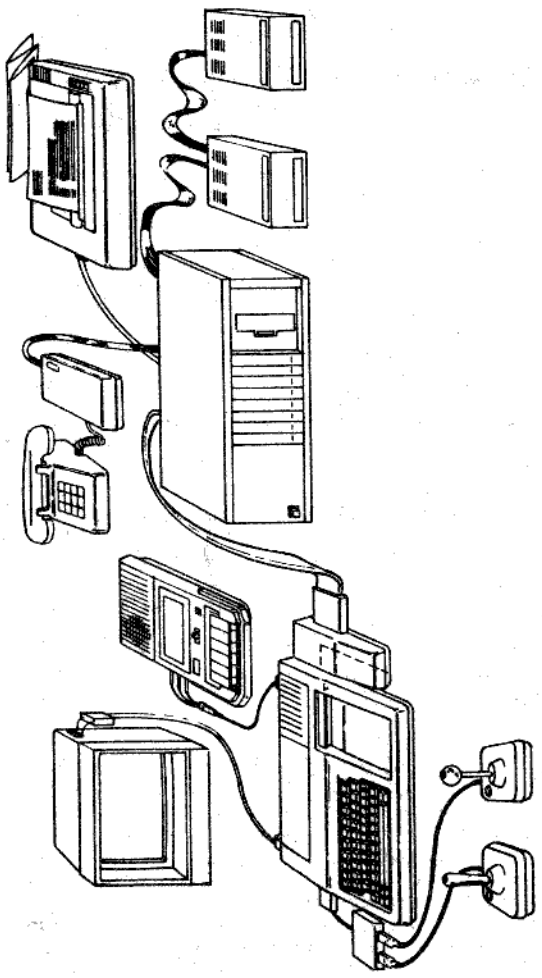
PROGRAM nittinian BITEN 85-4

FORTH spalten

REDAKTÖR: Lars-Erik Svahn



Innehåll

ISSN 0281-1146

Redaktören. . .

Vi har i detta nummer fått en bättre blandning mellan olika typer av program. Vi behöver dock ytterligare program och artiklar från dig som medlem. För att underlätta redaktörens arbete skall du skriva med 55 tecken per rad. Vi trycker det sedan med en printer som ger 12 tecken per tum. I vissa speciella fall har vi använt en printer med 10 tecken per tum, som då ställs in för 46 tecken per rad.

Den vanligaste printern vi har ger vissa konstiga tecken som inte motsvarar svensk standard. Detta gäller följande ASCII-koder:

```
33 = "  utropstecken
35 = ñ  nummertecken
64 = ¦  alfaslang eller snabel-A
94 = ^  tak eller upphöjt till
```

Om du upptäcker fel i programmen så vill vi att du talar om detta.

Ett program bör vara bra strukturerat så att andra kan begripa det och ändra i det. Skriv om möjligt programmet i vanlig BASIC. Extended BASIC bör endast användas när du utnyttjar de speciella finesserna där som t.ex. sprites, DISPLAY AT/ACCEPT AT och LINPUT.

MYARC 256K TI-DATOR

Från Peter Odelryd kommer följande intressanta nyhet, att MYARC kommer att lansera en TI-kompatibel dator i början av november. Den kommer att ha 256k RAM + 64k video-RAM med möjlighet att adressera 2000 kbytes. Datorn har tre grafikmoder 512x212, 512x424 och 256x192 och video-utgång eller RGB-utgång. Tangentbordet har 84 tengenter och kan anslutas till expansionsboxen.

Utöver detta kommer en ny Extended BASIC II för TI-99/4A från MYARC som skall användas tillsammans med 128k minne från MYARC. Den skall vara tre gånger snabbare än TI-XB och har 40 tecken/rad, fönster och många grafikkommandon.

Jan Alexandersson
Springarvägen 5, 3tr
142 00 TRÄNGSUND

Tel. 08-771 05 69

Reservera LÖRDAGen den
8 FEBRUARI 1986
för

Årsmöte

PLATS meddelas senare

I REDAKTIONEN

Redaktör	Jan Alexandersson
Utmaningsredaktör	Anders Persson
Forth-redaktör	Lars-Erik Swahn
Programförmedlare	Göran Nygren
Allt-i-allo	Claes S
Nya rubriker	Börje Häll

Detta nummer av PB hand-assemblerat av Claes + Jan

Föreningens och redaktionens adress:

Föreningen Programbiten
c/o Schibler
Wahlbergsgatan 6 1 tr ned
121 46 Johanneshov

Postgiro 19 83 00 - 6

```
++++++++++++++++++++++++++
+  DATAINSPEKTIONENs   +
+     LICENSNUMMER     +
+                      +
+       82100488       +
++++++++++++++++++++++++++
```

Medlemsavgift för 1985 är .....120 SEK
Nittinian, årgång 1983 (nr 1, 2, 3, 4/5) kostar 80 SEK
Programbiten - Nittinian 1984, 1 - 5 -"- 100 SEK

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

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

FÖRENINGENs TILLBEHÖRs-FÖRSÄLJNING

Följande finns att köpa för medlemmar genom att motsvarande belopp sätts in på postgiro 19 83 00 - 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 + TIFManual SAMTIDIG beställning (600 för ICKE medlem)!!			400:-
TI FORTH	Diskett		60:-
TI FORTH	Manual		190:-
PB FORTH Källkod	(3 disketter)	(*)	120:-
TI FORTH Källkod	(3 disketter)	(**)	120:-
GAME FORTH	Diskett		60:-
FORTH&99	(4 disketter)	(*)	160:-

(*) Säljes endast till den som tidigare köpt PB FORTH
(**) -"- -"- " " " -"- " TI FORTH

MULTIPLAN/TI WRITER	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, nr 1-5,7-9/83 (per styck)		20:-

Nittinian, årgång 1983		80:-
Programbiten - Nittinian årgång 1984		100:-
Programbiten, årgång 1983	Kalkylatorer	60:-
Programbiten, årgång 1982	-"-	60:-
Programbiten, årgång 1981	-"-	60:-
Programbiten, årgång 1980	-"-	60:-
Programbiten, årgång 1978/79	-"-	60:-
Programbiten, fem årgångar 1978-1983		200:-
Programbiten, sex årgångar 1978-1984		280:-
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 sidan 30) per block		11:-
Patenthandlingar TI-59		25:-
40 st tomma magnetkort med plånbok		150:-
Tom magnetkortsplånbok		10:-

ORDFÖRANDEN har ORDET

När jag skriver det här har det just blivit november. Vi börjar få föraningar om vintern. En passande årstid att joxa med datorn, tycker jag. Det har blivit en hel del tid i datorrummet nu - framförallt med försök till FORTH-programmering. Det är verkligen roligt - men borde jag inte få fram godtagbara resultat fortare? Jag kanske inte är en så klurig programmerare som jag skulle vilja vara"

Ett skäl till att jag bör fråga mig det är bl a att jag skrev en artikel om lagring på disketter och en annan om lagring på kassett - till största delen översättningar av förlagor på engelska från olika ställen. Det såg litet konstigt ut när det var färdigt, så jag skickade materialet till Anders Persson i Lund - han med Utmaningen - för att få hans omdöme.

Jodå" Det var inte många siffror rätt, berättade Anders på sin sköna skånska. Artiklarna kommer nu, hoppas jag - men skrivna av Anders. Det var ju tur att jag i alla fall förstod att jag borde låta någon annan titta på dem.

Och då slog det mig att det är ju det där vi har föreningen till" Att byta erfarenheter och dela med oss av våra kunskaper och kontakter. Anders studerar datorkunskap på universitetet och lägger ned mycket tid på sin förkovran inte bara på 99-an. Därför har han intressanta kunskaper att förmedla. Jag å andra sidan bidrog med ett försök till svar på en fråga, där Anders var bättre på att svara än jag. Men hade jag inte frågat hade inget svar blivit artikulerat.

Sensmoral: Frågorna är minst lika viktiga som svaren"" FRÅGA""
Vi hade en medlemsträff i Stockholm i september. Bra, tyckte många. Det var inget märkvärdigt - vi träffades och talade med varandra och tittade på litet olika program, som vi körde upp på en stor video-anläggning i en samlingssal. Vi drack kaffe och delade oss i intressegrupper.

Sensmoral: Det behövs inga störtfiffiga programideer för att göra trevliga träffar. Det räcker med lokal och kallelse, faktiskt"

Och det är väl precis vad som skall hända i Staffanstorp den 7 december. Där har några medlemmar hyrt lokal av kommunen för en billig penning - som föreningen bidrar till - och tänker träffas och prata dator. Det blir den 7/12.

I Stockholm har vi en ny träff den 14/12. På Militärhögskolan som vanligt. Ungefär samma program som i september.

Kanske har de här träffarna varit när Du får tidningen. I så fall bör Du planera för nästa träff igen - nämligen årsmötet lördagen den 8 februari. Då är det dags att välja ny styrelse bl a. Som det ser ut nu avgår kassören, Gösta Blume, och jag. Vi tycker båda att det kan vara lämpligt att yngre krafter tar vid. Se till att Du väljer den styrelse Du vill ha" Sammankallande i valberedningen är Anders Törnkvist i Södertälje. Han har redan en del namn. Prata med honom" Det är inte likgiltigt vilka som leder föreningen - tvärtom"

Vi har sett till att föreningen finns med i en förteckning över 99 User Groups, som publicerats i en amerikansk populär 99-tidning, Millers Graphics. Det har lett till en del kontakter. Bl a skriver en kanadensare, som tydligen är militär, stationerad i Tyskland, att han vill byta program med oss. Hans hemmaförening har 3300 program på disketter" Så se till att vi får något i programbanken att byta med" Jag menar allvar: gör färdigt Dina programideer och deponera dem i programbanken. Men det skall vara Dina EGNA program som DU gjort. Det där som skrivs av ur tidningar eller som Du byter Dig till kan vi i allmänhet inte ta in. Upphovsrätten blir oklar. Bara om det är alldeles klart att programmen av programmeraren lämnats till allmänt bruk kan vi med gott samvete använda dem i programbanken.

GÅ OCH PROGRAMMERA NU"

OK då" Läs färdigt först.

En annan kontakt i USA vill byta erfarenheter av FORTH, PASCAL och telekommunikation. Det skall han väl få.

I förra numret funderade jag över om det skulle annonseras en storebror till 99-an i somras. Så blev det inte, men ryktena fortsätter. Vi lyssnar vidare. Under tiden kan vi ånyo glädja oss åt många nya fina och avancerade progran för 99-an. Värst är nog Craig Miller med sin GRAM-Cracker, som sägs skola göra det möjligt att knäcka vilken modul som helst, föra över programmet på disk eller kassett, förbättra det och ladda in det i en modul igen. I en utbyggd version säger Craig att det blir möjligt att modifiera TI Basic och det inbyggda operativsystemet" Rent fysiskt är det en kloss som stoppas in i modulporten och som räcker ända fram till kanten på konsolen. I denna kloss sätter man sedan sin ordinarie modul, som så att säga sugs över i GRAM Cracker. Modulen kan sedan tas bort. Det här kommeratt kosta mellan 150 och 200 dollar. Multiplicera med dollarkursen och lägg till 30% för att få de svenska importpriserna ungefär.

Till sist: Vi har anslagit en del pengar till anskaffning av s k free-ware i främst USA. Det vi skaffar kommer att läggas in i programbanken.

När tog Du kontakt med en annan medlem sist? Har Du försökt genom matrikeln? Har Du ett specialintresse Du vill publicera i tidskriften? Gör det" Det är kontakterna mellan människor med gemensamma intressen det handlar om. Men vi måste göra oss synliga för varandra för att det skall bli något.

Lennart Lindberg,
ordförande

VALBEREDNINGEN: Anders Törnkvist, tfn 0755-689 20

■

MAXIMEM

av Lennart Lindberg

Från en kontakt i Kanada har vi fått en sida som berättar om en ny modul till 99-an. Modulen heter MAXIMEM och är en slags minnesexpansion. Den kräver konsol, disk och 32K minnesexpansion. Den ger vid start valmöjligheter mellan TI Basic, en förbättrad Editor/Assembler och så en katalog över moduler som lagts över på disk.

Vadå lagts över på disk? Jo, den här MAXIMEM innehåller nämligen en möjlighet att lägga över moduler på disk" Det låter som om det skulle vara något som liknar den GRAM-Cracker som Craig Miller säger att han tänker släppa ut snart(se ordförande-spalten). MAXIMEM håller data även när datorn är avstängd precis som Mini Memory.

Det här ger alltså goda möjligheter att slippa byta modul. Man kan istället laborera med program på disk - även program vi är vana vid att ha på modul som X-Basic, Editor/Assembler, TI-Writer m fl.

Det kostar 199 dollar plus frakt mm - c:a 30%. Han som säljer vill ha en månads leveranstid. Han finns i Kanada:

- Guy Gournay, 146 178 CAN INC 933 Delorimier - LONGUEUIL
P.Q. - CANADA - J4K 3M8

■

UTMANINGEN!

Redaktörens adress:
Anders Persson
Kämnärsvägen 4:1078
222 45 LUND

Kära läsare, som alla framstående svammelmakare brukar inleda sina mer eller mindre lysande alster. Jag vet inte riktigt om jag med glädje eller sorg ska konstatera att problemlösheten återigen har lagt sig tungt över församlingen. Det är ju roligt för er, men inte så lätt för mig...
Nåväl, för att fylla ut det hela lite har jag försökt redogöra för några vanliga missuppfattningar angående assemblerprogram. Jag hoppas att det kan vara till någon hjälp för en och annan.

EN MODERN PROGRAMMERINGSMILJÖ

Det låter ju fint värre, eller hur? Bo Carleö, känd bland annat för grafikprogrammet Forit, anser att det uppförande som är karakteristiskt för exempelvis Apples Macintosh är synnerligen trevligt. Det bär lite väl långt att beskriva det här för den som inte har sett det, men går i korthet ut på att olika funktioner väljs genom att peka på motsvarande symboler. Dessutom kan man köra flera olika program samtidigt genom att dela upp skärmen i olika fönster, där de olika programmen residerar. Menyer kan plockas fram när de behövs ungefär som när man drar ner en rullgardin. Sedan försvinner de igen när man har gjort sitt val. Härligheten styrs huvudsakligen med en mus, vilket är en liten apparat som man för över bordet i den riktning som man vill flytta markören (en liten pil) på skärmen. Genom att, vid lämpliga tillfällen, trycka på den knapp som finns ovanpå musen, talar man om för datorn att man vill göra något med det som pilen pekar på just då. Tangentbordet används mest till att skriva in text och siffror med.

Vad har nu det här med 99:an att göra? Jo, Bo tycker att man borde kunna åstadkomma någon sorts ramprogram i Forth, som andra program ska kunna utnyttja. Denna ram ska då ge ett slags Mac-miljö åt olika applikationer i Forth, och alltså innehålla ord som underlättar utnyttjandet av ikoner, fönster osv. Musen får väl ersättas av en joystick, även om det inte kan bli lika smidigt. Det går naturligtvis att koppla in en mus till 99:an också, men inte med något normalt tillbehör som jag känner till. Det får alltså specialbyggas, vilket inte roar alla.

Nu menar jag inte på fullt allvar att det här skulle motsvara hela rasket som Macintosh kan stå åt med. Bara minnesåtgången reser hinder i vägen för ett sådant projekt. Den minsta Mac:en är ganska handikappad som den är, trots att den har 128 kbytes från början. Men någon lämplig delmängd av Macs funktioner går nog att åstadkomma.

Jag misstänker att en och annan är böjd att hålla med Bo, när han anser att det här är en värdig Utmaning...

DET FÖRSTA ASSEMBLERPROGRAMMET

Såvitt jag kan förstå av de kontakter jag har haft med olika medlemmar, har en del av er vid upprepade tillfällen försökt göra ett assemblerprogram, men inte lyckats med något annat än de som finns som exempel i Editor/ Assembler manualen. Svårigheterna tycks främst koncentrera sig på 9900:s registersystem och sammankoppling av program i BASIC och assembler.

Detta är inte så konstigt, eftersom TMS 9900 har en ovanlig arkitektur (eller uppbyggnad, om man så vill) och samarbete med BASIC program är lite besvärligt. Jag tänkte mig nu ett tappert försök att vimsa till det hela ytterligare... nej, jag menar reda ut det här"

Innan du läser längre vill jag säga ifrån, att jag inte har för avsikt att behandla sambandet mellan bit, byte och word (ord), hur det hexadecimala talsystemet fungerar eller vad det är för skillnad på en adress och ett värde som finns på adressen. Det gör nämligen alla andra som skriver om assembler, vilket jag tycker kan räcka. Dessutom förutsätter det här att du vet, eller kan se efter, vad en instruktion egentligen gör, när du hittar den i ett exempel i den här artikeln. Det är alltså inte fråga om någon "absolut från början" kurs.

REGISTER

De flesta processorer håller sig med ett antal register. Ett sådant är programräknaren (PC för program counter), som visst inte räknar program, även om man kan tro det. PC pekar på den instruktion som ska utföras härnäst, och borde alltså kallas PP (program pointer). Men någon humoristisk amerikan bestämde sig en gång för PC, och så har det, troligtvis enligt principen pappa vet bäst, fått förbli.
Ett annat register är statusregistret (ST), vilket bl. a. används till att hålla reda på om två tal är lika eller i annat fall vilket som är störst.

Utöver de här brukar det finnas några register som kan användas för beräkningar, lagring av mellanresultat och liknande. Här är 9900 ett underligt djur, eftersom den försöker inbilla programmeraren att den har 16 register (R0 - R15), trots att den egentligen inte alls har det" I stället för att ha dessa register inuti den krets som utgör processorn, hugger 9900 till sig 16 words i minnet och betraktar dessa som sina register. Notera att det inte är sagt något i förväg om var i minnet de ska finnas. De får lov att vara på en godtycklig plats, eller adress, bara det finns något minne där.

Hur kan den då veta var de finns just nu, när jag sitter och skriver det här? Jo, den har ett register, men bara ett, som pekar på det första av dessa 16 i minnet. Detta register, som kallas för WP, talar alltså om för 9900 var R0 är beläget. Vet man det är det ju enkelt att räkna ut var R10 finns, eftersom de kommer i följd. Ta WP+10*2, så får du adressen. Tvåan kommer av att minnet adresseras byte för byte, medan ett register är två bytes stort. WP betyder workspace pointer. De sexton sammanhängande orden i minnet som utgör "låtsasregisterna" kallas för en workspace (ung. arbetsarea). I fortsättningen kallar jag R0 - R15 för register, trots att det egentligen är minne.

En berättigad fråga i sammanhanget är väl vad det här arrangemanget ska vara bra för? Fördelen är att man inte behöver nöja sig med att alltid ha samma uppsättning register. Antag att ett program som utnyttjar de tillgängliga registren anropar ett underprogram, vilket i sin tur behöver ett antal register till andra, interna uppgifter. Då kan man helt fräckt byta till en annan omgång register i underprogrammet, register som man kan härja i efter behag, utan att störa huvudprogrammets uppsättning. De finns ju i verkligheten inte på samma ställe. Dock får man se till att byta tillbaks igen, innan man återgår till huvudprogrammet. Annars blir datorn lätt konfys och kommer liksom av sig...

Nackdelen är att datorn blir långsammare, eftersom det elektriskt sett tar längre tid att få tag i ett värde som rent fysiskt finns ute i minnet, jämfört med ett som finns inuti processorkretsen. Men det behöver man inte bry sig om vid programskrivning, åtminstone inte i början.

Något som däremot kan vara bra att veta är var WS (dvs workspace, registren alltså) verkligen finns i minnet. På så vis kan man undvika att använda det minnesutrymmet till något annat.

R11 - R15 har speciella funktioner, som vi snart ska se, men om man bara känner till dessa kan man ändå utnyttja dessa register till egna uppgifter, vid de tillfällen då detta inte kolliderar med specialiteterna.

SUBRUTINER

Underprogram är viktiga hjälpmedel i alla språk. Gemensamt för de olika varianter som finns är att man, när underprogrammet är klart, ska återvända till instruktionen som finns efter den man kom ifrån. För att detta ska fungera måste datorn på något sätt komma ihåg var den kom ifrån. 9900 utnyttjar R11 till detta. När den anropar en subrutin, lagras automatiskt återhoppsadressen i R11. Efter exekvering av subrutinen utnyttjar man den adressen till att komma tillbaks på rätt ställe i huvudprogrammet igen.

Se här ett exempel där ett program lägger in några olika värden i R0, och sedan anropar underprogrammet ADD100, vilket lägger till 100 till talet i R0. Det här programmet, liksom alla andra exempel i den här artiklen, har ett utseende som E/A modulen tycker om. För Mini Memory blir det något annorlunda, huvudsakligen beroende på att labels bara får bestå av två tecken, inte sex. De kommentarer som är skrivna med gemena bokstäver har dock inte alltid en form som E/A modulen kan hantera. Att de ser ut som de gör här beror på rent praktiska orsaker.
Krumeluren före ADD100 ska vara "kanelbullen", SHIFT 2. Skrivarna kan ju misshandla assemblerprogram till oigenkännelighet ibland.

```
      LI   R0,125
      BL   'ADD100
```
Här lagrar BL automatiskt adressen till nästa instruktion i R11.
```
      MOV  R0,R1
```
Resultatet sparas i R1.
```
      LI   R0,480
      BL   'ADD100
      MOV  R0,R2
```
Det här resultatet lagras i R2.
Vad svaren ska användas till är ovesäntligt för stunden. Det är nämligen hanteringen av R11 som är intressant.

Någon annan stans i programmet finns ADD100

```
ADD100 AI   R0,100
       B    *R11
```
Vid återhoppet utnyttjas innehållet i R11 som adress. Det gör att datorn återvänder till platsen efter rätt anrop varje gång. *R11 i stället för R11 används för att tala om att datorn inte ska fortsätta köra program i R11, utan på det ställe som R11 anger. Som ren kuriosa kan nämnas att man faktiskt kan köra program i WS också, eftersom det i verkligheten är minne. Resultaten av sådana manövrer är däremot knappast ägnade att underlätta förståelsen för programmet ifråga.

Vad gör man då om ett subprogram vill anropa ett annat? Om A anropar B, lagras returadressen till A i R11. Men om sedan B anropar C, ersätts värdet i R11 med returen till B. Från C tillbaka till B går det bra, men när R11 sedan används igen för att komma tillbaka till A, kommer man i stället "tillbaka" till B. B hoppar alltså tillbaka till sig själv, fortsätter med sista delen av B, hoppar tillbaka till sig själv... Det kan den roa sig med länge, länge, allt medan programmeraren (med andra ord du) blir mer och mer fundersam över vad den egentligen sysslar med.

Knepet är att spara undan returen till A på någon annan plats, innan den skrivs över av returen till B. Se här ett nytt exempel, där ADD100 fullgör sin uppgift genom att anropa ADD50 två gånger. Hrm, jag ser att exemplen blir dummare och dummare, ju längre jag håller på...

```
      LI   R0,125
      BL   'ADD100
      MOV  R0,R1
      LI   R0,480
      BL   'ADD100
      MOV  R0,R1
```
Det här är precis det samma som ovan. Skillnaden kommer i ADD100.
Här uppstår naturligtvis frågan om var man ska göra av returen till huvudprogrammet. Det enklaste är att lagra den i ett annat, ledigt register.

```
ADD100 MOV  R11,R10
       BL   'ADD50
       BL   'ADD50
       B    *R10
```
Här finns ju den önskade adressen i R10, varför man lika gärna kan skriva *R10 med en gång. Det kvittar datorn lika om man använder *R10 eller *R11 eller *R-vad-som-helst, bara man tar det rätta, vill säga...

```
ADD50  AI   R0,50
       B    *R11
```
Här behövs inga specialmanövrer. Det gäller alltid för den sista av subrutinerna i kedjan.

Nå, men antag nu att varken R10 eller något annat register är ledigt till att komma ihåg returadresser. Var ska man då göra av den?
Jo, på något annat ställe i minnet, helt enkelt" Om man definierar något ord som kan heta SAVRTN, kan man spara returen där i stället. ADD100 ovan ser då ut så här:

```
ADD100 MOV  R11,'SAVRTN
       BL   'ADD50
       BL   'ADD50
       MOV  'SAVRTN,R11
       B    *R11
```

Dessutom måste det ju finnas ett
```
SAVRTN DATA 0
```
någonstans i programmet. Övriga delar av programmet ändras inte något.
Men vad är det för dumheter i slutet av ADD100? Kan man inte skriva
```
       B    *SAVRTN
```
när returen finns i SAVRTN? Nej, det kan man inte, helt enkelt därför att TMS 9900 inte klarar av * ihop med något annat än ett register. Eller, mer högtravande, indirekt adressering kan endast ske via register. SAVRTN är en minnesadress, och indirekt via minnet finns inte, inte på den här datorn vill säga.

Så långt allt väl. Men om nu A anropar B, som anropar C, som anropar D, som... Det blir en förskräcklig massa olika SAVRTN (SAVRT1, SAVRT2, SAVRT3 osv). Dessutom är det lätt att det blir kollisioner i alla fall, om plötsligt F kan anropas både från A-B-C-D-E-F, från A-G-H-I-F och från A-G-H-C-D-E-F, beroende på olika villkor i rutinerna. Resultatet blir att varje rutin måste ha sin lagringsplats. Men det är ju dumt om det finns 100 olika rutiner, men aldrig mer än 10 är på gång samtidigt. Då har man alltid minst 90 ounyttjade adressplatser, vilket är 180 bytes. Det är rätt mycket det, om programmet skrivs in med radassemblatorn i Mini Memory. Dessutom kan man inte klara fall där ett underprogram anropar sig självt. (Skratta lagom" Det kallas rekursivt anrop, och används förvisso ibland.)

En bättre metod är då att utnyttja en stack. Det är en minnesarea där man trycker på returadresser i den ordning som de behöver sparas när anropen görs, och sedan hämtar tillbaks dem i motsatt ordning när returhoppen ska ske. Det är uppenbart att det här inte fungerar om man inte kan vara säker på att man får tillbaka adresserna i exakt motsatt ordning, jämfört med när de sparades. För att klara det behöver man ett register som brukar kallas för en stackpekare (SP). Den talar om vilket värde i stacken som står i tur när man ska hämta tillbaks ett, respektive var ett värde ska placeras när man vill lagra det.

På 99:an utnyttjar man något av de vanliga registren till SP. Vilket spelar ingen roll, så länge det inte måste användas till något annat. R11 är, av förhoppningsvis uppenbara skäl, klart olämpligt som SP. Låt oss ta R14. (Jag har visserligen tidigare sagt att det är ett register med specialuppdrag, det också, men det kan vi glatt strunta i för tillfället. Om någon tvunget ska ha ett motiv, kan väl det faktum att R14 är SP i en DEC VAX-11 minidator vara skäl nog")

Eftersom man får lov att referera till R14 både med just R14 och dessutom med bara 14, kan man göra en liten fint och plötsligt kalla R14 för SP. Om programmet innehåller
```
SP     EQU  14
```
anser assemblern att SP är detsamma som 14.

Vad måste då läggas till vårt roliga exempel ovan, för att det här ska fungera?

Dels måste man ha en stack...

```
SP     EQU  14
STACK  BES  100
```

...och dels måste ADD100 spara R11 på stacken, och även hämta tillbaks värdet därifrån...

```
ADD100 DECT SP
       MOV  R11,*SP
       BL   'ADD50
       BL   'ADD50
       MOV  *SP+,R11
       B    *R11
```

...och till sist, men inte minst, måste SP få ett värde i början av HUVUDprogrammet...

```
       LI   SP,STACK
```

Vad innebär nu detta, egentligen?

```
STACK  BES  100
```

reserverar en 100 bytes stor stack. På den kan man då lagra 50 adresser, vilket torde vara en bra bit mer än du behöver den här sidan jul, så länge du nöjer dig med att lagra returadresser på stacken. Den kan ju användas till annat också, om man vill och behöver.
Eftersom direktivet BES används (inte BSS), kommer STACK att vara adressen efter stacken. När man har gjort

```
       LI   SP,STACK
```

pekar SP alltså utanför det reserverade utrymmet. För att nu inte tramsa till det måste vi alltså först minska SP med två och sedan flytta dit det vi vill spara.

```
       DECT SP
       MOV  R11,*SP
```

gör just detta.

När man sedan ska ha tag i värdet igen, gäller det ju att först hämta det och sedan öka SP med två igen. Annars kommer SP i "otakt", varvid det blir omöjligt att veta vilket värde i stacken som man egentligen ska hämta härnäst.

```
       MOV  *SP,R11
       INCT SP
```

fungerar alltså bra, men det finns ett adresseringssätt som gör båda dessa operationer i ett svep, enligt ovan.

```
       MOV  *SP+,R11
```

ser till att först hämtas värdet som SP pekar på, sedan ökas SP med två. Tragiskt nog finns det inte något som heter

```
       MOV  R11,*-SP
```

vilket alltså först skulle minska SP med två och sedan flytta R11 dit SP nu pekar. Somliga processorer klarar det också, men vi får klara oss med tvåstegsvarianten i början av ADD100 ovan.

Härmed har vi klarat ut hur subrutiner som anropas med BL kan hanteras. För att undvika omgående idiotförklaring vill jag gärna påpeka att det enda vettiga sättet att skriva programmet ovan är så här:

```
       LI   R1,225
       LI   R2,580
```

Dessutom kan man som allmäna tips säga: Använd register i nummerordning. Då är det mycket lättare att hålla ordning på vilka man har utnyttjat och vilka som är lediga. Dokumentera vad registren används till. Om ett register har en och samma uppgift i hela programmet, kan det vara bra att ge det ett namn (jämför SP ovan).

HUR MAN BYTER WORKSPACE

Antag nu att vi har ett program som rotar omkring med sina register, och gärna vill ha dem ifred. Men av och till anropar vi SUB1, som också vill rota omkring i register efter eget behag. Jag har redan avslöjat lösningen på det här, nämligen: Byt registeruppsättning"

I ett program kan det se ut så här:

```
MAINWS BSS  32     Huvudworkspace
SUB1WS BSS  32     Underworkspace nr 1


MAIN   LWPI MAINWS
```

I början av programmet ser man till att ladda in rätt WS.

Här följer diverse instruktioner.

```
       BL   'SUB1
```

Mer instruktioner.

```
       BL   'SUB1
```

Ännu fler instruktioner.

Slut på huvudprogrammet. Var man då tar vägen ska vi snart bekymra oss om.

Här finns underprogrammen.

```
SUB1   LWPI SUB1WS
```

Ladda genast egna register.
Härja, härja med dem utan att det gör något"

```
       BL   'SUB2
```

Det här går ju också bra, eftersom det inte är det kritiska R11 som behövs för att komma tillbaka till huvudprogrammet som utnyttjas nu.

```
       LWPI MAINWS
       B    *R11
```

Det gamla R11 finns kvar i den förra WS som användes. Lämpligen kan man stanna upp ett tag här och fundera över hur man ska få tag i ett värde som finns i R0 i MAINWS från SUB1. En metod är att innan

```
       LWPI SUB1WS
```

i SUB1 kopiera värdet till något ställe som vi kan hitta även med SUB1WS.

```
       MOV  R0,'TEMP
       LWPI SUB1WS
       MOV  'TEMP,R0
```

På det här viset får man R0 i MAINWS över till R0 i SUB1WS. Men det finns ett enklare, om än kanske inte lika självklart sätt.
Eftersom registren egentligen är minne, kan man komma åt ett register på mer än ett sätt. R0 i MAINWS kan adresseras både som R0 (när MAINWS är i bruk) och som 'MAINWS. R4 har adressen MAINWS+8.
Dessa adresser ändras naturligtvis inte när vi byter till SUB1WS. Alltså kan SUB1 inledas med

```
SUB1   LWPI SUB1WS
       MOV  'MAINWS,R0
```

om det är R0 (i MAINWS) som innehåller ett intressant värde. Resultat kan skickas tillbaka på motsvarande sätt.

CONTEXT SWITCHING

Herregud, vad är nu detta?

Jo, det är ett uttryck för något som används när man i ett program mer eller mindre huvudstupa kastar sig från en uppgift till något annat. Detta hindrar inte att uppgifterna kan ha något samband, men de är i alla fall olika sidor av samma sak. Fritt översatt betyder det sammanhangsbyte.

Kännetecknande för en context switch i datorsammanhang är att man, när ett byte har skett från A till B och tillbaka till A igen, "sopar undan" spåren efter sig. A ska inte märka, eller åtminstone inte störas av, att B har ägt rum. Ett typexempel:

Antag att vi kör ett program på datorn. En stund senare kör vi programmet igen, men nu börjar vi ha bråttom. Därför vill vi ha aktuellt klockslag på skärmen, så att vi kan hetsa upp oss i lagom mån allt eftersom tiden går. Genom att ha någon klocka i systemet kan vi läsa av den exempelvis 10 gånger i sekunden och skriva ut värdet på skärmen. Men eftersom det program som vi egentligen kör tar mer än 1/10 sekund, innebär detta att vi måste avbryta det vi håller på med, skriva ut tiden, fortsätta, avbryta igen osv.

Om nu utskriften av tiden på skärmen på något sätt påverkar vårt andra program, till exempel genom att ändra något av "våra" register, blir resultatet olika, beroende på om klockan var igång eller ej. En sådan tingens ordning kan vi givetvis inte acceptera. Utskriften av tiden är här en klart avgränsad uppgift, och en context switch ska äga rum när datorn byter arbetsuppgift. (Inom parentes kan sägas att det här inte är några fria fantasier, utan precis vad jag kan göra med min maskin, med hjälp av det klockkort som är mitt senaste tillbehörspåfund.)

Ett något mindre "avlägset" exempel är utskrift på skärmen, av vilken text som helst, från ett assemblerprogram. Då är det inte frågan om något avbrott som kommer när vi minst anar det, utan en i högsta grad planerad context switch. Varför ska man då överhuvudtaget göra en context switch bara för att skriva på skärmen, när det inte är nödvändigt (det är det nämligen inte)?

Se det så här: I vårt program har vi en text som ska skrivas på ett visst ställe. Genom att ange var texten finns, hur lång den är och var på skärmen den ska hamna, har ett underprogram som sköter utskriften alla nödvändiga uppgifter. När väl programmet anropats, ÄR DET OSS FULLKOMLIGT LIKGILTIGT HUR DET GÅR TILL ATT SKRIVA PÅ SKÄRMEN, bara det blir gjort. Varför då besvära oss med att ta hänsyn till hur det går till? Om utskriften rentav kan ske på ett sådant sätt att vi inte kan märka det alls, är det så mycket bättre. Vi vet ju ändå att den har skett, eftersom vi vet vad vi anropade. Finessen är att vi har en garanti för att utskriften inte ställde till något för oss i "vårt" program. (Jag bortser här från "illvilliga" program som mer eller mindre dolt ställer till med något på rent jäkelskap.)

Nu är det naturligvis så på en maskin som 99:an, att vi alltid kan avgöra vad som har hänt efter en context switch, kanske genom att "spana" in i det minnesutrymme som den andra rutinen använder. Men det räknas i sammanhanget som ett "fult" knep, och kan lämnas därhän så länge vi försöker skriva välartade program, program som vi lätt ska kunna hantera.

Efter all denna teori torde det vara hög tid att klämma fram hur det går till. Planerad context switching göres med instruktionen BLWP. (Hur det uttalas? Fråga TE-II med pratorn") Utskriftsexemplet ovan var i själva verket precis vad som händer när man skriver

```
      BLWP 'VMBW
```

i sitt program.

Vad gör då BLWP? Genom ett enkelt resonemang är det ganska lätt att inse vad, om än inte hur. Eftersom det är ett slags anrop av ett underprogram, måste BLWP någonstans lagra adressen till nästa instruktion. Dessutom måste den byta register - vi såg ju innan att nya register är ett effektivt sätt för olika program att uppnå självständighet - och alltså även hålla reda på vilka vi hade innan. De ska ju åter träda i funktion efter returhoppet. Dessutom, vilket kanske inte är helt uppenbart av det hittillsvarande resonemanget, ser BLWP till att ST kan återställas till utseendet före anropet. Statusregistret beskriver ju resultatet av den sista jämförelsen som processorn gjorde, något som naturligtvis ändras i den anropade rutinen, eftersom det bara finns ett ST och nästan alla instruktioner påverkar ST på något sätt. Då vi vill att context switchen skall märkas så lite som möjligt, är det naturligt att även ST återställs. Det är för övrigt helt nödvändigt i vissa fall då context switch sker genom avbrott (jämför klockan).

Vid återhoppet är det alltså tre saker som processorn måste veta, nämligen vad PC, WP och ST ska bli. Detta måste lagras någonstans, och 9900 använder de tre sista registren (R13, R14 och R15) till detta. Observera att det är registren i den WS som underprogrammet (VMBW i utskriftsexemplet) använder som utnyttjas. Ordningen är R13=WP, R14=PC och R15=ST. Instruktionen RTWP avslutar en sådan rutin, genom att flytta värdena från låtsasregistren ovan till deras verkliga motsvarigheter.

En stunds eftertanke leder till frågan hur BLWP kan veta både var subrutinen finns och vilken WS den ska ha?

```
      BLWP 'VMBW
```

innehåller ju bara en adress. Att det ändå kan fungera beror på att BLWP utför något så fint som ett vektoriserat hopp("). Adressen efter BLWP talar om var denna hoppvektor (transfer vector) finns någonstans. Vektorn i sin tur, talar om var programmet börjar och vilken WS som ska användas. Varde exempel:

```
SUB1WS BSS  32      WS som ska användas

Vektorn:
SUB1   DATA SUB1WS  Adress till WS
       DATA START1  Adress till program
```

Här kan finnas vad som helst. Det är inget krav att SUB1 ska finnas i närheten av programmet START1, eller nära SUB1WS för den delen.

```
START1 instruktion
Här börjar programmet...

... och här slutar det.
      RTWP
```

Observera att det inte går att använda R14 som SP inuti ett program av den här typen, eftersom R14 är upptaget av information för återhoppet. Men det finns inget hinder för att använda R13, R14 och R15 i huvudprogrammet, eftersom det inte ska göra någon RTWP. Dessutom går det bra att göra BL en nivå djupt, eftersom R11 där varken har med RTWP eller med huvudprogrammets R11 att göra. Dessutom är det oftast så, att en stack som utnyttjas av ett program inom en context, inte ska kunna nås av ett program som arbetar inom en annan context. Program som har R13 osv upptagna kan naturligtvis använda något annat register som SP, om de behöver en stack.

I och med att R13 pekar på den WS som användes innan, kan man komma åt R0 i förra WS med *R13. R7 kan nås med '14(R13), vilket innebär att 14+innehållet i R13 används som adress.

SUBRUTINER TILL BASIC

Vi har nu tittat på hur underprogram återvänder till huvudprogram i olika situationer. Hur återvänder då huvudprogrammet till systemet, när körningen avslutas? Genom att betrakta vårt huvudprogram som ett underprogram till systemet blir svaret uppenbart: Det finns ett R11 någonstans. Var finns då det, eller, annorlunda uttryckt, vad är WP när BASIC anropar ett assemblerprogram?

Svaret på det här är: Det beror på" Den WS som används vid anropet är >83E0 om det är Extended BASIC som anropar, >70B8 om TI BASIC anropar via Mini Memory eller >20BA om E/A används. Detta gäller dock inte för automatstartande program. Då är den >83E0. Assemblerprogram under provning med hjälp av E/A:s debugger kan anropas med godtycklig WP, medan de som testas med Easy Bug alltid anropas med WP= >83E0, när Execute beordras. Det R11 som ska användas är i samtliga fall det som finns i den WS som är aktiv vid anropet. Gissa en gång om den här röran krånglar till det hela en aning.

EXTENDED BASIC

För att inte försöka redogöra för alltför mycket på en gång tänker jag begränsa mig till assemblerprogram som anropas av X-BASIC. De är ju också de intressantaste, eftersom man då kan förena möjligheterna hos Extended BASIC med de konster som kan utföras i assembler. Nackdelen är att det krävs en hel del prylar för att kunna göra det här: Skivminne, E/A modul, minnesexpansion och X-BASIC. Huruvida den delen av läsarskaran som har tillgång till exempelvis Mini Memory, men inget av det andra, önskar liknande utläggningar inom framtida Utmaningar vet jag inte, men jag räknar kallt med att DU i så fall ringer/skriver och berättar det"

Nåväl, vid anrop med CALL LINK har man alltså en WS vid >83E0. Det är den WS som går under beteckningen GPLWS. Den är inte bra att använda generellt, eftersom man inte ostraffat kan utnyttja alla register i den. Dessutom använder de systemprogram som man eventuellt behöver anropa undantagslöst GPLWS, och kan alltså styggt nog röra om i de register du använt. Botmedlet är att ladda in ett eget värde i WP vid rutinens början, och återgå till GPLWS när det är dags för återhopp. Så här kan det se ut när man har gjort en CALL LINK("KALLE") i X-BASIC.

```
       DEF  KALLE  För LINK

MYWS   BSS  32
GPLWS  EQU  >83E0
GPLST  EQU  >837C

KALLE  LWPI MYWS
```

Här kommer programmet.

```
       CLR  'GPLST
       LWPI GPLWS
       B    *R11
```

Det där med CLR 'GPLST beror på att den byte som kallas GPL STATUS BYTE används för att ge felmeddelanden till BASIC. Om den har satts till något annat än noll i programmet, kan man få ett oavsiktligt fel vid återhopp till BASIC, med därtill hörande BÖÖP. Egentligen är GPLST bara en byte, medan CLR raderar ett ord, men den byte som kommer efter GPLST kan man radera utan påföljd när man återvänder från ett assemblerprogram.

Låt oss nu se på ett litet program. Det ska anropas med CALL LINK ("INC2", TAL). Programmet ska ta emot flyttalet TAL, omvandla det till ett heltal (16-bitars), lägga till två, omvandla till flyttal igen och lämna tillbaka svaret.

Ett sådant här program kan ta nytta av en del systemrutiner som redan finns färdiga på olika ställen i minnet.

NUMREF hämtar en numerisk parameter.
NUMASG ger en dylik parameter ett värde.
CFI (Convert Floating to Integer) omvandlar flyttal till heltal.
CIF (Convert Integer to Floating) är motsatsen.
XMLLNK förenklar anropet av omvandlingarna.

NUMREF hämtar ett värde till FAC (Floating Point Accumulator), vilket helt enkelt är ett ställe i minnet som TI valde till att utföra flyttalsoperationer på. Det motsvarar ungefär sifferfönstret på en räknare, bortsett från att det inte "syns". CFI och CIF använder FAC vid omvandlingarna, och NUMASG använder FAC vid tilldelningen. Se E/A manual, kap 16 och 17.

Först ska vi ha lite adresser till diverse platser. De finns i 24.3.1 och 24.4.8 i ovannämda bibel.

```
NUMASG EQU  >2008
NUMREF EQU  >200C
XMLLNK EQU  >2018
CFI    EQU  >12B8
CIF    EQU  >20
FAC    EQU  >834A
GPLST  EQU  >837C
GPLWS  EQU  >83E0

Så vår egen WS

OURWS  BSS  32

Extern definition av INC2

       DEF  INC2

Själva programmet.

INC2
       LWPI OURWS
       CLR  R0        Ange vilken parameter
       LI   R1,1
       BLWP 'NUMREF   Talet till FAC
       BLWP 'XMLLNK   Omvandla till heltal
       DATA CFI

       INCT 'FAC      Själva operationen

       BLWP 'XMLLNK   Tillbaks till flyttal
       DATA CIF
       BLWP 'NUMASG   Tilldela parametern
       CLR  'GPLST    Förbered återhopp
       LWPI GPLWS
       B    *R11
```

Finessen med att använda XMLLNK (det är nämligen inte nödvändigt) är att CFI och CIF måste anropas med GPLWS. Dessutom måste R11 i GPLWS sparas undan och sedan återställas. Detta gör XMLLNK automatiskt. Ovanpå detta skiljer XMLLNK automatiskt på CFI, som är adressen till rutinen i konsolens ROM, och CIF, som är adressen till en tabell i ett ROM i Extended BASIC modulen.

En intressant iakttagelse är att stycket längst ner på sidan 261 i E/A manualen (Note: This routine...) är en ren lögn. CIF finns visst, om än inte i minnesexpansionen. Men var den är har ju ingen betydelse. Den är för övrigt skriven på exakt samma sätt i modulen.

BASIC programmet kan se ut så här:

```
CALL INIT
CALL LOAD("DSK1.INC20")
INPUT "TALET? ":TAL
CALL LINK("INC2",TAL)
PRINT "NYTT TAL";TAL
```

CALL INIT och CALL LOAD behövs bara en gång, eftersom assemblerprogrammet stannar kvar i minnet efter inladdningen. Det gäller även om man skriver NEW och raderar sitt BASIC program. Men en ny CALL INIT tar bort tidigare laddade assemblerprogram. Strängt taget gör den inte det, men den tar bort informationen om det faktum att programmet laddats, varför man inte kommer åt det med CALL LINK längre.

Det går bra att skriva flera CALL LOAD efter en CALL INIT, eftersom laddaren är intelligent nog att lägga nya program efter de tidigare i minnet.

Så här dags har jag naturligtvis bara skrapat lite på ytan av allt som behandlar programmering av TMS 9900 i 99:an. Men jag misstänker att en del av er, de som vet lagom mycket för att ha nytta av artikeln, kan behöva en paus nu.

Synpunkter på vad som bör behandlas mottages tacksamt.

REALTIDSKLOCKA

Det här är något som väl egentligen hör hemma i den tekniska avdelningen, men jag trampar på som vanligt i alla fall. En och annan av er har väl uppsnappat ryktet att jag hållit på med en klocka till 99:an nu en tid. Arbetet har mest stått stilla, men har nu kommit så långt att klockan fungerar. Den är försedd med batteri, så att den håller igång även när resten av datorn vilar sig. Noggrannheten är tämligen bra. För närvarande är avvikelsen mindre är 1 PPM, vilket motsvarar ungefär 30 sekunder per år.

Programvaran till klockan blir nu nästa steg. Tanken är att klockan ska kunna anropas som en fil från BASIC. Dessutom inbillar jag mig att det ska gå att ordna så att ordbehandlare typ TI-Writer ska kunna läsa in dagens datum och klockslag direkt i den text man håller på att skriva. Detta utan att ändra i TI-Writer, alltså.

Någon undrar kanske vad man ska med en klocka till. Sådana har ju de flesta på armen. Själv ser jag flera anledningar.

Den ger en möjlighet att skriva ut aktuell tid tillsammans med programlistningar. Det underlättar om man har flera olika varianter av ett program listade. Inga problem att avgöra vilken version som är den senaste.

I vissa program är det intressant att veta hur lång tid som har förflutit. Antag att man använder datorn som hjälpmedel till en speaker på en idrottstävling. Då kan man lätt få exakt information om exempelvis hur lång tid en löpare som inte är i mål har på sig, för att slå någon som redan är kommen.

Tidtagning på olika program kan också göras (så kallade benchmarks). I Programbiten 84-2 fanns ett sorteringsprogram skrivet i assembler. Det är tillräckligt snabbt för att det ska vara omöjligt att ta tid på det med ett vanligt stoppur. Men genom att använda den här klockan har jag nu konstaterat att det sorterar 1000 heltal på 0.477 sekunder...

Om man använder p-systemet lagrar det information om när en fil skapades i diskkatalogen. Om man ändrar något i filen uppdateras datumet, så att man kan se när en fil modifierades sist. Bra när flera varianter på samma program finns på en diskett. Men för att systemet ska veta vad det är för datum idag, måste man gå in och ändra datumet varje gång systemet startas. Det enda tillfälle då detta inte behövs är om det har startats tidigare samma dag, eftersom datum för förra körningen lagras på disketten.
Genom att nu utnyttja min klocka får jag en helt automatisk ändring av systemets datum varje gång p-systemet startas. Dessutom ordnade jag så att jag får reda på när den diskett som sitter i användes sist, både med datum, veckodag och klockslag.

Klockan sitter på ett kort i expansionsboxen. Men för närvarande passar det inte i normala boxar, eftersom kontakten är ett europadon, inte en kortkantskontakt. Jag har helt enkelt bytt ut några kontakter i boxen. Någon forskning pågår dock beträffande ett lämpligt kort att bygga på, så att klockan kan passa i vilken box som helst. Visst intresse tycks finnas, varför jag då kanske kan börja bygga klockor i parti och minut...

ÅRSTIDEN

Nu är ju vintern på gång, med tillhörande mörker och kyla. Då kan man kanske tänka sig att en och annan av er kryper in i något hål och sätter igång datorn, om inte annat så för att den ger en del värme ifrån sig. Vågar jag dessutom tänka lite till och tro att ni då kommer med några små uppslag till den här avdelningen?

I sammanhanget kan jag nämna att jag tycker att det är kul att Sven Lundgren i Staffanstorp äntligen svarat på mina utmaningar angående en träff, och ordnar en sådan. Om ni nu sätter igång allihop och fixar varsin, bör vi kunna ha en i månaden åtminstone 30 år framöver"

Vi ses(?)" ■

MODIFIERAD LT-WRITER

av Lennart Thelander

Här är en helt ny version av LTWRITER. Det som skiljer sig mest är följande:

Input har blivit snabbare om man vill skjuta in en rad i början på texten. Förut gjordes en del onödigt arbete.

Change och Locate fungerar på annat sätt nu. Den delimiter man vill ha skriver man direkt på raden i stället omvägen via eXchange delimiter (den sistnämnda är borttagen). Emempel: C%10/10%10/11%12 (byter 10/10 mot 10/11 på rad 12). Nu behöver man inte ange radnummer i change om det är den aktuella raden (actline) som avses.

ñ har kommit till. På detta sätt får man snabbt reda på hur mycket text man har (i rader).

Write to file. Om RS232 anges som filnamn så sker det omvandling av koder till printern. CTRL-koderna från tangentbordet ligger ju över 128 och inte under 32 som annars är brukligt.

CTRL-B = escape (skickar ett escape tecken till skrivaren)

CTRL-D = startkod. Om man t.ex. vill skriva ut kod 138 så skriver man "CTRL-D138" i texten. CTRL-D följes alltså av 1 till 3 siffror som är numret på det tecken som skickas ut.

CTRL-E och CTRL-F är interna koder för LTWRITER och tas bort vid utskrift.

CTRL-H = backspace

CTRL-I = Horisontal tab

CTRL-J = Line feed

CTRL-K = Vertikal tab

CTRL-L = Form feed

CTRL-M = Carriage return

CTRL-N = Shift out

CTRL-O = Shift in

Dessa koder skrivs också ut på skärmen. Omvandlingen av koderna har Anders Persson skrivit i assembler så de går snabbt.

Förut fanns en liten bug vid utskrift. Om RS232 kortet väntar på skrivaren kollas FCTN-4 tangenten. Om tangenten är nedtryckt generas ett I/O fel som kunde få LTWRITER att inte längre fungera. Tack vare den förträffliga instruktionen ON ERROR har detta avhjälpts. Utskriften kan nu avbrytas var som helst genom att FCTN-4 hålls nere (CALL KEY mellan varje rad).

Du som vill ha den nya versionen kan sända in 10 kr samt en tom kassett eller flexskiva och ett frankerat svarskuvert till:

Lennart Thelander
Dalhemsvägen 103A
252 65 HELSINGBORG

Tel. 042-15 18 73 ■

FRÅN 16K TILL 128K

av Lennart Lindberg

Hösten 1983 ville jag bygga ut min TI 99/4A med expansionsminne och disk drive. Valet stod mellan 32K-minne och två st disk drives å ena sidan och 128K-minne och en disk drive å den andra. Jag räknade med att kunna använda 128K-minnet bl a som en sk RAM-disk, d v s kunna simulera en disk drive i minnet. Iden var då att läsa in programskivans innehåll i minnet och sedan hämta programmen där mycket snabbt.

Jag köpte 128K-minnet från det lilla amerikanska företaget Foundation, som just då annonserade detta minne som det första 128K-minnet till 99-an på marknaden. Vi var några stycken i föreningen som importerade från USA.

När paketet kom blev det spännande värre. Hur skulle det här fungera?

Tyvärr blev jag ganska besviken. Jag hade litet naivt trott att jag skulle kunna adressera 128K som vanligt RAM-minne. Så var det nu inte.

128K-minnet visade sig bestå av fyra st skilda 32K-minnen monterade i ett kort för expansionsboxen. Varje 32K-minne var uppbyggt av 4 st 8K-RAM chips. De fyra 32K-minnena kunde kopplas in i adresseringssystemet ett i taget med hjälp av några enkla CRU-instruktioner i assembler. Med hjälp av programmering kunde man sedan med fantasi och skicklighet konstruera program som utnyttjade detta. Problemet var bl a att de här växlingsinstruktionerna ju måste ligga någonstans och då gärna i en del av minnet som inte kopplades i och ur. Fabrikanten rekommenderade att lägga dem i Mini Memory.

Mors" Jag har inget Mini Memory, jag ville använda minnet med TI FORTH, som kräver E/A-modulen, eller med X-Basic, som ju har en egen modul. Dessutom var jag inte tillräckligt slängd för att programmera erforderliga rutiner i assembler.

Senare lånade jag ut kortet till Tony Rogvall, som mera på skoj skrev en rutin för "bank switching" - det heter så när man kopplar in och ur olika minnesbanker - som han lade med en del i varje minnesbank och som kunde skifta banker och hoppa till den del av programmet som fanns i den ny-inkopplade banken. Men det var också det enda den kunde så det var inte mycket nytta med den - bortsett från att den visade att minnet fungerade.

Bankerna är numrerade 0,1,2,3. När maskinen slås på kopplas bank 0 in och fungerar då som ett vanligt 32K-minne. De andra tre finns där tillgängliga för den som kan använda dem. Nu hade förstås fabrikanten tänkt på ett sätt att använda dem utan Mini Memory och assemblerprogrammering. Han har lagt in ett ROM-chip med DSR-rutiner i. DSR betyder Device Service Routin och är en liten programsnutt som finns till varje "Device" som kan kopplas till 99-an - disk drive, printer etc. Rutinen ser till att "devicen" kan anropas på ett standardiserat sätt från BIOS - Basic Input/Output System, dvs de rutiner som hanterar in/ut-matning och som är inbyggda i maskinen när den levereras. Det är de rutinerna som - tillsammans med olika DSR - bestämmer att en disk drive anropas med DSKn., där n är numret t ex, och som bestämmer att bandspelaren anropas med CS1 eller CS2 och som innehåller de rutiner som avgör hur filer får se ut - DISPLAY, FIXED etc. Jag är oklar över hur uppgifterna är fördelade mellan BIOS kärna och de olika DSR, men det går uppenbarligen att via nya DSR skapa nya namn på in/ut-enheter.

(Låt mig här parentetiskt nämna att Tony Rogvall tycker att BIOS i 99-an och en del DSR kunde vara bra mycket smartare än de nu är. Han har då bl a skrivit ett disassemblerprogram som gör att han kan undersöka hur dessa rutiner är skrivna. Nästa steg förefaller vara att han river ut de rutiner som finns och sätter in egna. I så fall får vi nog höra talas om det")

Den där DSR i 128K-kortet innehåller ett program som tillåter BIOS att hitta filer som heter MEM96, MEM96A, MEM96B och MEM96C. Talet 96 kommer av 3x32, vilket emellertid inte är helt korrekt. De tre lediga bankerna 1,2,3 innehåller visserligen 96K men tillgängligt för filer är endast 32+32+24. I bank 3 är nämligen 8K RAM utbytta mot ett 8K ROM-chip innehållande DSR-rutinerna.

MEM-filerna kan användas antingen så att hela utrymmet tilldelas en fil, som då heter MEM96, eller så att utrymmet delas på tre filer: MEM96A och MEM96B på max 32K vardera och MEM96C på 24K.

MEM-filerna kan lätt hanteras i X-Basic eller i TI FORTH, vilket jag har provat. Lustigt nog är de tillgängliga även från Basic har jag upptäckt. Det lustiga sammanhänger med att Basic inte kan arbeta i expansionsminnet. Däremot hittar Basic MEM-DSR och MEM-filerna"

Tyvärr är den DSR som skrivits inte särskilt sofistikerad. Den tillåter t ex endast filer av RELATIVE-typ och med FIXED recordlängd i UPDATE mode. Den tillåter inte att RELATIVE-filer läses sekvensiellt och ger ingen EOF-signal. Det här betyder att man för att kunna bekvämt använda MEM måste programmera egna in/ut-rutiner som håller reda på saker som operativsystemet och DSR faktiskt borde klara. Men det går alltså att lagra data i MEM vilket naturligtvis ger stora möjligheter att få snabb tillgång till ansenliga datavolymer mätt med hemdatormått.

Ett besvärande problem för mig var att jag länge trodde att den här DSR inte fanns i mitt kort eftersom jag inte fick tag i den med de anrop jag gjorde. Troligen berodde det på någon glappkontakt i kortet, som jag behandlade med sprittvättning bl a sedan jag förstått av kamrater att de hade DSR i sina kort.

Nu har emellertid ett lyckligare tillstånd inträtt ty i somras fick jag äntligen för mig att beställa en förbättring av DSR-chip-et från Foundation. Det kom i en liten låda tillsamman med ett brev som beskrev hur jag skulle skruva isär kortet och peta i det nya chip-et och med en beskrivning av vad de nya DSR kunde göra för mig. För att börja med det rent praktiska : det är faktiskt lätt att förstöra ett sådant där litet chip när man sätter i det. Benen är små, många och sköra. Jag lyckades ordna till en glappkontakt genom att böja ett eller ett par ben. Hade inte Tony varit med tillsammans med sitt disassemblerprogram skulle det nog inte gått bra.

Den här nya DSR har kvar de gamla egenskaperna men har fått några till. Det intressantaste är att jag nu kan anropa en fiktiv disk som heter DSKx. och som kan innehålla tre filer. Dessa filer kan ha samma karakteristika som filer på vanliga disk drives. I applikationsprogram som definierar den diskdrive som skall användas kan jag använda denna RAM-disk - för det är det det har blivit nu. Det betyder att jag t ex i TI Writer kan spara text på RAM-disk och ha programskivan i min hittills enda disk drive. Tyvärr tillåter inte Disk Manager-modulen att jag definierar DSKx. Den kollar att jag endast använder DSK1. - DSK3..

En annan finess är att jag kan få se innehållet i RAM-disk med ett speciellt kommando, som då visar namn mm på de filer jag har där.

MEM-filerna kan i X-Basic hanteras med OPEN och CLOSE, med INPUT och PRINT, med EOF, RESTORE och DELETE. Det går att göra LOAD och SAVE.

I FORTH håller jag på och utvecklar program för MEM. Jag har gjort ett disk-kopieringsprogram, som kopierar en skiva på en enda genomgång med min enda disk drive. Jag håller på med sorteringsprogram och program för att kunna lagra overlay-program-strukturer i MEM och alltså kunna arbeta med mycket stora program och snabbt skifta overlay-moduler.

För dem som programmerar i Editor/Assembler rekommenderar fabrikanten att man sparar källkoden i en fil, objektkoden i en annan och listningen i den tredje. Tidsvinsten vid assembleringar blir betydande.

Under 1985 har ett ytterligare 128K-kort annonserats och presenterats i amerikank press. Det kommer från Myarc och tycks ha fler finesser. Bl a utnyttjas hela 96K-utrymmet för data - ja, det har t o m förekommit uppgifter om att 128K kan utnyttjas för data om man har ett vanligt 32K-kort i boxen samtidigt. Dessutom har utbyggnadsmöjligheter till 512K (") annonserats. Jag tror det moderna uttrycket är "Häftigt"" eller "Grymt"". Myarcs RAM-disk kan också adresseras så, att den kan ges en vanlig disk drive-adress, t ex DSK1. eller DSK2. . Man flyttar alltså den adressen från den vanliga disk drive till RAM-disken med ett kommando. Det ökar användningen för sådana program som fast adresserar DSK1., som är vanligt, eller som kontrollerar att DSK följs av en siffra. Här kan t ex vid användning av TI Writer programskivan kopieras till RAM-disk. TI FORTH sägs också vara helt kompatibelt med Myarcs RAM-disk, vilket betyder att man vid FORTH-programmering kan jobba utan de ganska tröttsamma diskaccesserna vid tester och omkompileringar t ex. Ytterligare en finess med Myarcs kort är att det kan användas som s k "print spooler". Det innebär att man kan skriva ut en listning - om den inte är alltför lång - till extra-minnet, varifrån det sedan förs över till printern i den takt som passar printern. Själva poängen är att överföringen från extra-minnet till printern sker på ett sådant sätt att datorn blir tillgänglig för annat arbete. Medan printern står och tuggar kan jag alltså jobba vidare vid tangentbordet"

Ytterligare ett 128K-kort annonseras på marknaden - från Morning Star Software - men det vet jag inget ytterligare om.

Men det är klart - allting har ett pris. Myarc vill ha 250 dollar för sitt kort i USA nu. Med direkt import och en dollarkurs på 8:00 kr och ett schablonpåslag för tull, moms och frakt på 30 procent blir det c:a 2 600:-. Mitt kort kostade mig litet under 2 000:- kr 1983. Foundation tar nu 180 dollar, vilket med samma beräkningar blir c:a 1900:- kr. Jag har sett annonser där postorderfirmor säljer Myarcs kort för 200 dollar - c:a 2 100:- kr.

Det tycks vara så att det är tillbehören som kostar i alla möjliga branscher och kanske särskilt i databranschen.

Låt mig avsluta med något fantasiegande: Michael Öhman hälsade på mig häromdagen och passade då på att kopiera DSR-programmet i chip-et till disk. Programmet finns förstås i maskinspråk i ROM-et och får disassembleras innan man kan arbeta med det. Det finns en disassemblator i föreningens FORTH, skriven av Björn Gustavsson förutom den som Tony gjort och andra. Ett nästa steg blir nu att analysera programmet och göra förbättringar - kanske de som finns i Myarcs kort, om det är möjligt. En enkel sak är t ex att byta ut alla texter till svenska. Andra förbättringar tycker jag förstås är angelägnare. Den viktigaste är att kunna ställa om RAM-disken till att bli DSK1. . Då uppkommer det verkliga lyftet" Hela operationen avslutas med man bränner ett nytt ROM med det förbättrade DSR-programmet i och petar in det i kortet.

■

DART-PROGRAM

av Lennart Thelander.

Jag är med i en DART-förening i skåne, där vi träffas och kastar litet ibland. Vi har ofta någon form av utslagsturnering när vi träffas. Det var ganska trist att bli utslagen i första omgången så jag köpte en dart-tavla för att träna hemma. Ganska snart kom jag på att det var tråkigt att bara stå och kasta ensam. Vad gör man då? Jo, man sätter sig ner och skriver ett litet datorprogram som fungerar som en motståndare för att göra kastningen litet mer intressant.

Detta program 'kastar' tre pilar och skriver ut resultatet på skärmen. Därefter matar jag in vad jag har kastat och så får jag en match mot datorn. I programmet har jag matat i hur en dart-tavla ser ut och också litet taktik dvs vad datorn skall sikta på.

Det kan kanske vara på sin plats att förklara reglerna för dart. Varje spelare börjar på samma poäng: 301, 501 eller t.o.m. 901. Vanligast är 501. Det gäller därefter att så snabbt som möjligt komma ner till precis 0. Den sista pilen måste dessutom vara en dubbel eller i mitten (ox-ögat, bulls eye eller bulle). Dubbel är den yttre smala ringen, trippel den inre smala ringen. Bullen är värd 50 poäng och ringen strax utanför är värd 25 poäng. Om du kastar fler poäng än vad du har kvar, får en enda poäng kvar eller inte går ut på dubbel eller bulle står du kvar på samma poäng som innan omgången. (Du kan ju inte gå ut på dubbel om du bara har en poäng kvar.)

Nog om detta. Nu övergår jag till programmet. Jag börjar med litet taktik. Det ställe på dart-tavlan som ger flest poäng är 3*20 (trippel 20). Det är också det stället som datorn siktar på när den har fler än 110 poäng kvar. När du kommit till lägre än 110 får du se till att kasta så du alltid lämnar ifrån sig ett jämnt antal poäng så att du kan gå ut på nästa (två) pil(ar). Det finns en tabell bland datasatserna som beskriver vad datorn siktar på vid olika poäng. Dessa data lagras i vektorn P. T.ex. P(69) beskriver vad datorn skall sikta på när den har 69 poäng kvar. Står det 19.1 betyder det 19, 19.2 dubbel 19 osv.

När datorn kastar har den en viss spridning på pilarna i både höjd- och sidled. Spridningen är ungefär normalfördelad med en viss standardavvikelse. Denna avvikelse matar du in när programmet startas. Du får prompten 'HÖJDSIGMA' resp. 'SIDOSIGMA'. Ett lagom värde för nybörjare är 4 till 5 ungefär. Om datorn blir för långt efter eller för långt före korrigeras siktet för att det skall bli mera jämspelt. När du matat in spridningen matar du in startpoängen och sitt eget namn. Datorn frågar vem som skall börja kasta och därefter börjar själva kastningen.

Överst på skärmen står båda spelarnas namn. Under det finns en lucka där datorns pilar skrivs in. I mitten finns en tabell över de drygt 10 senaste kastomgångarna. Längst ned finns en s.k. inmatningsprompt där du matar in bl.a. ditt resultat. Datorns kastresultat skrivs på följande sätt: 1*12 är en enkel tolva, 2*12 en dubbel 3*12 trippel och 0*12 en miss, men utanför sektorn där 12 befinner sig. Det enda som nu finns att tillägga är en del nyttiga variabler som programmet använder:

SCORED är datorns poäng
SCOREP är din poäng
RAD är den rad datorns resultat skirvs på
RAP är den rad ditt resultat skrivs på
MAXPOANG är det poängantal kastningen börjar på
HOJD är höjdspridningen i centimeter
SIDO är sidospridningen i centimeter
SIKTA är det ställe datorn siktar på

DART–PROGRAM

Programmets uppbyggnad:

```
100-130 Init, prescan, svenska tecken.
140-200 Dart-tavlans geometri beskrivs
210-280 Inmatning av data
290-320 Ritar skärmen
330-340 Initiera variabler
350     Vem kastar?
360-700 Datorn kastar
370-400 Korrigera siktet
470-570 Sikta och kasta 3 pilar
580     Skriv ut kastet
610-640 Datorn vann
660-690 Subrutin för att sikta
710-770 Data-satser
780-940 Du kastar
```

P.S. Eftersom detta program är skrivet efter den välkända 'terminal approach'-metoden är det ganska rörigt, men det fungerar. D.S.

P.P.S. För den som inte vet vad 'terminal approach'-metoden innebär ska jag förklara det här:
Du har en lös id' om vad ett program ska göra men inte HUR programmet ska se ut. Du låter emellertid inte detta faktum hindra dig från att börja skriva programmet. Programmet brukar av denna anledning bli ganska rörigt med GOTO hit och dit, en massa variabler du egentligen inte behöver samt allmänt klumpig lösning på programmeringsproblemet. D.D.S.

```
100 RANDOMIZE :: CALL CHAR(91,"00280038447C4444002800
    7C4444447C0038283847C4444"):: ON WARNING NEXT
110 DEF RNN(SIGMA)=(RND+RND+RND+RND+RND+RND+RND+RND+R
    ND+RND+RND+RND-6)*SIGMA
120 DIM P(110),T(20),Q(20):: CALL CLEAR
130 GOTO 140 :: A$,B$ :: CALL KEY :: CALL SOUND :: SU
    M,RAP,RAD,K,S,F,A,POANG,HOJD,SIDO,MAXPOANG,PO,SCO
    RED,SCOREP,X,Y,FI,R,SIKTA,I,NOLLA :: "'P-
140 CALL CLEAR
150 PRINT "VÄNTA ETT TAG": : : :
160 RESTORE
170 T(0)=11 :: T(1)=14 :: T(2)=9 :: T(3)=12 :: T(4)=5
     :: T(5)=20 :: T(6)=1 :: T(7)=18 :: T(8)=4 :: T(0
    9)=13 :: T(10)=6 :: T(11)=10 :: T(12)=15
180 T(13)=2 :: T(14)=17 :: T(15)=3 :: T(16)=19 :: T(1
    7)=7 :: T(18)=16 :: T(19)=8 :: T(20)=11
190 FOR I=0 TO 110 :: READ P(I):: NEXT I
200 FOR I=0 TO 20 :: READ Q(I):: Q(I)=Q(I)+90 :: NEXT
     I
210 INPUT "HÖJDSIGMA: ":SIDO :: INPUT "SIDOSIGMA: ":H
    OJD
220 INPUT "ANTAL POÄNG: ":MAXPOANG
230 PRINT :: INPUT "VAD HETER DU? ":A$
240 A$=SEG$(A$,1,14):: PRINT :: PRINT
250 IF B$<>"" THEN CALL CLEAR
260 PRINT "HÖJDSIGMA:";HOJD:"SIDOSIGMA:";SIDO :: PRIN
    T : : : :"VILL DU BÖRJA (J/N)?"
270 PRINT
280 ACCEPT AT(22,22)SIZE(1)VALIDATE("JN"):B$
290 CALL CLEAR
300 DISPLAY AT(1,1):"TEXAS",A$
310 DISPLAY AT(8,8)SIZE(4):USING "ÄÄÄÄ":MAXPOANG
320 DISPLAY AT(8,24)SIZE(4):USING "ÄÄÄÄ":MAXPOANG
330 RAP,RAD=2
340 SCOREP,SCORED=MAXPOANG
350 IF B$="N" THEN B$="J" :: GOTO 360 ELSE B$="N" ::
    GOTO 780
360 FOR I=3 TO 6 :: DISPLAY AT(I,1)SIZE(15):"" :: NEX
    T I
370 IF SCORED>SCOREP+100 THEN HOJD=HOJD*.85 :: SIDO=S
    IDO*.85
380 IF SCORED>SCOREP+150 THEN HOJD=HOJD*.85 :: SIDO=S
    IDO*.85
390 IF SCOREP>SCORED+150 THEN HOJD=HOJD/.85 :: SIDO=S
    IDO/.85
400 IF SCOREP>SCORED+100 THEN HOJD=HOJD/.85 :: SIDO=S
    IDO/.85
410 POANG=SCORED
420 SUM=0
430 RAD=RAD+1
440 IF RAD=15 THEN DISPLAY AT(7,1):RPT$(" ",32*3)
450 IF RAD=16 THEN RAD=RAD-14
460 DISPLAY AT(RAD+8,1)SIZE(32):""
470 FOR I=1 TO 3
480 IF POANG>110 THEN SIKTA=20.3 ELSE SIKTA=P(POANG)
490 GOSUB 660 :: X=X+RNN(HOJD):: Y=Y+RNN(SIDO)
500 R=SQR(X*X+Y*Y):: FI=ATN(X/Y):: IF Y<0 THEN FI=FI+
    PI
510 FI=(FI+PI/2)/2/PI*20
520 IF R<.75 THEN PO=50 ELSE IF R<1.75 THEN PO=25 ELS
    E IF R<9.65 THEN F=1 ELSE IF R<10.65 THEN F=3 ELS
    E IF R<16 THEN F=1 ELSE IF R<17 THEN F=2 ELSE F=0
530 IF R>1.75 THEN PO=T(FI)*F
540 CALL SOUND(10,-7,5):: POANG=POANG-PO :: IF PO=50
    OR PO=25 THEN DISPLAY AT(I+2,2):PO ELSE DISPLAY A
    T(I+2,2):F;"*";T(FI)
550 IF POANG=0 AND(F=2 OR PO=50)THEN 610 ELSE IF POAN
    G<2 THEN DISPLAY AT(6,2):"TJOCK" :: DISPLAY AT(RA
    D+6,3)SIZE(3):"---" :: DISPLAY AT(RAD+6,8)SIZE(4)
    :USING "ÄÄÄÄ":SCORED :: GOTO 600
560 SUM=SUM+PO
570 NEXT I
580 DISPLAY AT(RAD+6,3)SIZE(3):USING "ÄÄÄ":SUM :: DIS
    PLAY AT(RAD+6,8)SIZE(4):USING "ÄÄÄÄ":POANG
590 SCORED=POANG
600 GOTO 350
610 DISPLAY AT(RAD+6,3)SIZE(3):USING "ÄÄÄ":SCORED ::
    DISPLAY AT(RAD+6,8)SIZE(4):USING "ÄÄÄÄ":NOLLA
620 DISPLAY AT(24,1):"JAG BESEGRADE DIG, KOM IGEN"
630 DISPLAY AT(23,1):"TRYCK PÅ NÅGON TANGENT"
640 CALL KEY(0,K,S):: IF S=0 THEN 640 ELSE CALL CLEAR
     :: GOTO 240
650 STOP
660 IF SIKTA=50 THEN 690 ELSE A=Q(SIKTA):: X=-COS(A/1
    80*PI):: Y=SIN(A/180*PI):: A=SIKTA-INT(SIKTA)
670 IF A=.3 THEN X=X*10.15 :: Y=Y*10.15 ELSE IF A=.2
    THEN X=X*16.5 :: Y=Y*16.5 ELSE X=X*13.5 :: Y=Y*13
    .5
680 RETURN
690 X=0 :: Y=0 :: RETURN
700 STOP
710 DATA 0,0,1.2,1.0,2.2,1.0,3.2,3.0,4.2,1.0,5.2,3.0,
    6.2,5.0,7.2,7.0,8.2,1.0,9.2,3.0,10.2
720 DATA 1.0,11.2,3.0,12.2,5.0,13.2,3.0,14.2,5.0,15.2
    ,7.0,16.2,1.0,17.2,3.0,18.2,5.0,19.2,7.0,20.2
730 DATA 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,50,11.0,
    12.0,13.0,14,15,16,17,18,19,20
740 DATA 7.3,10.3,9.3,8.3,11.3,10.3,9.3,12.3,11.3,10.
    3,13.3,12.3,11.3,14.3,13.3,12.3,15.3,14.3,13.3,16
    .3
750 DATA 15.3,14.3,17.3,16.3,15.3,18.3,17.3,16.3,19.3
    ,18.3,17.3,20.3,19.3,18.3,19.3,20.3,19.3,20.3,19.
    3,20.3
760 DATA 17.3,20.3,20.3,18.3,20.3,20.3,19.3,20.3,20.3
    ,20.3
770 DATA 0,18,144,180,54,-18,90,216,252,-54,108,270,-
    36,72,288,126,234,162,36,198,0,DUMMY
780 DISPLAY AT(6,17)SIZE(12):""
790 DISPLAY AT(24,1):"DIN TUR.  RESULTAT:"
800 ACCEPT AT(24,22):PO
810 PO=INT(ABS(PO))
820 IF PO>180 THEN DISPLAY AT(24,1):"DU KAN INTE FÅ M
    ER ÄN 180" :: CALL SOUND(100,200,0)
830 IF PO<=180 THEN 850
840 FOR I=1 TO 1000 :: NEXT I :: GOTO 780
850 RAP=RAP+1 :: IF SCOREP-PO=0 THEN DISPLAY AT(RAP+6
    ,18)SIZE(3):USING "ÄÄÄ":SCOREP :: DISPLAY AT(RAP+
    6,24)SIZE(4):USING "ÄÄÄÄ":NOLLA :: SCOREP=0 :: GO
    TO 870
860 IF SCOREP-PO<2 THEN DISPLAY AT(6,17):"TJOCK" :: D
    ISPLAY AT(RAP+6,18)SIZE(3):"---" :: DISPLAY AT(RA
    P+6,24)SIZE(4):USING "ÄÄÄÄ":SCOREP
870 IF SCOREP-PO>1 THEN SCOREP=SCOREP-PO :: DISPLAY A
    T(RAP+6,18)SIZE(3):USING "ÄÄÄ":PO :: DISPLAY AT(R
    AP+6,24)SIZE(4):USING "ÄÄÄÄ":SCOREP
880 IF SCOREP=0 THEN DISPLAY AT(24,1):"GRATTIS"  DU V
    ANN"
890 IF SCOREP<>0 THEN 920
900 DISPLAY AT(23,1):"TRYCK PÅ NÅGON TANGENT"
910 CALL KEY(0,K,S):: IF S=0 THEN 910 ELSE CALL CLEAR
     :: SCORED,SCOREP=MAXPOANG :: GOTO 240
920 DISPLAY AT(24,1):"MIN TUR ATT KASTA"
930 IF RAP=15 THEN RAP=1
940 GOTO 350
```

MINNESUTRYMME

av Jan Alexandersson

Tillgängligt minnesutrymme i TI-99/4A för program varierar mycket beroende på viken modul och vilken kringutrustning som används. Följande tabell visar läget uttryckt i bytes:

	program	stack
BA	14536	-
XB	13928	-
XB + EM	(24488)*	13928
BA + DS	12448	-
XB + DS	11840	-
XB + DS + EM	24488	11840
BA + DS + CF(1)	13480	-
XB + DS + EM + CF(1)	24488	12876
SAVE MINIMEM	4088	-

Förkortningar:
BA = grund-BASIC
XB = Extended BASIC
EM = Expansionsminne 32k
DS = Disk drive
CF = CALL FILES

FÖR XB + EM bestämmer stack-utrymmet hur långa program som kan sparas på kassett. Detta problem med kassettlagring finns även tillsammans med flexskiveenhet. Du kan inte göra reservkopior av långa program på kassett. Om du bara använder flexskiva går det bra att ha 24488 bytes långa program. I praktiken kan XB ha lika långt program som summan av dina flexskiveenheter eftersom varje delprogram kan ladda in och köra nästa del med RUN "DSK1.NAMN".

Som du kanske ser så fattas 8 kbytes vid XB + EM. Denna minnesdel används endast för Assembler-program. Grund-BASIC kan inte använda EM även om den ansluts till maskinen.

SIZE I BASIC

I Extended BASIC finns ett kommando SIZE som säger hur mycket fritt utrymme som finns i maskinen. För att göra detta i BASIC måste man använda ett knep. I Nittinian 83-2 fanns följane kortis:

```
1 M=M+8
2 GOSUB 1
```

Detta placeras först i programmet som som sedan körs tills det stannar på grund av MEMORY FULL. PRINT M ger då nästan svaret. Eftersom programet körs kommer alla variabler att tilldelas minnesutrymme vid "pre scan" som alltid görs innan programmet löper.

Ett sätt att testa minnet utan att köra programmet är med DIM A(X). Testa med olika värden på X tills du får MEMORY FULL. Kom bara ihåg att editera med t.ex. 1 (ENTER) innan DIM(X) så att inga andra variabler ockuprerar minnet.

Ett tredje sätt är att använda MINI MEMORY eller EDITOR/ASSEMBLER I BASIC:

```
CALL PEEK(-31974,A,B)
PRINT 256*A+B-1817
```

I Extended BASIC utan expansionsminne kan du använda samma teknik om man skriver PRINT 256*A+B-2413.

LADDA LÅNGA PROGRAM FRÅN KASSETT

Om du har ett mycket långt program på kassett som skall föras över till flexskiva kan det bli problem.
Första åtgärden är att skriva CALL FILES(1) följt av NEW. Om detta inte räcker för XB kan du ladda in det i BASIC. Du kan inte köra programmet, men OLD och SAVE fungerar.
Om du har ännu längre program måste du korta av programmet och lagra tillbaka detta på kassettband. Tag bort alla onödiga REM-satser och skriv flera PRINT-satser på samma rad med kolon(:). Om detta inte hjälper måste programmet delas i två delar och sedan smältas ihop med MERGE.

Även om du fått in programmet på flexskivan återstår ett problem. Du är tvungen att alltid skriva CALL FILES(1) varje gång du använder det eftersom det lagras i PROGRAM-format. I Extended BASIC bör du göra om långa program till INT/VAR 254-format. Gör så här:

```
CALL FILES(1)
NEW
OLD CS1
SAVE DSK1.NAMN,MERGE
CALL FILES(3)
NEW
MERGE DSK1.NAMN
SAVE DSK1.NAMN
```

Det finns dock en liten skilnad på SAVE och OLD(med CALL FILES(3)). Maximal längd för SAVE är 11840 bytes som väntat medan OLD klarar 11872 bytes med PROGRAM-format.

ANNONS

Det kom ett brev från Polen. En nybliven 99-ägare vill ha kontakter och köpa prylar. Han är särskilt intresserad av

- RS232

Han har nog bara själva grundmaskinen och talar också om expansionsbox, printer mm. Han vill gärna köpa begagnat.

Du som vill sälja - tag kontakt med

Andrzej Pusz
UL. Zwyciestwa 57/13
44-100 GLIWICE

Poland

Skriv på engelska" (genom Lennart Lindberg)

RITNINGAR

Vem har tillgång till ritningar till

- RS232
- Disk Controller
- Minnesexpansion 32K
- Expansionsboxens bakplan

Om någon har erfarenheter av bygge att dela med sig av är det ännu bättre.

Per-Ove Andersson
Rörverksgatan 88
724 74 Västerås

Tfn 021-35 24 08.

SÄLJES

Spelmodul	Blasto	100:-
Spelmodul	Munch Man	100:-

Enkelsidig diskdrive, TI-orginal, för inbyggnad i expansionsboxen 700:-

Ev frakt och postförskottsavgift tillkommer.
Börje Häll
0758-174 24

SÄLJES

Programming Aids I	3 st	- 75 SEK per st.
Household Budget Management	2 st	- 115 SEK per st.
Personal Record Keeping	1 st	- 155 SEK.

Säljes mot postförskott (postförskottsavgift tillkommer)

Niklas Mårtensson
Makrillvägen 1
231 00 Trelleborg

TI 59

Roten ur ...

Kvadratrot med 410 eller 420 siffrors precision

Med risk att bli tjatig återkommer jag till detta ämne, men helt mitt fel är det inte.
I början av Augusti fick jag ett brev från Holland, alltså Robert Prins. Han hade en algoritm för uträkning av SQR.
Han hade försökt att själv få ihop ett fungerande program men..

Algoritm för uträkning av kvadratrot Q

Approximering = A	B = 5 * Q - A	
0	50	roten ur 10
5	45	A sätts till 5
15	30	A = A + 10
25	5	
35	-30	B är neg. 3 är första siffran
305	500	A = 10A - 45, B = B * 100
305	195	
315	-120	31 klara
3105	19500	
3105	16395	
3115	13280	
3125	10155	
3135	7020	
3145	3875	
3155	720	
3165	-2445	316 klara

osv.

Jag ändrade lite på uppställningen och div med 5 samt införde en räknare R

R	A	B = B - A	
0	1	10	A = 1
1	3	9	A = A + 2
2	5	6	
3	7	1	A > B 3 är 1:a
0	61	100	R = 0, A = 10A - 9, B = 100B
1	63	39	A är större än B

Och så vidare

När jag kommit hit skrev jag ett program:

```
LBL A CMS STO 0 OP 21 RCL 1 x:t RCL 0 INV GE 028
x:t INV SUM 00 2 SUM 1 OP 22 GTO 007 0 EXC 2 PRT
10 PRD 1 X^2 PRD 00 9 INV SUM 1 GTO 007.
```

Körning med detta program t.ex. tryck 2 A ger 25 riktiga siffror på roten ur 2 med enbart 3 minnen.
För att stanna programmet tryck R/S.

Nu när "principen var given" gjorde jag ett grovt fungerande program som trots Fast Mode hade en exekveringstid på ca 24h. Detta sände jag över till Holland för frisering (mitt program räknade ut 400 siffror).
Och i slutet av Augusti kom Roberts variant på mitt program med exekveringstid ca 11h, fantastiskt elegant men framför allt uppsnabbat.
Efter att ha studerat Prins program tog jag mig friheten att ändra lite grann i hans program för att få ut 410 siffror.
Programmet bifogas; Program 1:

Bruksanvisning, uträkning roten ur Q
Mata in Q (1 SE Q < 100 (SE = smaller or equal))
tryck A, programmet stannar, mata in önskad precision N_1 (N_1 * 10 = precision, antalet siffror N_1 får max vara 41) tryck C (eller R/S), när 1.10 blinkar tryck 7 EE och programmet startar.
Om du efter utförd exekvering vill fortsätta, mata in N_2 ($N_1 + N_2$ får max vara 41) tryck C (= Continue) när 1.10 blinkar tryck 7 EE.

Program 2 är samma program som program 1 men R6 är utbytt mot R4 se steg 124-129 i program 2.
Dessa 6 steg gör program 2 något långsammare.
Bruksanvisning Program 2 se Program 1 bruksanvisning, men byt ut 41 mot 42 vid precisionen.
Exempel på utskrift bifogas SQR(5) med 420 siffror.

Robert Prins har kontrollerat att programmet är riktigt med följabde X-BASIC-program.

```
100  DEF INT A-B
110  INPUT N
120  DIM A(N),B(2*N)
130  FOR I=N TO 1 STEP -1
140  READ A(I)
150  NEXT I
160  FOR I=1 TO N
170  IF A(I)=0 THEN 220
180  P=I
190  FOR J=1 TO N
200  B(P)=B(P)+A(I)*A(J): P=P+1
210  NEXT
220  NEXT
230  FOR I=1 TO 2*N-1
240  H=INT(B(I)/10): B(I)=B(I)-10*H:B(I+1)=B(I+1)+H
250  NEXT
260  P=2*N
270  FOR I=1 TO N
280  H=B(I):B(I)=B(P):P=P-1
290  NEXT
300  IF B(I)=0 THEN P=2 ELSE P=1
310  FOR I=P TO N*2
320  PRINT USING "#";B(I);
330  NEXT
340  DATA 2,2,3,6,0,6,7,9,7,7,4,9,9,7,8,9,6,9,6,
          4,0,9,1,7,3,6,6,8,7,3
```

(Rad 340 de trettio första siffrorna i SQR(5) övriga siffror får Nä själva fylla i)
Detta program kvadrerar dessa siffror (antalet matas in som N på rad 110 (t.ex 420)

Sven-Arne Wallin
N. Parkg 5
343 00 Älmhult

TI 59

PROGRAM 1

```
000  92 RTN
001  97 DSZ
002  00  00
003  00  00
004  24  24
005  24 CE
006  82 HIR
007  17  17
008  99 PRT
009  01   1
010  00   0
011  48 EXC
012  00  00
013  82 HIR
014  07  07
015  01   1
016  82 HIR
017  36  36
018  97 DSZ
019  02  02
020  00  00
021  24  24
022  00   0
023  81 RST
024  43 RCL
025  01  01
026  42 STO
027  03  03
028  01   1
029  00   0
030  82 HIR
031  47  47
032  64 PD*
033  05  05
034  69 DP
035  25  25
036  97 DSZ
037  03  03
038  00  00
039  32  32
040  33 X²
041  64 PD*
042  06  06
043  69 DP
044  26  26
045  97 DSZ
046  01  01
047  00  00
048  41  41
049  02   2
050  42 STO
051  01  01
052  82 HIR
053  16  16
054  48 EXC
055  03  03
056  29 CP
057  22 INV
058  74 SM*
059  06  06
060  69 DP
061  36  36
062  32 X↕T
063  74 SM*
064  06  06
065  73 RC*
066  06  06
067  55  ÷
068  82 HIR
069  18  18
070  54  )
071  59 INT
072  65  ×
073  32 X↕T
074  82 HIR
075  18  18
076  54  )
077  97 DSZ
078  03  03
079  00  00
080  57  57
081  04   4
082  09   9
083  42 STO
084  06  06
085  97 DSZ
086  01  01
087  00  00
088  52  52
089  09   9
090  22 INV
091  44 SUM
092  48  48
093  04   4
094  09   9
095  75  -
096  82 HIR
097  16  16
098  42 STO
099  01  01
100  85  +
101  42 STO
102  03  03
103  42 STO
104  05  05
105  04   4
106  01   1
107  54  )
108  42 STO
109  04  04
110  42 STO
111  06  06
112  73 RC*
113  03  03
114  69 DP
115  23  23
116  32 X↕T
117  73 RC*
118  04  04
119  69 DP
120  24  24
121  67 EQ
122  01  01
123  12  12
124  22 INV
125  77 GE
126  00  00
127  01  01
128  29 CP
129  73 RC*
130  05  05
131  22 INV
132  74 SM*
133  06  06
134  73 RC*
135  06  06
136  77 GE
137  01  01
138  51  51
139  82 HIR
140  18  18
141  74 SM*
142  06  06
143  69 DP
144  36  36
145  01   1
146  22 INV
147  74 SM*
148  06  06
149  44 SUM
150  06  06
151  69 DP
152  25  25
153  69 DP
154  26  26
155  97 DSZ
156  01  01
157  01  01
158  29  29
159  01   1
160  82 HIR
161  37  37
162  02   2
163  61 GTO
164  00  00
165  91  91
```

PROGRAM 2

```
000  92 RTN
001  97 DSZ
002  00  00
003  00  00
004  24  24
005  24 CE
006  82 HIR
007  17  17
008  99 PRT
009  01   1
010  00   0
011  48 EXC
012  00  00
013  82 HIR
014  07  07
015  01   1
016  82 HIR
017  36  36
018  97 DSZ
019  02  02
020  00  00
021  24  24
022  00   0
023  81 RST
024  43 RCL
025  01  01
026  42 STO
027  03  03
028  01   1
029  00   0
030  82 HIR
031  47  47
032  64 PD*
033  05  05
034  69 DP
035  25  25
036  97 DSZ
037  03  03
038  00  00
039  32  32
040  33 X²
041  64 PD*
042  04  04
043  69 DP
044  24  24
045  97 DSZ
046  01  01
047  00  00
048  41  41
049  02   2
050  42 STO
051  01  01
052  82 HIR
053  16  16
054  48 EXC
055  03  03
056  29 CP
057  22 INV
058  74 SM*
059  04  04
060  69 DP
061  34  34
062  32 X↕T
063  74 SM*
064  04  04
065  73 RC*
066  04  04
067  55  ÷
068  82 HIR
069  18  18
070  54  )
071  59 INT
072  65  ×
073  32 X↕T
074  82 HIR
075  18  18
076  54  )
077  97 DSZ
078  03  03
079  00  00
080  57  57
081  04   4
082  08   8
083  42 STO
084  04  04
085  97 DSZ
086  01  01
087  00  00
088  52  52
089  09   9
090  22 INV
091  44 SUM
092  47  47
093  04   4
094  08   8
095  75  -
096  82 HIR
097  16  16
098  42 STO
099  01  01
100  85  +
101  42 STO
102  03  03
103  42 STO
104  05  05
105  04   4
106  02   2
107  54  )
108  42 STO
109  04  04
110  82 HIR
111  05  05
112  73 RC*
113  03  03
114  69 DP
115  23  23
116  32 X↕T
117  73 RC*
118  04  04
119  69 DP
120  24  24
121  67 EQ
122  01  01
123  12  12
124  48 EXC
125  04  04
126  82 HIR
127  15  15
128  48 EXC
129  04  04
130  22 INV
131  77 GE
132  00  00
133  01  01
134  29 CP
135  73 RC*
136  05  05
137  22 INV
138  74 SM*
139  04  04
140  73 RC*
141  04  04
142  77 GE
143  01  01
144  57  57
145  82 HIR
146  18  18
147  74 SM*
148  04  04
149  69 DP
150  34  34
151  01   1
152  22 INV
153  74 SM*
154  04  04
155  44 SUM
156  04  04
157  69 DP
158  24  24
159  69 DP
160  25  25
161  97 DSZ
162  01  01
163  01  01
164  35  35
165  01   1
166  82 HIR
167  37  37
168  02   2
169  61 GTO
170  00  00
171  91 R/S
172  00   0
193  76 LBL
194  11  A
195  32 X↕T
196  69 DP
197  00  00
198  69 DP
199  05  05
200  09   9
201  69 DP
202  17  17
203  47 CMS
204  32 X↕T
205  42 STO
206  89  89
207  99 PRT
208  69 DP
209  05  05
210  01   1
211  82 HIR
212  06  06
213  42 STO
214  48  48
215  08   8
216  93  .
217  07   7
218  85  +
219  00   0
220  92 RTN
221  76 LBL
222  13  C
223  42 STO
224  02  02
225  02   2
226  93  .
227  07   7
228  02   2
229  85  +
230  01   1
231  00   0
232  42 STO
233  00  00
234  22 INV
235  28 LOG
236  82 HIR
237  08  08
238  54  )
239  86 STF
```

```
220  92 RTN
221  76 LBL
222  13  C
223  42 STO
224  02  02
225  02   2
226  93  .
227  07   7
228  02   2
229  85  +
230  01   1
231  00   0
232  42 STO
233  00  00
234  22 INV
235  28 LOG
236  82 HIR
237  08  08
238  54  )
239  86 STF
```

5.

```
2236067977.
4997896964.
 917366873.
1276235440.
6183596115.
2572427089.
7245410520.
9256378048.
9941441440.
8378782274.
9695081761.
5077378350.
4253267724.
4470738635.
8636012153.
3452708866.
7781731918.
7916581127.
6645322639.
8565805357.
6135041753.
3785003423.
3924140644.
4208643253.
9097252592.
6272288762.
9951740244.
 681611775.
9089094984.
9237139072.
9728898482.
 886415426.
8989409913.
1693577019.
7486788844.
2508975413.
2956183176.
9214999774.
2480153043.
4115035957.
6683325124.
9881517813.
```

FORTH spalten

REDAKTÖR: Lars-Erik Svahn
Mellanbergsv. 23
13545 TYRESÖ
tel: 08-742 61 07

FORTH-ordet 'fetch' (ASCII-kod 64 decimalt) skrivs normalt ut som 'a-slang' eller 'bulle'. Här blir det ett paragraf-tecken - §.

Både TI-FORTH och PB-FORTH är utvidgningar av fig-FORTH, den dialekt av FORTH som tagits fram av FORTH Interest Group i USA. Detta betyder att kärnan i 99:ans olika FORTH-system är en vida spridd standard-dialekt av FORTH.
Källkod skriven enbart med ord i FORTH-kärnan (ord med lägre parameterfälts-adress än ordet SYSTEM) bör därför kunna kompileras med fig-FORTH-system på andra datorer utan att några problem uppstår - i varje fall om källkoden är skriven med en smula eftertanke, och inte utnyttjar sådant som är speciellt för 99:an.
I förra spalten visade jag några ord som troligen är portabla, dvs kan användas på andra fig-FORTH-system (jag kan dock inte garantera detta eftersom jag inte provat på något annat system, men rutinerna är skrivna enbart med fig-kärnan och med viss eftertanke!)
De rutiner kag presenterade gjorde det möjligt att kompilera ord utan programhuvud. Sådana huvudlösa ord kan sedan inte anropas från tangent bordet. De fungerar ungefär som procedurer och subrutiner gör i andra språk: de kan bara anropas från huvudrutinen.
De två fördelarna med att använda denna metod är att Du spar minne och att sökningen i ordlistan blir snabbare.

I förra spalten presenterade jag en del FORTH-begrepp och en del FORTH-ord. Jag fortsätter här denna presentation.

Mer om ordlistan

Ordlistan ligger i TI (och PB) FORTH på högt minne (hex A000-FFFE).
Dina FORTH-ord kompileras in i ordlistan.
Den kompilerade koden består av två delar: programhuvudet och programkroppen. I kroppen finns själva programmet, och i huvudet finns namnet och en del nödvändiga adresser lagrade.
Alla huvuden konstrueras efter en standard mall:

```
     ----------------
lfa  länkfält 2 byte  (Link Field Address)
     ----------------
nfa  namnfält         (Name Field Address)
     2 - 32 byte
     ----------------
cfa  kodfält  2 byte  (Code Field Address)
     ----------------
pfa  kropp            (Parameter Field Address)
```

I förra spalten beskrev jag vad de olika fälten i huvudet hade för funktion. Det återstår nu bara att beskriva några detaljer i namnfältet:
Namnfältet består på 99-an alltid av ett jämnt antal bytes. Alla dessa bytes utom den första (och ibland den sista), innehåller själva namnet i form av ASCII-koder.
I den första byten i namnfältet finns antalet tecken i namnet lagrat. Eftersom ett FORTH-namn får vara högst 31 bytes långt upptar detta tal endast de 5 minst signifikanta bitarna i denna byte. De 3 mest signifikanta bitarna används som flaggor på följande sätt:
Bit 7 (den mest signifikanta biten) markerar början av namnfältet. Ordet TRAVERSE (tex i ordet NFA), kan söka sig igenom namnfältet för att finna nfa, med hjälp av denna flagga.
Bit 6 är ettställd om ordet är ett IMMEDIATE-ord (se nedan).
Bit 5 ettställes resp nollställes omväxlande av ordet SMUDGE. För att ordet ska kunna hittas (tex av FORGET, -FIND, INTERPRET eller andra ord som letar i ordlistan) måste bit 5 vara nollställd. SMUDGE finns i definitionerna för ':' och ';' . Först ogiltig-förklarar ':' det just bildade ordet, och till slut (om inga fel upptäcks!) så görs definitionen giltig av ';'. (Jag uppfattar detta vara litet irriterande, eftersom jag då inte kan använda FORGET direkt på det felaktiga ordet. Istället kan jag förstås skriva SMUDGE FORGET 'ordet').

Den sista byten i namnfältet har den mest signifikanta biten ettställd av samma skäl som den första byten har det. TRAVERSE kan nämligen användas att söka åt båda hållen!

TOGGLE (adr b --) är det ord som används för att växla flaggorna ovan. Byten b jämförs med hjälp av XOR med den byte som finns på adressen adr.
Med TOGGLE definieras (HEX gäller):

```
: SMUDGE LATEST 20 TOGGLE ;  och

: IMMEDIATE LATEST 40 TOGGLE ;
```

Indataflödet och det virtuella minnet

Indataflödet (eng: input stream), är den text som matas till datorn antingen från tangentbordet eller från det virtuella minnet (vanligen skärmar på disk). Här följer några indata-ord och deras funktion:

BLK (-- adr) är en variabel som innehåller numret på det block (i TI FORTH lika med skärm) som har indata. BLK är 0 om indataflödet ska komma från tangentbordet.

IN (-- adr) är en variabel som anger hur stor del av indataflödet som har lästs. IN är 0 från start och justeras av vissa indataord. (I andra dialekter - t ex FORTH -79 och polyFORTH - heter denna variabel >IN).

TIB (-- adr) är en user-variabel som innehåller adressen till indatabufferten. (adderas innehållen i TIB och IN fås adressen till nästa tecken i indataflödet om detta kommer från tangentbordet).

QUERY (--) accepterar indata från tangentbordet (max 64 tecken) till indatabufferten (på adress TIB).

WORD (b --) läser text från indataflödet (från adress TIB + IN om BLK=0. Om BLK<>0 hämtas data från det block som markeras av innehållet i BLK och där från den position som markeras av IN) tills en byte med värdet b påträffas. Den lästa texten (utom tecknet med ASCII-koden b) lagras på adressen HERE som en sträng med antalet tecken i första byten (kallas dimensionerad sträng i FORTH).

INTERPRET (--) tolkar text från indataflödet (se WORD).

NUMBER (adr -- d) omvandlar en dimensionerad sträng (med antal tecken i första byten) på adressen adr till ett dubbelt heltal med tecken (i den talbas som markeras av det tal som finns lagrat i variabeln BASE). NUMBER kan användas med QUERY och WORD för inmatning av tal (jämför INPUT i Basic). Om den sträng som skall omvandlas inte är ett tal uppkommer ett fel.

Några exempel:

```
: GN ( -- d )
     BL WORD HERE NUMBER ;
```
Hämtar ett tal från indataflödet, och lägger det som ett dubbelt heltal på stacken

```
: (INPUT") SPACE QUERY GN DROP ;
: INPUT" ( INPUT" sträng" -- n )
     ÄCOMPILEÅ ."
     COMPILE (INPUT") ; IMMEDIATE
```
Skriver prompt-text, tar emot indata som en sträng, förvandlar denna till ett tal och lägger detta som ett heltal på stacken

```
: GETREC ( rad skärm -- )
     BLK ! Ö lagrar skärmnummer
     6 SLA Ö 64 *, men snabbare
     IN !  Ö första tecknet som ska läsas
     ASCII " WORD     Ö läser till HERE
     0 BLK ! 0 IN ! ; Ö återställer BLK och IN
```
Hämtar en post på specifierat rad- och skärmnummer och placerar denna på adress HERE. Posten skall på skärmen avslutas med " .

```
: TEST ." skriv två tal (space emellan)"
       QUERY GN DROP GN DROP ;
```
Lägger de två talen på stacken

```
: TEST2 ." exekvera FORTH-ord:" CR
        QUERY INTERPRET ;
```
Gör ett avbrott i det program där ordet förekommer för att interpretera godtyckliga FORTH-ord

```
: TEST3 INPUT" Postnummer:" CR
        ." numret är: " . ;
```
Visar hur INPUT" kan användas

Att utvidga kompilatorn!

En fascinerande och ovanlig möjlighet med FORTH, är att kunna utvidga språket. FORTH är utbyggbart i ordets rätta bemärkelse. Man kan inte bara namnge procedurer, utan också bestämma syntaxen, helt fritt! Detta är möjligt därför att det finns så många ord för att påverka kompileringen.

Ett sådant ord är IMMEDIATE. Detta ord ettställer en flagga på den adress som ges av LATEST (namnfältsadressen i det sist kompilerade programhuvudet; se ovan). Skriver man IMMEDIATE direkt efter ';' när man definierar ett ord kommer kompilatorn att uppfatta detta ord på ett speciellt sätt:

> normalt vid kompilering av en kolondefinition lagras varje ingående ords kodfältsadress i tur och ordning i parameterfältet för det ord som definieras

> ett ord som är märkt som IMMEDIATE (immediate-ord), får däremot inte sin kodfältsadress lagrad! Istället börjar detta ord att exekveras - mitt under kompileringen! Immediate-ord konstrueras för att de ska exekveras varhelst de påträffas, istället för att kompileras. I immediate-ord ser Du därför ofta ord som: COMPILE , ÄCOMPILEÅ , ALLOT m fl - dvs ord som påverkar ordlistan (de "tar över" från kompilatorn, tillfälligt).

Några viktiga definitioner:

COMPILE (COMPILE 'ord' --), lagrar kodfältsadressen för 'ord' på adressen angiven av variabeln DP (dictionary poiter), samt ökar DP med 2 (det är innehållet i DP som läggs på stacken av ordet HERE).

ÄCOMPILEÅ (ÄCOMPILEÅ 'immediate-ord' --), fungerar som COMPILE - men för immediate-ord! Det enda sättet att kompilera immediate-ord!

ALLOT (n --) reserverar minne i ordlistan, dvs ökar DP med talet överst på stacken.

, (n --) lagrar det tal som finns överst på stacken, på adressen som finns lagrad i DP. DP ökas med 2.

BRANCH får programmet att hoppa så många bytes, som anges av det tal som finns lagrat på adressen direkt efter i ordlistan (alltså: först står kodfältsadressen för BRANCH, och sedan kommer hoppdata).

OBRANCH (f --) fungerar som BRANCH, men hopp sker bara om översta talet på stacken = 0. Annars fortsätter exekveringen 4 bytes längre ned!

?COMP kontrollerar att STATE<>0. Om STATE=0 stoppas exekveringen med felmeddelandet "in definitions only!"

?PAIRS kontrollerar att de två översta talen på stacken är lika! Annars stoppas programmet med felmeddelandet "conditionals not paired!"

Exempel på "syntax-ord" som är skrivna i FORTH är:

```
: IF ( f -- ) ( komp: -- adr 2 ) ?COMP
     COMPILE OBRANCH HERE 0 , 2 ; IMMEDIATE
: ENDIF ( -- ) ( komp: adr f -- )
     ?COMP 2 ?PAIRS HERE OVER - SWAP ! ; IMMEDIATE
: THEN  ( -- ) ( komp: adr f -- )
     ÄCOMPILEÅ ENDIF ; IMMEDIATE
: ELSE  ( -- ) ( komp: adr1 f -- adr2 2 )
     ?COMP 2 ?PAIRS COMPILE BRANCH HERE 0 ,
     SWAP 2 ÄCOMPILEÅ ENDIF 2 ; IMMEDIATE
```

Förklaringar:

```
: IF ?COMP COMPILE OBRANCH
     HERE  Ö adressen för hoppdata
     0 ,   Ö minne reserveras för hoppdata
     2     Ö ett meddelande till ELSE och THEN
; IMMEDIATE

: ENDIF ?COMP
     2 ?PAIRS    Ö läser meddelandet från IF/ELSE
     HERE OVER - Ö beräknar hoppdata
     SWAP !      Ö lagrar hoppdata åt IF/ELSE
; IMMEDIATE
```

THEN är ett makro-synonym till ENDIF.
Exakt samma sekvens kompileras, om du
använder THEN som om du använder ENDIF

```
: ELSE ?COMP 2 ?PAIRS Ö läser meddelande från IF
     COMPILE BRANCH
     HERE  Ö adressen för hoppdata
     0 ,   Ö minne reserveras för hoppdata
     SWAP
     2
     ÄCOMPILEÅ ENDIF Ö makro-kopia av ENDIF
                     Ö som skriver hoppdata åt IF
     2               Ö meddelande till ENDIF
; IMMEDIATE
```

STATE (-- adr) är en variabel som är nollställd vid interpretering, och har ett värde<>noll vid kompilering (i TI FORTH är det värdet decimalt 192=HEX C0). Med hjälp av denna variabel kan man konstruera ord som exekverar olika, beroende på om de anropas i indataströmmen (tangentbordet eller en skärm) eller finns med i definitionen av ett annat ord. ." är ett sådant exempel:

```
: (.") R COUNT DUP 1+ =CELLS R> TYPE ;
: ."   ASCII " STATE §
       IF COMPILE (.") WORD HERE C§ 1+ =CELLS
ALLOT
       ELSE WORD HERE COUNT TYPE
       THEN ; IMMEDIATE
```

Förklaringar:

```
: ."   ASCII "  Ö ascii för " på stacken
       STATE §  Ö kompilering eller tolkning?
       IF COMPILE (.")  Ö kompilering!
          WORD     Ö tar STRÄNG" i indataflödet
          HERE C$ Ö antalet byte i STRÄNG
          1+       Ö även antalet ska få plats
          =CELLS   Ö gör talet på stacken jämnt
          ALLOT    Ö minne reserveras för STRÄNG
       ELSE WORD   Ö hämtar STRÄNG" för tolkning
          HERE     Ö adressen för STRÄNG
          COUNT    Ö preparerar för TYPE
          TYPE     Ö strängen skrives ut
       THEN ; IMMEDIATE

: (.") R  Ö tar fram returadressen för rutinen
       COUNT ( adr -- adr+1 b ) Ö b läses från adr
       DUP 1+  Ö stack: b b+1
       =CELLS  Ö gör  b+1 till jämn adress
       R> + >R Ö returadressen korrigeras
       TYPE ;  Ö adr+1 b --
```

Om STATE<>0 lagrar ordet ." först kodfältsadressen för (.") i ordlistan och lägger sedan hela den dimensionerade strängen (dvs första byten innehåller antalet tecken) direkt efter.
Ordet (.") läser den sträng som finns lagrad på adressen efter, och ser till att återvända till nästa FORTH-ord (och inte till strängen, som den inre interpretorn skulle åstadkomma opåverkad).

Det finns några viktiga ord till som Du kan använda för att utvidga FORTH. Viktigast bland dem är kanske <BUILDS och DOES>. Dem tar jag upp någon annan gång.

Makrorutiner

Till sist vill jag presentera en kaka med många russin i!
Kakan kommer ifrån FORTH Dimensions 1/85, och är anrättad av Don Taylor från Sydney i Australien. Det handlar om ett ord för att definiera makrorutiner.
En makrorutin är som en subrutin men med en viktig skillnad:
för en subrutin kompileras bara hoppadressen för en makrorutin kompileras koden för hela rutinen!

Dons lösning är verkligen elegant:

först definierar Du det definierande ordet MACRO: enligt Dons recpt nedan. När Du sedan definierar ett FORTH-ord med hjälp av MACRO: lagras ordets hela källkod i parameterfältet (MACRO: är ett <BUILDS-DOES>-ord).
Eftersom det definierade ordet är immediate startas exekvering av dess DOES>-kod när det skall kompileras. Och denna DOES>-kod ser till att hela källkoden i parameterfältet kompileras och inte bara en hoppadress!
Exempel:

```
MACRO: DO'   Ö testande loop av ALGOL-typ
   2DUP > IF DO ;
MACRO: LOOP'
   LOOP ELSE 2DROP THEN ;

: EXEMPEL
   CR DO' I . CR LOOP' ;
     kompilerar exakt samma kod som
: EXEMPEL
   CR
   2DUP > IF DO
   I . CR
   LOOP ELSE 2DROP THEN ;
```

Dons rutin:

```
: MACRO:
   <BUILDS IMMEDIATE  Ö skapar immediate-ord
    ASCII ; WORD      Ö källkoden lagras i HERE
    HERE C$           Ö # tecken på stacken
    BL C,             Ö och 32 lagras i HERE
    ALLOT             Ö plats för källkoden
    ASCII ö C,        Ö ordet ö ( lilla ö!) sist
    BL C,             Ö
    DOES> ( ger pfa!) Ö kompilerer makron
    R>                Ö returadr. på stacken
    BLK § >R          Ö blocknummer sparas
    IN § >R           Ö indatapekaren sparas
    TIB § >R          Ö TIB-adressen sparas
    >R                Ö returadr. åter på plats
    TIB !             Ö pfa blir temp. TIB
    0 BLK ! 0 IN ! ;  Ö nya TIB interpreteras!

: ö                   Ö obs! lilla ö !
    R>                Ö returadr. på stacken
    R> TIB !          Ö TIB återställes
    R> IN !           Ö indatapek. återställes
    R> BLK !          Ö BLK återställes
    >R ; IMMEDIATE    Ö returadr. åter på plats

TIB § CONSTANT TIB§   Ö en säkerhetsåtgärd:
: TIB! TIB§ TIB ! ;   Ö vid krasch - TIB!
```

Gracious and FORTHright!

■

Läsarnas egen Assemblerskola

S Bernle
P1 280 Tursköp
260 38 KATTARP
Tel: 042-93305

Till DIG som har EDITOR/ASSEMBLER
men INTE begriper ett SMACK!

Detta är ett försök att hjälpa dig komma igång med assemblerprogrammering.

Jag kan mycket litet om assembler och just därför tror jag att jag kanske kan hitta rätt nivå till Dig som är lika nyfiken och okunnig eller kanske ännu mera okunnig än jag om ämnet.

Skulle det vara så väl att Du även är utrustad med en printer, så skulle Du väl till och med kunna ha lite nytta av nedanstående program, men det är absolut inget krav, för huvudavsikten med denna lilla artikel är att lära Dig, något lite om, hur det går till att skriva i assembler!

De konskaper jag har om datorer och programmering har jag inhämtat från Texas-manualerna samt PB.

Vintern 83-84 köpte jag Editor-Assemblerprogrammet. Jag förväntade mig att den medföljande manualen även denna gång, lätt och enkelt, skulle ge mig tillräckliga kunskaper för att åtminstone göra en liten programsnutt.

Aj! Aj! Där bet jag mig i tummen!

Redan på sid 17 får jag veta, att det (nästan) krävs kunskap i något annat assemblerspråk för att kunna tillgodogöra sig innehållet!!!

'Okey! Skaffa dig en bok som lär dig ett assemblerspråk då!', tänkte jag.

Var enda bok jag kom över, behandlade endast språk för 8:a bitars processorer. I dessa böcker fanns ett flertal flertal begrepp som inte kunde återfinnas i manualen och tvärt om.

Jag var alltså helt utelämnad till manualen!

Jag bläddrade och läste och bläddrade och läste. Ibland begrep jag ingenting och ibland trodde jag att jag begrep, men litet senare begrep jag att jag inte begrep.

Jag har i perioder hållit på så, i ca 1 år.

Jag beklagar, men jag var faktiskt inte mycket hjälpt av de i PB publicerade ASSEMBLER-SKOLAN artiklarna.

Men, HURRA! NU HAR DET HÄNT! Jag har lyckats göra en liten programsnutt som fungerar och som jag nästan helt förstår hur den fungerar.

'Skelettet' till programmet är taget från manualen sid 303-304.

Programmet printar ut min logotype med text på printer, se nedan. Det tar datorna 1 sek i arbetstid mot tidigare i Ext Basic ca 57 sek.

Vad skulle nu Du kunna ha för nytta av mitt Logotype-program?
Jo, Du kan ändra det till Din logotype och därigenom lära Dig hur programmet fungerar.

Eftersom manualen är skriven på engelska har jag skrivit alla kommentarer också på engelska. Jag tror nämligen att det blir mindre förvirrande om man håller sig till ett språk.
En ytterligare vinst med detta med engelska är att listningen blir helt skärmliknande.

Programlistningen skiljer sig från vanliga listningar genom att alla kommentarer är skrivna ovanför och inte, som brukligt är efter det de kommenterar. På det lediga radutrymme som därigenom uppstått, anges var i manualen det sökta begreppet eller förklaringen återfinnes.
Detta för att eliminera bläddrandet samt att Du ska få veta var Du bör fördjupa Dig.

Nu är det så att Du måste ha läst och någorlunda förstått sid 18-32 för att Du överhuvudtaget ska kunna få ut något vettigt på skärmen.

Därtill måste Du nogsamt plugga in och förstå sid 57-62, innehållet på dessa sidor är mycket viktigt!
Dessa är de minimala förkunskaper Du måste ha för att ha någon chans att lära Dig något mera via min programlistning!

Det som alla BYTE-satserna innehåller är ej förklarat i listningen. Jag gör därför ett förklaringsförsök här. Grafiken i min logotype är utförd i 16 punkters grafik. Detta medför att det krävs 32 bytes information till printern för att printa en yta motsvarande en bokstavsyta.

Eftersom min logotype är 4 bokstavstecken bred och 6 tecken hög, åtgår alltså 4 x 6 x 32 = 768 bytes till grafiken.

Till texten åtgår 1 byte per ASCII-tecken.

Utöver dessa informationer måste printern även få information om i vilken mode den skall arbeta, ex vis vilket textmode el vilket grafikmode.

Därtill kräver printern ytterligare information för ex vis vagnretur, radmatning, radgöjdsinställning, tabulatorinställning ,tabulering etc etc.

De första 2 byten på DAT1-raden, i listningen nedan, ger kommando till printern att nollställa alla moden.
Nästa 2 bytes ställer radhöjden till 8/27 tum, nästa 3 bytes ställer printern i halvfarts printning, nästa byte ställer printern i condensed-mode (förtätad text), nästa 5 bytes ställer printern i 16 punkters grafikmode.
De efterföljande 128 byten är grafikbytes för 4 teckenbredders grafik, därefter följer 1 byte som ger kommandot ny rad och vagnretur.

Jag behöver nog inte förklara vad övriga bytes uträttar, det kan Du nog själv räkna ut med hjälp av Din egen printermanual.

Om Du tänker knacka in programmet, så räcker det förstås med programsatserna, men Du kan även om Du vill knacka in kommentarerna till dessa. Vad Du absolut inte får knacka in är kolumn-begränsnings-tecken och vad som finns därefter på raden.

När Du är färdig med programradsinknackningen gör Du som det står på sid 30 stycke 2.1.3.

Nu är Du klar för assembleringsrutinen. Läs om denna på sid 33-34 och gör som det står!
OBS! Glöm inte knacka in ett "R" under "OPTIONS?"

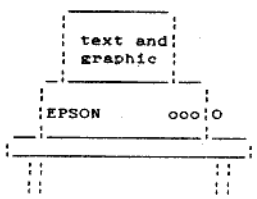
När assembleringen är klar, kan Du köra programmet via LOAD AND RUN. Läs om hur man gör på sid 36.

En fråga till Dig, som kan det här med E/A.
Går det att köra detta program med TI-Writer-modulen under huvudmenyvalet 3 dvs Användning?
Om svaret är ja, under vilket objektkodsnamn skall programmet då lagras och vad gäller för DEF satsen?

Det var allt!

S Bernle
P1 280 Tursköp
260 38 KATTARP
Tel: 042-93305

```
*********************************
*                               *
* Print a LOGOTYPE on a PRINTER *
*                               *
* 26 aug. 1985                  *
*                               *
* S. Bernle                     *
* Pl 280                        *
* 260 38 KATTARP                *
* Tel 042-93305                 *
*                               *
*********************************
```




```
*
*****************************************
*                                       *
* This is a programlisting for people   *
* that have no knowledge of programming *
* in Ti assembly language.              *
*                                       *
* All knowledge is in the manual,       *
* but where to find it?                 *
*                                       *
* I hope that this form of listing will *
* help You!                             *
*                                       *
*****************************************
```

W A R N I N G !
===============

Do NOT let Your programlines contain anything else except the contents in the column named PROGRAM WITH COMMENTS. Otherwise You will get a lot of errors during the assembly time, and the program will NOT function properly!

PROGRAM WITH COMMENTS	WHERE	in THE MANUAL
`       DEF  START`	DEF	36, 227-229, 246
`       REF  VMBW,DSRLNK,VSBW`	REF	227-229, 246
`PABBUF EQU  >1000`	PABBUF	293 (byte 2,3), 402-403
	EQU	224
`PAB    EQU  >F80`	PAB	402-403
`STATUS EQU  >837C`	STATUS	405
`PNTR   EQU  >8356`	PNTR	262-263, 404
`PDATA  DATA >0002,PABBUF,>00DC,>0000,>0004`	PDATA	293-294
	DATA	225
`       TEXT 'PIO.'`	TEXT	225
`       EVEN`	EVEN	213
`WRITE  BYTE >03`	WRITE	296
	BYTE	225
`CLOSE  BYTE >01`	CLOSE	295
`** Move PAB data into PAB in VDP-RAM **`		
`START  LI   R0,PAB`	LI	163
`       LI   R1,PDATA`		
`       LI   R2,>0E`		
`       BLWP @VMBW`	BLWP	109
	@VMBW	248
`** Store pointer to name length in >8356 **`	"Why?"	262
`       LI   R6,PAB+9`	PAB+9	294
`       MOV  R6,@PNTR`	MOV	166-167
`** Open file **`		
`       BLWP @DSRLNK`	@DSRLNK	262
`       DATA 8`	DATA 8	262
`** Change I/O op-code to write **`		
`       MOVB @WRITE,R1`	MOVB	168
`       LI   R0,PAB`		
`       BLWP @VSBW`	@VSBW	248
`** Prepare registers to move data **`		
`** into PABBUF area in VDP-RAM    **`		
`       LI   R0,PABBUF`		
`       LI   R1,DAT1-220`		
`       LI   R2,220`		
`** Load loop-counter **`		
`       LI   R3,4`		
`** Restore pointer to name length and  **`		
`** move 220 bytes into PABBUF in VDP-RAM **`		
`LOOP   MOV  R6,@PNTR`	"Why?"	262, + manualcorrection in 99:an #2,(16.2.4)
`       A    R2,R1`	A	80
`       BLWP @VMBW`		
`** Print out **`		
`       BLWP @DSRLNK`		
`       DATA 8`		
`** Has all been printed out? **`		
`       DEC  R3`	DEC	86
`       JNE  LOOP`	JNE	119
`** Restore pointer to name length **`		
`       MOV  R6,@PNTR`		

Fortsättning nästa sida

Fortsättning från nästa sida

```
BYTE >00,>1C,>00,>3C,>00,>38,>00
BYTE >30,>00,>60,>00,>40,>00,>40
BYTE >00,>40,>00,>80,>00,>80,>00
BYTE >80,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>80,>00,>80
BYTE >00,>80,>00,>40,>00,>40,>00
BYTE >40,>00,>60,>00,>30,>00,>38
BYTE >00,>3C,>00,>1C,>00,>1E,>00
BYTE >0F,>00,>0F,>C0,>07,>E0,>03
BYTE >F8,>03,>FF,>01,>FF,>00,>FF
BYTE >00,>7F,>00,>3F,>00,>1F,>00
BYTE >07,>00,>03,>00,>01,>00,>00
BYTE >00,>00,>00,>00,>0A,>1B,'^'
BYTE >11,>40,>00,>01,>80,>0F,>F0
BYTE >7F,>FE,>FF,>FF,>FF,>FF,>FF
BYTE >FF,>FF,>FF,>FF,>FF,>FF,>FF
BYTE >FF,>FF,>F0,>0F,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>01,>F0,>03,>FF
BYTE >07,>FF,>0F,>FF,>0F,>FF,>07
BYTE >FF,>03,>FF,>01,>F0,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>F0,>00,>FF
BYTE >F0,>FF,>F8,>FF,>FC,>FF,>FC
BYTE >FF,>FE,>FF,>FF,>FF,>FF,>7F
BYTE >FE,>0F,>F0,>01,>80,>0E,>20
BYTE 'S',>20,'B','e','r','n','l'
BYTE 'e',>0A,>0D,>0D,>1B,'^',>11
BYTE >40,>00,>00,>00,>00,>00,>00
BYTE >00,>80,>00,>C0,>00,>E0,>00
BYTE >F8,>00,>FC,>00,>FE,>00,>FF
BYTE >00,>FF,>80,>FF,>C0,>1F,>C0
BYTE >07,>E0,>03,>F0,>00,>F0,>00
BYTE >78,>00,>38,>00,>3C,>00,>1C
BYTE >00,>0C,>00,>06,>00,>02,>00
BYTE >02,>00,>02,>00,>01,>00,>01
BYTE >00,>01,>00,>00,>C0,>00,>FF
BYTE >80,>FF,>F0,>FF,>F0,>FF,>80
BYTE >C0,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>0A,>1B,'^'
BYTE >11,>40,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>03
BYTE >01,>FF,>0F,>FF,>0F,>FF,>01
BYTE >FF,>00,>03,>00,>00,>80,>00
BYTE >80,>00,>80,>00,>40,>00,>40
BYTE >00,>40,>00,>60,>00,>30,>00
BYTE >38,>00,>3C,>00,>1C,>00,>1E
BYTE >00,>0F,>00,>0F,>C0,>07,>E0
BYTE >03,>F8,>03,>FF,>01,>FF,>00
BYTE >FF,>00,>7F,>00,>3F,>00,>1F
BYTE >00,>07,>00,>03,>00,>01,>00
BYTE >00,>00,>00,>00,>00,>1B,'='
BYTE >20,>20,'P','1',>20,'2''8',
BYTE '0',>20,'T','u','r','s','k'
BYTE '|','p',>0A,>1B,'^',>11,>40
BYTE >00,>01,>80,>0F,>F0,>7F,>FE
BYTE >FF,>FF,>FF,>FF,>7F,>FF,>3F
BYTE >FF,>1F,>FF,>1F,>FF,>0F,>FF
BYTE >00,>0F,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>0F,>F0,>FF,>FF,>FF,>FF
BYTE >FF,>FF,>FF,>FF,>FF,>FF,>FF
BYTE >FF,>0F,>F0,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>F0,>0F,>FF,>FF,>FF
BYTE >FF,>FF,>FF,>FF,>FF,>FF,>FF
BYTE >FF,>FF,>FF,>FF,>7F,>FE,>0F
BYTE >F0,>01,>80,>20,>20,'2','6'
BYTE '0',>20,'3','8',>20,'K','A'
BYTE 'T','T','A','R','P',>0A,>1B
```

Fortsättning nästa sida

```
** Change I/O op-code to close **
*--------------------------------*
       MOVB @CLOSE,R1
       LI   R0,PAB

       BLWP @VSBW

** Close file **
*--------------*
       BLWP @DSRLNK
       DATA 8

** So that no errors is reported **
*---------------------------------*
       CLR  R0                      CLR     184  \
       MOVB R0,@STATUS                           = "Why?"  441
       RT                           RT      207  /


**    Data that will be printed       **
*---------------------------------------*
DAT1   BYTE >1B,'@',>1B,'A',>08,>1B,'s'   DAT1    225
       BYTE >01,>0F,>1B,'>',>1B,'^',>11
       BYTE >40,>00,>00,>00,>00,>00,>00   + + + +  + + + + + + + + + + + + + +
       BYTE >00,>00,>01,>00,>03,>00,>07
       BYTE >00,>1F,>00,>3F,>00,>7F,>00   END     234
       BYTE >FF,>01,>FF,>03,>FF,>03,>E4
       BYTE >07,>E0,>0F,>C0,>0F,>00,>1E
```

Fortsättning föregående sida

Fortsättning från föregående sida

```
       BYTE '^',>11,>40,>00,>00,>00,>00
       BYTE >00,>00,>00,>80,>00,>C0,>00
       BYTE >E0,>00,>F8,>00,>FC,>00,>FE
       BYTE >00,>FF,>00,>FF,>80,>FF,>C0
       BYTE >1F,>C0,>07,>E0,>03,>F0,>00
       BYTE >F0,>00,>78,>00,>38,>00,>3C
       BYTE >00,>1C,>00,>0C,>00,>06,>00
       BYTE >02,>00,>02,>00,>02,>00,>01
       BYTE >00,>01,>00,>01,>00,>00,>C0
       BYTE >00,>FF,>80,>FF,>F0,>FF,>F0
       BYTE >FF,>80,>C0,>00,>00,>00,>00
       BYTE >01,>00,>01,>00,>01,>00,>02
       BYTE >00,>02,>00,>02,>00,>06,>00
       BYTE >0C,>00,>1C,>00,>3C,>00,>38
       BYTE >00,>78,>00,>F0,>03,>F0,>07
       BYTE >E0,>1F,>C0,>FF,>C0,>FF,>80
       BYTE >FF,>00,>FE,>00,>FC,>00,>F8
       BYTE >00,>E0,>00,>C0,>00,>80,>00
       BYTE >00,>00,>00,>00,>00,>00,>20
       BYTE >20,'T','e','1',':',>20,'0'
       BYTE '4','2','-','9','3','3','0'
       BYTE '5',>0A,>0D,>1B,'@'

       END
```

ALPHA LOCK CHECK

av Jan Alexandersson

Från den tyska tidningen COMPUTER KONTAKT kommer följande lilla assembler-program som kan anropas från BASIC eller EXTENDED BASIC med CALL LINK("ALPHA"). Programmet kontrollerar om ALFHA LOCK är uppe. I annat fall får man en varningstext.

```
       REF VMBW,VMBR  * ändras till EQU för XB

RTS    BSS  2
SAVE   BSS  16
ALPHA  MOV  R11,'RTS
       LI   R0,744
       LI   R1,SAVE
       LI   R2,16
       BLWP 'VMBR
DRIN   CLR  R12

       SBZ  >0015
       TB   >0007
       JEQ  ARAUS

       BL   'RAUS
       JMP  DRIN

RAUS   LI   R0,744
       LI   R1,RA
       LI   R2,16
       BLWP 'VMBW
       LIMI 2
       LIMI 0
       B    *R11

RA     TEXT 'ALPHA LOCK RAUS"'

ARAUS  LI   R0,744
       LI   R1,SAVE
       LI   R2,16
       BLWP 'VMBW
       SBO  >0015
       MOV  'RTS,R11
       RT
```

MER OM PERSONAL RECORD KEEPING

av Jan Alexandersson

I PB 85-2 fanns en beskrivning av de BASIC-kommandon som kan användas tillsammans med Personal Record Keeping eller Statistics. Jag har efter att detta skrevs fått tag i en beskrivning som getts ut av TEXAS. Det visar sig då att det finns ytterligare möjligheter.

CALL D

Man kan ange flera DISPLAY-satser i samma CALL D t.ex. CALL D(1,8,4,"MENY",4,1,6,"1 ADAM",6,1,8,"2 BERTIL"). Den rätta syntaxen är:
CALL D(rad,kolumn,längd,data,...)

CALL A

Värdet av test kan även bli 3-7 om följande tangent trycks:

(AID)	3
(REDO)	4
(PROC'D)	5
(BEGIN)	6
(BACK)	7

Utöver den syntax som beskrev i PB 85-2 finns ytterligare två möjligheter:

```
CALL A(rad,kolumn,längd,test,data,fält)
```

Om fält anges kontrollerar ACCEPT att värdet stämmer med definitionen för fältet(typ, bredd, antal decimaler osv)

```
CALL A(rad,kolumn,längd,test,data,min,max)
```

Min och max kontrollerar att ACCEPT ligger mellan dessa två värden.

NUC–SPELET

av Lars-Åke Ragnarsson

Detta är ett spel för Extended BASIC och JOYSTICK. Det gäller att skjuta ned den fallande bomben med kanonen innan den når marken. Du siktar genom att ställa härkorset i lämplig vinkel och trycka på (FIRE). Redaktören har tagit sig friheten att ändra rad 400 så att den blir "IF S=1 THEN 420". Detta förhindrar dubbeltryckning när du ska skriva ditt namn i början av programmet. Du bläddrar bland bokstäverna med JOYSTICK åt vänster och åt höger.Tryck på (FIRE) när du hittat rätt bokstav.
Samma rutin finns i spelet PUC MAN som fanns publicerat i Nittinian 1983-3. Rad 2760 bör där ändras på samma sätt "2760 IF SLASK=1 THEN 2890".

```
100 "DEN 15/8-16/8-1984 AV LARS-AKE RAGNARSSON. IDE'
    NIKLAS INGMARSSON
110 "EXTENDED BASIC OCH JOYSTICK
120 GOTO 180 :: CALL CHAR :: CALL CLEAR :: CALL COINC
     :: CALL COLOR :: CALL DELSPRITE :: CALL DISTANCE
     :: CALL JOYST :: CALL KEY
130 CALL LOCATE :: CALL PATTERN :: CALL POSITION :: C
    ALL SCREEN :: CALL SOUND :: CALL SPRITE :: CALL V
    CHAR :: CALL HCHAR :: CALL CHARSET
140 CALL MOTION :: CALL MAGNIFY :: CALL LOCATE
150 DATA 10,3,3,7,121
160 NAMN$,HI$,BOK,TR1,Q,BO2,DOD,BO,D,POS6,A,I,J,V,H,H
    1,AN,CH,X,Y,K,S,SKOTT,POS1,POS2,POS3,POS4,DIST,DI
    ST1,PO,POS5,TR,HI
170 "!P-
180 CALL DELSPRITE(ALL):: CALL CLEAR :: CALL SCREEN(2
    ):: CALL MAGNIFY(4)
190 A=0
200 FOR I=2 TO 8 :: CALL COLOR(I,8,1):: NEXT I
210 CH=96 :: POS3=86
220 CALL CHAR(96,"183C3E3E3F3F3F3D3D3D3C3C3C3C3C18183
    C3C3C3C3CBCBCFCFCFCFCFC7C7C18")
230 CALL CHAR(100,"183C3C3C3C3C3C3C3C3C3C3C3E1F1F0F0318
    3C3C3C3C3C3C3C3C3C3C7CF8F8F0C0")
240 CALL CHAR(104,"070F0E1E1C3C383838383C1C1E0E0F07F0
    F83818000000000000000001838F8F0")
250 CALL CHAR(63,"3C4299A1A199423C")
260 FOR I=1 TO 3 :: CALL SPRITE(#I,CH,INT(RND*13+3),9
    6,1,0,70)
270 CALL SOUND(-200,-5,0):: CALL POSITION(#I,POS1,POS
    2):: IF POS2>=POS3 THEN CALL MOTION(#I,0,0):: CAL
    L LOCATE(#I,40,POS3)ELSE 270
280 CH=CH+4 :: POS3=POS3+24
290 NEXT I
300 DISPLAY AT(20,8):"? 1984   UNO" :: DISPLAY AT(14,
    7):"SKRIV DITT NAMN"
310 CALL HCHAR(17,15,45,3)
320 NAMN$=""
330 FOR I=1 TO 3 :: BOK=65
340 A=A+1 :: CALL JOYST(1,X,Y):: CALL VCHAR(16,14+I,B
    OK)
350 FOR D=1 TO 10 :: NEXT D
360 IF X<0 THEN BOK=BOK-1 :: IF BOK>=64 THEN 370 ELSE
     BOK=93
370 IF X>0 THEN BOK=BOK+1 :: IF BOK<=93 THEN 380 ELSE
     BOK=65
380 CALL VCHAR(16,14+I,BOK)
390 IF A=130 THEN GOTO 180
400 CALL KEY(1,K,S):: IF S=1 THEN 420 ! REV AV RED
410 GOTO 340
420 NAMN$=NAMN$&CHR$(BOK)
430 NEXT I
440 CALL SOUND(250,523,0):: CALL SOUND(125,392,0):: C
    ALL SOUND(250,370,0):: CALL SOUND(125,392,0):: CA
    LL SOUND(350,415,0)
450 CALL SOUND(550,392,0):: CALL SOUND(250,494,0):: C
    ALL SOUND(250,523,0)
460 CALL CLEAR :: CALL DELSPRITE(ALL)
470 CALL CHAR(96,"0001012111090101FF01010911210101000
    0000810200000FE0000201008")
480 CALL CHAR(100,"0000000000050307070305000000000000
    00000000A0C0E0E0C0A0")
490 CALL CHAR(104,"0107030101030303030707070703010080
    E0C08080C0C0C0C0E0E0E0E0C08000")
500 CALL CHAR(120,"FFFFC3C3C3FFFFABD5ABD5ABD5ABD5")
510 CALL CHAR(128,"FF9999FFFF9999FF",136,"FFB9AFE9BFA
    9EFFF")
520 CALL CHAR(116,"000306071F7F3F7F1F3F4F1F2602000040
    48B0FCF9FEF8FEF8F0FCF0E0380000")
530 CALL CHAR(108,"0000030303030707070737F7F7FFFFF00
    0080808080C0C0C0C0D8F8FCFCFEFE")
540 CALL CHAR(40,"18187E7E18181818",41,"1C08081C1C1C1
    C08")
550 CALL CLEAR :: CALL SCREEN(2):: RANDOMIZE :: CALL
    MAGNIFY(3)
560 FOR I=2 TO 14 :: CALL COLOR(I,2,2):: NEXT I
570 A=1 :: BO=0 :: DOD=0
580 CALL HCHAR(15,1,112,96):: CALL HCHAR(18,1,104,96)
    :: CALL HCHAR(21,1,96,128)
590 DATA 12,13,21,5,121,9,26,26,8,121,9,1,1,10,136,8,
    23,24,11,136
600 RESTORE 150
610 DATA 7,5,6,14,128,10,10,11,11,128,18,12,13,3,128,
    10,14,16,11,128,10,28,29,11,128
620 DATA 5,7,9,18,120,7,31,32,16,120
630 FOR I=1 TO 12 :: READ V,H,H1,AN,CH
640 FOR J=H TO H1 :: CALL VCHAR(V,J,CH,AN):: NEXT J :
    : NEXT I
650 FOR I=5 TO 8 :: CALL COLOR(I,16,8):: NEXT I
660 DISPLAY AT(5,12)BEEP:"ALPHA-LOCK"
670 CALL COLOR(2,16,8,3,1,16,4,1,16,9,4,4,10,3,3,11,1
    3,13,12,2,16,13,2,12,14,2,10)
680 CALL SCREEN(6)
690 DISPLAY AT(1,24):HI$ :: DISPLAY AT(1,8):NAMN$ ::
    DISPLAY AT(3,2):"POINTS" :: DISPLAY AT(1,15):"HI-
    SCORE"
700 DISPLAY AT(3,16):HI
710 DISPLAY AT(3,8)SIZE(6):PO :: IF PO>HI THEN DISPLA
    Y AT(3,16)SIZE(6):PO
720 CALL HCHAR(24,3,41,BO)
730 FOR I=1 TO 100 :: NEXT I :: DISPLAY AT(5,12)
740 CALL SPRITE(#1,96,16,96,128,#4,108,2,174,128)
750 A=A+.1 :: CALL SPRITE(#2,104,16,1,INT(RND*125*2+3
    ),A*INT(RND*-3+5),A*INT(RND*6-3))
760 CALL KEY(1,K,S):: IF K=18 AND SKOTT=0 THEN 770 EL
    SE 800
770 CALL SPRITE(#3,100,2,176,128):: SKOTT=1 :: CALL P
    OSITION(#1,POS1,POS2,#3,POS3,POS4):: CALL DISTANC
    E(#3,#1,DIST):: DIST1=SQR(DIST)
780 CALL MOTION(#3,(POS1-POS3)/DIST1*20,(POS2-POS4)/D
    IST1*20)
790 FOR I=0 TO 30 STEP 10 :: CALL SOUND(-150,-6,I)::
    NEXT I
800 IF POS4>240 OR POS4<10 THEN CALL DELSPRITE(#3)::
    SKOTT=0
810 GOSUB 900 :: IF SKOTT=0 THEN 820 ELSE CALL COINC(
    #3,#2,6,TR):: IF TR THEN 850
820 CALL POSITION(#3,POS3,POS4):: IF POS3>190 OR POS4
    <10 OR POS4>240 THEN CALL DELSPRITE(#3):: SKOTT=0
830 PO=PO+10 :: CALL POSITION(#2,POS5,POS6):: IF POS5
    >165 THEN 910
840 GOSUB 900 :: IF SKOTT=0 THEN 880 ELSE CALL COINC(
    #3,#2,6,TR):: IF TR THEN 850 ELSE 880
850 CALL DELSPRITE(#3):: BO=BO+1 :: SKOTT=0 :: CALL H
    CHAR(24,2,41,BO)
860 CALL COLOR(#2,12):: CALL PATTERN(#2,116):: FOR I=
    10 TO 0 STEP -2 :: CALL SOUND(-100,-7,I):: NEXT I
     :: CALL DELSPRITE(#2)
870 CALL DISTANCE(#2,166,128,D):: PO=PO+10*INT(10000/
    SQR(D)):: FOR I=0 TO 30 STEP 6 :: CALL SOUND(-400
    ,-7,I):: NEXT I :: GOTO 750
880 DISPLAY AT(3,8)SIZE(6):PO :: IF PO>HI THEN DISPLA
    Y AT(3,16)SIZE(6):PO :: IF SKOTT=0 THEN 760 ELSE
    CALL COINC(#3,#2,6,TR):: IF TR THEN 850
890 GOTO 760
900 CALL JOYST(1,X,Y):: CALL MOTION(#1,-Y*5,X*5):: RE
    TURN
910 CALL DELSPRITE(ALL):: FOR I=6 TO 12 :: FOR J=2 TO
     14 :: CALL COLOR(J,I,I):: CALL SCREEN(I-1):: NEX
    T J :: CALL SOUND(-1500,-7,0):: NEXT I
920 CALL SCREEN(12):: FOR I=0 TO 30 :: CALL SOUND(-50
    0,-7,I):: NEXT I :: DOD=DOD+1 :: IF DOD>3 THEN 94
    0
930 CALL HCHAR(1,2,40,DOD):: GOTO 580
940 CALL CLEAR :: CALL DELSPRITE(ALL):: FOR I=2 TO 8
    :: CALL COLOR(I,16,8):: NEXT I :: CALL SCREEN(6):
    : IF PO<HI THEN 1000
950 CALL SOUND(125,262,0):: CALL SOUND(125,262,0):: C
    ALL SOUND(250,262,0):: CALL SOUND(250,330,0):: CA
    LL SOUND(250,392,0)
960 CALL SOUND(500,523,0,392,0,330,0)
970 DISPLAY AT(4,11):"GRATTIS ";NAMN$ :: DISPLAY AT(6
    ,13):"DU SLOG" :: DISPLAY AT(8,13):"HI-SCORE"
980 DISPLAY AT(10,13)SIZE(10):"MED ";PO-HI :: DISPLAY
     AT(10,24):"POANG"
990 HI=PO :: HI$=NAMN$
1000 DISPLAY AT(12,1):"DU FICK ";PO;"POANG" :: DISPLAY
     AT(14,1):"HI-SCORE= ";HI;"INNEHAS AV ";HI$
1010 PO=0 :: DISPLAY AT(22,5):"TRYCK KNAPP OM DU VILL
     SPELA IGEN"
1020 FOR I=200 TO 1 STEP -1 :: DISPLAY AT(1,3):INT(I/1
     0):: CALL KEY(1,K,S):: IF K<>-1 THEN 550
1030 NEXT I
1040 GOTO 180
```

Subrutiner

av Jan Alexandrsson

Vid medlemsmötet 15 sept framfördes önskemål om subprogram i EXTENDED BASIC som motsvarar CALL POKEV och CALL PEEKV i Mini Memory eller Editor/Assembler. Jag trodde då att jag hade gamla program som fixade detta. Det enda som jag har är ett program som gör PEEKV och kräver expansionsminne 32k. Det är skrivit för över ett år sedan och är något slaskigt gjort. Det går att göra det mycket bättre. Jag hoppas att någon av läsarna kan skriva ett riktigt CALL PEEKV och CALL POKEV för EXTENDED BASIC och sända det till mig eller Claes.

```
100 REM PEEKV XB
110 REM JAN ALEXANDERSSON
120 REM 1985-09-24
130 DEF HB(X)=INT(X/256)
140 DEF LB(X)=INT(X-HB(X)*256)
150 ADR=16376
160 CALL INIT
170 CALL LOAD(12288,192,32,47,240,2,1,47,240,2,2,0,1,4,
    32,32,44,4,91)
180 CALL LOAD(ADR,80,69,69,75,86,32,48,0)
190 CALL LOAD(8196,HB(ADR),LB(ADR))
200 CALL CLEAR
210 INPUT "VDP DEC ADR ":DEC
220 INPUT "ANTAL BYTES ":ANT
230 FOR I=1 TO ANT
240 CALL LOAD(12272,HB(DEC+I-1),LB(DEC+I-1))
250 CALL LINK("PEEKV")
260 CALL PEEK(12272,A)
270 GOSUB 330
280 PRINT HEX$;
290 NEXT I
300 PRINT
310 GOTO 210
320 REM DEC TO HEX
330 VH=INT(A/16)
340 IF VH>9 THEN 400
350 HEX$=STR$(VH)
360 VL=A-VH*16
370 IF VL>9 THEN 420
380 HEX$=HEX$&STR$(VL)
390 RETURN
400 HEX$=CHR$(VH+55)
410 GOTO 360
420 HEX$=HEX$&CHR$(VL+55)
430 GOTO 390
```

Vid programmering i BASIC eller EXTENDED BASIC är det mycket praktiskt att ha ett bibliotek av subrutiner som vid behov kan stoppas in i nya program. För dig som har diskdrive bör dessa lagras i MERGE-format dvs DIS/VAR 163. Har du bra subrutiner så sänd in dessa till tidningen.

Exempel på några av mina subrutiner visas nedan.

```
1 REM SVENSKA BA
8000 CALL CHAR(91,"00280038447C4444")
8010 CALL CHAR(92,"0028007C4444447C")
8020 CALL CHAR(93,"00382838447C4444")
8030 CALL CHAR(123,"0000280038447C44")
8040 CALL CHAR(124,"000028007C44447C")
8050 CALL CHAR(125,"0000382838447C44")
```

```
1 REM SVENSKA XB
8000 CALL CHAR(91,"00280038447C4444002800 7C4444447C0038
     2838447C4444")
8010 CALL CHAR(123,"0000280038447C4400002800 7C44447C000
     0382838447C44")
```

```
1 REM SUBJOYST
9000 SUB JOYST(NR,X,Y)
9010 CALL KEY(NR,KEY,STA)
9020 X=-4*(KEY=3)+4*(KEY=2)
9030 Y=-4*(KEY=5)+4*(KEY=0)
9040 SUBEND
```

99-AN SOM NYTTODATOR

av Jan Alexandersson
(efter utkast av Lennart Lindberg)

Paul Bergehamn i Värnamo har lyckats rationalisera en del av sitt arbete som företagare med hjälp av TI-99/4A. Han använder den för en rutin som hanterar packsedlar och fakturor samt ger ett resultatmått per order. Det fantastiska är att han gör det med en oexpanderad 99:a.

Nu vill emellertid Paul gå vidare och sköta sin bokföring med hjälp av datorn. Kan någon hjälpa honom. Kontakta honom direkt tel. 0370/110 11 eller 110 24. Om du tycker att dina program är värda att publicer så skicka in dem till tidningen också.

För att stimulera dig att skriva nyttoprogram tänkte vi ha en liten tävling om bra nyttoprgram. Vi vill ha dina bidrag senast 31 mars 1986. Programera mera numera"

För att resultatet skall passa alla delar vi in det hela i 5 klasser:

1.BASIC med kassett.
2.Extended BASIC med kassett.
3.Extended BASIC med 32k expansions-RAM och kassett.
4.Mini-Memory med kassett.
5.Personal Record Keeping/Statistcs med kassett.

Du kan titta på tidigare publicerade nyttoprogram:

P:TEXTIN/P:TEXTUT	av Börje Häll	PB 84-3
TELEFONREGISTER	av Anders Persson	PB 84-4
LT-WRITER	av Lennart Thelander	PB 85-1,2,3,4
SNABBFIL	av Jan Alexandersson	PB 85-2

Vi har även fått in ett program för Extended BASIC med 32k RAM och kassett från Sven Lungren, Ingvarshem 5, 245 00 STAFFANSTORP, tel 046-25 64 21. Han kallar det SLDATABAS med TURBO.

Vi publicerar inte detta i tidningen just nu men om du vill ha programmet så sänd en tom kassett, frankerat svarskuvert och 10 kr till Sven.

Han har satt TURBO på kassettspelaren på samma sätt som LT-WRITER. Programmet hanterar telefonregister med namn, adress och telefonnummer. Data lagras i PROGRAM-format utan uppehåll med möjlighet till CHECK av att det fastnat på bandet.

■

```
1 REM SUB CHARFILTER
9000 SUB CHARFILTER(TEXT$,REV$)
9010 FOR I=1 TO LEN(REV$)
9020 CH$=SEG$(REV$,I,1)
9030 IF CH$="$" THEN CH=175 ELSE IF CH$="'" THEN CH=170
     ELSE IF CH$="^" THEN CH=206 ELSE 9070
9040 IF POS(TEXT$,CH$,1)=0 THEN 9070
9050 TEXT$=SEG$(TEXT$,1,POS(TEXT$,CH$,1)-1)&CHR$(CH)&SE
     G$(TEXT$,POS(TEXT$,CH$,1)+1,80)
9060 GOTO 9040
9070 NEXT I
9080 SUBEND
```

CHARFILTER använder jag när jag har svenska tecken på printern men vill få amerikanska tecken för $, ' eller ^. Det är anpassat till en SEIKOSHA 550-printer men går att modifiera för annat.
Subpogrammet anropas med CALL CHARFILTER(A$,R$) där A$ är den textsträng som skall ändras och R$="$'^" om alla tre ASCII-koder skall ändras. ■

LANDER

av Bo Asmundsson, Linköping

Detta spel som är skrivet i Extended BASIC går ut på att försöka styra din landningsfarkost från starten till landningsplatsen. Styr med:

E=gas
S=styr vänster
D=styr höger

Akta dig för att komma för nära väggarna och landa inte för hastigt. Kortaste tid skrivs på skärmen.

```
 70 REM LANDER
 80 REM EXTENDED BASIC
 90 REM AV BO ASMUNDSSON
100 LT=1000000
110 CALL CLEAR
120 CALL SCREEN(2)
130 FOR A=1 TO 9
140 CALL COLOR(A,16,1)
150 NEXT A
160 CALL CHAR(93,"18183C42427E4242")
170 CALL CHAR(91,"24003C42427E4242")
180 CALL CHAR(92,"24003C424242423C")
190 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
200 CALL CHAR(97,"FFF9F9F9F9F9F9F9")
210 CALL CHAR(98,"F9F9F9F9F9F9F9F9")
220 CALL CHAR(99,"F9F9F9F9F9F9F9FF")
230 DISPLAY AT(2,1):"a  ``a a  a `a  ``a `a    b
    b b `a b b a b   b b   b  ``b b `b b b `b  `b
       b   b b b  b b c b   b a   "
240 DISPLAY AT(6,1):"``b c c c  c `c  ``b c c"
250 DISPLAY AT(8,1):"KÖR DIN LANDARE GENOM DE": :"TRÅ
    NGA GÅNGARNA TILL DIN LA-": :"NDNIGSPLATS PÅ KORT
    AST": :"MÖJLIGA TID."
260 DISPLAY AT(16,1):"AKTA DIG FÖR ATT ÅKA FÖR": :"NÄ
    RA VÄGGARNA": :"STYR MED:   E=GAS":"            S
    =STYR VÄNSTER":"            D=STYR HÖGER"
270 DISPLAY AT(24,1):"TRYCK NED EN TANGENT" :: CALL K
    EY(0,K,S):: IF S=0 THEN 270
280 CALL CLEAR :: DISPLAY AT(10,1):"OBS"  LANDA INTE
     FÖR SNABBT" :: FOR D=1 TO
500 :: NEXT D
290 CALL CLEAR :: CALL COLOR(9,14,1)
300 CALL CHAR(112,"000084761F1F0F3FFF03070D1921030201
    0286CCF8F3FCF8F0F8FE311804")
310 CALL HCHAR(1,1,96,32)
320 CALL VCHAR(1,32,96,24)
330 CALL HCHAR(24,1,96,32)
340 CALL VCHAR(1,1,96,24)
350 FOR A=6 TO 10
360 CALL HCHAR(A,9,96,23)
370 NEXT A
380 FOR A=14 TO 24
390 CALL HCHAR(A,1,96,29)
400 NEXT A
410 T=0
420 DISPLAY AT(20,2)SIZE(4):"TID:"
430 DISPLAY AT(22,2)SIZE(11):"LÄGSTA TID:"
440 FOR A=1 TO 8
450 CALL COLOR(A,16,1)
460 NEXT A
470 CALL MAGNIFY(3)
480 CALL CHAR(104,"00070F1E1C1E0F07070C18101010103800
    E0F0783878F0E0E03018080808081C")
490 CALL CHAR(108,"FFFFFFFF0000000000000000000000FF
    FFFFFF")
500 CALL SPRITE(#2,108,16,182,233)
510 CALL SPRITE(#1,104,8,16,230)
520 CALL KEY(1,K,S)
530 IF K=5 THEN G=G-.5 :: CALL SOUND(-800,125,8,-7,15
    ):: GOTO 560
540 IF K=2 THEN H=H-.3
550 IF K=3 THEN H=H+.3
560 CALL MOTION(#1,G,H)
570 G=G+.2
580 CALL POSITION(#1,X,Y)
590 IF X>3*8 AND X<11*8-7 AND Y>49 THEN 680
600 IF X>11*8 AND Y<232 THEN 680
610 IF Y<8 OR Y>234 THEN 680
620 IF X<8 OR X>184 THEN 680
630 CALL COINC(ALL,KG)
640 IF KG THEN 790
```

MER ÄNDRING AV P:TEXTIN

av Börje Häll

Som omtalades i den tidigare ändringen till ovan nämnda program, så stökar kommatecknet till det vid inmatning av texten, men framför allt vid inläsning från kassett/flexskiva. I den här ändringen görs kommatecknet om till ASCII 128 för att komma ifrån problemet vid läsning från kassett/flexskiva. Dock måste citationstecken börja och avsluta texten när den skrivs in från tangentbordet.

En enkel rättningsrutin har lagts till. Alla inmatade rader måste gås igenom vid rättningen. Om raden inte ska ändras då trycker man på ENTER utan någon inmatning. Om raden ska ändras så måste hela raden skrivas om.

Omvandlingen medför att programmet P:TEXTUT måste kompletteras för att kommatecknen ska återfås.

Ändringar i P:TEXTIN.

```
110 REM Version 1.3

410 PRINT "sluta med citationstecken ""."
411 Utgår

561 REM Gör om kommatecknet till ASCII 128
562 FOR J=0 TO I
563 P=POS(TEXT$(J),",",1)
564 IF P=0 THEN 567
565 TEXT$(J)=SEG$(TEXT$(J),1,P-1)&CHR$(128)&SEG$(TEXT$(
    J),P+1,255)
566 GOTO 563
567 NEXT J
568 INPUT "Några ändringar? (J/N) ":SVAR$
569 IF (SVAR$="J")+(SVAR$="j")+(SVAR$="N")+(SVAR$="n")=
    0 THEN 568
570 IF (SVAR$="J")+(SVAR$="j")<=-1 THEN 810
579 REM Ska kassett eller flexskiva användas?

800 STOP
810 CALL CLEAR
820 FOR J=0 TO I
830 PRINT TEXT$(J)::
840 INPUT "Ny text. OBS citationstecken":SVAR$
850 IF SVAR$="" THEN 870
860 TEXT$(J)=SVAR$
870 NEXT J
880 GOTO 568
890 END
```

Följande komplettering måste göras i P:TEXTUT för att kommatecknen ska återfås.

```
341 P=POS(TEXT$,CHR$(128),1)
342 IF P=0 THEN 350
343 TEXT$=SEG$(TEXT$,1,P-1)&","&SEG$(TEXT$,P+1,255)
344 GOTO 341
```

```
650 T=T+1
660 DISPLAY AT(20,6)SIZE(4):T
670 GOTO 520
680 REM ÑÑÑÑCRASHÑÑÑÑ
690 G,H=0 :: CALL MOTION(#1,0,0)
700 CALL COLOR(#1,9)
710 CALL PATTERN(#1,112)
720 FOR A=9 TO 16 :: CALL COLOR(#1,A):: CALL SOUND(-8
    00,125,A,-5,15):: NEXT A
730 CALL COLOR(#1,11)
740 CALL SOUND(100,-7,15)
750 CALL DELSPRITE(#1)
760 FOR A=1 TO 500
770 NEXT A
780 GOTO 410
790 CALL MOTION(#1,0,0)
800 IF G>2 THEN 680
810 IF T<LT THEN DISPLAY AT(22,13)SIZE(4):T :: LT=T
820 FOR A=200 TO 300 STEP 10
830 CALL SOUND(100,A,4)
840 NEXT A
850 GOTO 410
```

ORMAR

av Bo Asmundsson,Linköping

Detta spel är för BASIC och JOYSTICK. Två ormar syns på skärmen, en vit och en svart. De styrs av två spelare med var sin joystick. Man skall akta sig för att köra in i väggarna, motståndaren, sig själv eller korset i mitten, gör man det får motståndaren poäng.
Försök ta tärningen som sätts ut slumpvis på skärmen det ger ett poäng. Den spelare som först når 9 poäng är vinnaren.

```
 70 REM ORMAR
 80 REM BASIC OCH JOYSTICK
 90 REM AV BO ASMUNDSSON
100 P01=48
110 P02=48
120 F1=-1
130 FY2=-1
140 F=0
150 FY=0
160 CALL CLEAR
170 CALL SCREEN(13)
180 FOR Q=2 TO 8
190 CALL COLOR(Q,2,16)
200 NEXT Q
210 CALL COLOR(9,16,4)
220 CALL COLOR(10,16,13)
230 CALL COLOR(11,2,13)
240 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
250 CALL CHAR(97,"0000000000000000")
260 CALL CHAR(104,"7EFFC3C3C3C3FF7E")
270 CALL CHAR(105,"7E99997E7E427E7E")
280 CALL CHAR(112,"7EFFC3C3C3C3FF7E")
290 CALL CHAR(113,"7E99997E7E427E7E")
300 CALL CHAR(120,"FF9999E7E79999FF")
310 CALL CHAR(95,"24003C424242423C")
320 CALL CLEAR
330 PRINT "``VIT=````````SVART=````````"
340 PRINT "````````ORMARNAS`KAMP````````": : : : : : :
     : : : : : : : : : : : : : : :
350 CALL HCHAR(1,9,P01)
360 CALL HCHAR(1,23,P02)
370 CALL VCHAR(1,31,96,96)
380 CALL HCHAR(23,1,96,66)
390 CALL VCHAR(10,16,97,5)
400 CALL HCHAR(12,14,97,5)
410 RANDOMIZE
420 Y=12
430 U=12
440 X=10
450 Z=22
460 GOTO 870
470 CALL JOYST(1,LU,LI)
480 IF (LU=0)*(LI=0)THEN 510
490 F=LU/4
500 F1=LI/-4
510 CALL HCHAR(Y,X,104)
520 X=X+F
530 Y=Y+F1
540 CALL GCHAR(Y,X,C)
550 IF C=120 THEN 700
560 IF C>32 THEN 1020
570 CALL HCHAR(Y,X,105)
580 CALL JOYST(2,LY,LZ)
590 IF (LY=0)*(LZ=0)THEN 620
600 FY=LY/4
610 FY2=LZ/-4
620 CALL HCHAR(U,Z,112)
630 Z=Z+FY
640 U=U+FY2
650 CALL GCHAR(U,Z,C)
660 IF C=120 THEN 790
670 IF C>32 THEN 930
680 CALL HCHAR(U,Z,113)
690 GOTO 470
700 P01=P01+1
710 CALL HCHAR(1,9,P01)
720 CALL SOUND(100,362,0)
730 FOR A=1 TO 5
740 CALL HCHAR(Y,X,49)
750 CALL HCHAR(Y,X,104)
760 NEXT A
770 IF P01=57 THEN 1260
780 GOTO 870
790 P02=P02+1
800 CALL HCHAR(1,23,P02)
810 CALL SOUND(100,262,0)
820 FOR A=1 TO 5
830 CALL HCHAR(U,Z,49)
840 CALL HCHAR(U,Z,112)
850 NEXT A
860 IF P02=57 THEN 1110
870 RE=INT(RND*32+1)
880 ER=INT(RND*24+1)
890 CALL GCHAR(ER,RE,C)
900 IF C>32 THEN 870
910 CALL HCHAR(ER,RE,120)
920 GOTO 470
930 P01=P01+1
940 CALL HCHAR(1,9,P01)
950 CALL SOUND(100,-5,0,110,0,652,0)
960 FOR A=1 TO 10
970 CALL COLOR(11,9,13)
980 CALL COLOR(11,2,13)
990 NEXT A
1000 IF P01=57 THEN 1260
1010 GOTO 1300
1020 P02=P02+1
1030 CALL HCHAR(1,23,P02)
1040 CALL SOUND(100,-7,0,652,0,253,0)
1050 FOR A=1 TO 10
1060 CALL COLOR(10,9,13)
1070 CALL COLOR(10,16,13)
1080 NEXT A
1090 IF P02=57 THEN 1110
1100 GOTO 1300
1110 M$="SVART`HAR`VUNNIT`KAMPEN`````"
1120 G=23
1130 GOSUB 1380
1140 M$="TRYCK`SKJUTKNAPPEN`F_R`SPEL`"
1150 G=24
1160 GOSUB 1380
1170 CALL KEY(1,K,S)
1180 IF K=18 THEN 100
1190 CALL KEY(2,D,I)
1200 IF D=18 THEN 100
1210 GOTO 1170
1220 END
1230 P01=48
1240 P02=48
1250 GOTO 120
1260 M$="VIT`HAR`VUNNIT`KAMPEN```````"
1270 G=23
1280 GOSUB 1380
1290 GOTO 1140
1300 FOR W=3 TO 22
1310 CALL HCHAR(W,3,32,28)
1320 NEXT W
1330 F1=-1
1340 FY2=-1
1350 F=0
1360 FY=0
1370 GOTO 370
1380 V=16-INT(LEN(M$)/2)
1390 FOR I=1 TO LEN(M$)
1400 SD=ASC(SEG$(M$,I,1))
1410 CALL HCHAR(G,V+I,SD)
1420 NEXT I
1430 RETURN
```

CALL SAY

av Jan Alexandersson

Om du gör ett program med PRATOR så kan det vara lämpligt att det är körbart både med och utan tal. I början av programmet lägger du in
CALL PEEK(-28672,SP). Alla satser med tal skrives sedan IF SP THEN CALL SAY("XXX").

CALL KEY

av Jan Alexandersson

I ett tidigare nummer av Programbiten 84-4 förökte jag beskriva kommandot CALL KEY. Jag ska här ta upp en del andra egenheter i CALL KEY.

PÅVERKAR INPUT OCH ACCEPT

Prova följande lilla program med ALPHA LOCK uppe:

```
100 CALL KEY(5,K,S)
110 INPUT A$
120 PRINT A$
130 CALL KEY(3,K,S)
140 INPUT A$
150 PRINT A$
160 GOTO 100
```

Du ser att varannan INPUT-sats accepterar små bokstäver. Genom att ändra från tangentbord 5 till 3 får du något som liknar VALIDATE i Extended BASIC.

VÄNTA PÅ JA ELLER NEJ

Ett vanligt sätt att skriva detta är följande:

```
100 PRINT "FORTSÄTTA J/N"
110 CALL KEY(0,K,S)
120 IF(K=78)+(K=110) THEN 140
130 IF(K=74)+(K=106) THEN 100 ELSE 110
140 END
```

Samma resultat fås om du skriver CALL KEY(5,K,S).

Om du använder tangentbord 3 kan man skriva samma sak mycket enklare.

```
100 PRINT "FORTSÄTTA J/N"
110 CALL KEY(3,K,S)
120 ON 1-(K=78)-2*(K=74) GOTO 110,140,100
140 END
```

På rad 120 används ett logiskt uttryck som blir 1 om varken J eller N har tryckts. Om K=78 är sannt blir uttrycket 1+1=2 eftersom sannt betecknas med -1.

Min personliga uppfattning är att CALL KEY alltid är bättre i menyer än INPUT så att man inte behöver trycka på (ENTER) också.

AVBRYTA PÅGÅENDE LOOP

En ytterligare användining av CALL KEY är avbrytande av en pågående loop. Följane exempel visar detta.

```
100 PRINT "DET SNURRAR RUNT"
110 CALL KEY(3,K,S)
120 IF S=0 THEN 140
130 GOSUB 150
140 GOTO 100
150 CALL KEY(3,K,S)
160 IF S<1 THEN 150
170 RETURN
```

Om du trycker på tangenten en gång så stannar PRINT-loopen. Tryck en gång till och det snurra igen. Det är viktigt att både S=-1 och S=1 gör att man hoppar till subrutinen på rad 150. Om du inte gör detta är risken mycket stor att du aldrig får stopp på snurran oberoende av hur länge du trycker. Problemet uppstår när du snabbt efter det du tryckt igång vill stoppa snurran.
TEXAS har misslyckats med detta när det gäller LIST i Extended BASIC. Du måste där vänta någon extra rad så att CALL KEY eller motsvarande har passerats en gång efter starten utan att någon tangent tryckts. Jämför t.ex. med mitt DISK KATALOG program i PB 85-1 som alltid fungerar oberoende hur snabb du är.

FRÅGOR OCH SVAR

av Börje Häll

Lite då och då, särskilt i sorteringsprogram, behöver man byta värden mellan två variabler, t ex A och B. Jag har aldrig sett det ske på annat sätt än

```
10 C=A
20 A=B
30 B=C
```

Man måste altså använda en tredje variabel, C. Kan man inte göra bytet utan hjälp av den tredje variabeln?

Det kan man. Ett sätt är den här rutinen, som klarar alla tal.

```
10 A=A+B
20 B=A-B
30 A=A-B
```

En annan rutin är följande, som du kan använda om du har Extended BASIC och variabelvärdena är heltal i intervallet -32768,32767

```
10 A=A XOR B
20 B=A XOR B
30 A=A XOR B
```

I Extended BASIC finns kommandona AND, OR och XOR. Kan man på något sätt programmera dessa i BASIC?

Ja det går.
AND kan programmeras som multiplikation (*).

```
10 IF (A=3)AND(B>-2)AND(C<=12)THEN 123
```

kan programmeras

```
10 IF (A=3)*(B>-2)*(C<=12)<>0 THEN 123
```

OR kan programmeras som addition (+)

```
10 IF (A=3)OR(B>-2)OR(C<=12)THEN 123
```

kan programmeras

```
10 IF (A=3)+(B>-2)+(C<=12)<>0 THEN 123
```

XOR kan också programmeras som addition (+) men summan måste då jämföras med -1

```
10 IF (A=3)XOR(B>-2)XOR(C<=12)THEN 123
```

kan programmeras

```
10 IF (A=3)+(B>-2)+(C<=12)=-1 THEN 123
```

Det går lika bra att använda strängvariabler i stället för numeriska variabler.

Minnesregler: AND * <>0
OR + < 0
XOR + =-1

SPRITE I CIRKEL

av okänd programskrivare

Detta lilla program visar en sprite(A) som går runt i en cirkel.

```
100 REM CIRKSPRITE XB
110 CALL CLEAR
120 CALL SPRITE(#1,65,5,COS(PI/180),COS(PI/180))
130 FOR I=1 TO 360 STEP 5
140 A=I*PI/180
150 CALL LOCATE(#1,100+40*SIN(I*PI/180),100+40*COS(I*
    PI/180))
160 NEXT I
170 GOTO 130
```

VÄNTAN PÅ UPPREPADE TANGENTTRYCKNINGAR

Ofta görs många tangenttryckningar efter varandra när man passerar olika CALL KEY-satser. Det är då lämpligt att testa S<1 i stället för S=0. Du kan annars få iriterande dubbeltryckningar.Exempel:

```
100 CALL KEY(3,K,S)
110 IF S<1 THEN 100
120 A$=A$&CHR$(K)
130 IF K<>13 THEN 100
140 PRINT A$
```

PROGRAMING AIDS III OCH SMASH

av Jan Alexandersson

Det finns ett antal program som kan analysera eller komprimera ett BASIC-program. Exempel på detta är PROGRAMING AIDS III från TEXAS. Det kostar USD 9.95 hos TEXCOMP. För att visa hur det fungerar har jag låtit analysera programmet RITA KURVA i PB 85-2. Det visar vilka radnummer som använder funktioner och variabler samt vilka rader som har hopp till sig. Det är mycket användbart om du skall ändra i ett program som du ej gjort själv. Utskriften blir:

```
RITAKURVA


PROGRAM UNIT <MAIN>


 STRING ARRAYS

 H$()            140   200   660
 T$()            140   260   270   280   290   620
                 660   670


 STRING VARIABLES

 GET$            620   660
 HEX$            230   660


 NUMERIC VARIABLES

 AMPL            380   390   470
 CH              610   620   640   650
 CHAR            300   560   640   650
 DOTK            480   580   600
 DOTR            470   570   590
 I               180   200   220   460   470   480
                 500   700
 J               190   200   210
 KEY             520
 KOL             580   600   610   680
 MAXCHAR         250   560
 N               650   660   670   680
 P               630   660
 PERIOD          400   470
 PKTAVST         410   420   460
 RAD             570   590   610   680
 STA             520   530
 X               600   630   660
 XMAX            320   450   460   470   700
 XSTART          310   450   460   470
 Y               590   630
 YSTART          330   450   470


 BASIC KEYWORDS

 CALL            290   350   360   440   450   510
                 520   610   670   680
 DATA            160   170
 DIM             140
 END             540
 FOR             180   190   460
 GOSUB           490
 GOTO            710
 IF              390   420   530   560
 INPUT           380   400   410
 NEXT            210   220   500
 PRINT           370
 RANDOMIZE       240
 READ            200
 REM             100   110   120   130   150   340
                 430   550
 RETURN          690
 STEP            460


 BASIC FUNCTIONS

 &               660
 ABS             390
 ATN             470
 INT             250   470   480   570   580
 POS             660
 RND             250
 SEG$            660
 SIN             470


 SUBPROGRAMS

 CHAR            290   350   670
 CLEAR           360   440
 GCHAR           610
 HCHAR           450   680
 KEY             520
 SOUND           510


 LINE REFERENCES

   380           390
   410           420
   520           530
   550           490
   690           710

   700           560
```

En annan typ av program är SMASH från OAK TREE SYSTEMS. Pris USD 21.95 hos TENEX. Detta komprimerar ett BASIC eller EXTENDED BASIC-program genom att ta bort REM, skriva flera instruktioner per rad och använda korta variabelnamn. Du spar på detta sätt minnesutrymme och gör ofta programmet snabbare. Här följer RITA KURVA-programmet efter att ha gått igenom SMASH:

```
 140 DIM A$(15,3),B$(159)
 160 DATA 8,4,2,1,9,5,3,1,A,6,2,3,B,7,3,3,C,4,6,5,D,5,
     7,5,E,6,6,7,F,7,7,7
 170 DATA 8,C,A,9,9,D,B,9,A,E,A,B,B,F,B,B,C,C,E,D,D,D,
     F,D,E,E,E,F,F,F,F,F,'
 180 FOR A=0 TO 15 :: FOR B=0 TO 3 :: READ A$(A,B):: N
     EXT B :: NEXT A :: C$="123456789ABCDEF" :: RANDOM
     IZE (0):: C=143-16*(INT(RND*10)=8):: B$(31)="0000
     000000000000" :: B$(32)=B$(31)
 280 B$(33)="00000000000000FF" :: CALL CHAR(33,B$(33))
     :: D=33 :: E=17 :: F=240 :: G=96 :: CALL CHAR(93,
     "003828384470444"):: CALL CLEAR :: PRINT "* SINU
     S-KURVA *": :
 380 INPUT "AMPLITUD (MAX 1): ":H :: IF ABS(H)>1 THEN
     380
 400 INPUT "PERIODER (ANTAL): ":I
 410 INPUT "PUNKTAVSTÅND:     ":J :: IF J<=0 THEN 410
 440 CALL CLEAR :: CALL HCHAR(G/8,1+E/8,33,(F-E)/8)::
     FOR A=E TO F STEP J :: K=INT(G-95*H*SIN(I*8*ATN(1
     )*(A-E)/(F-E))):: L=INT(A):: GOSUB 550 :: NEXT A
     :: CALL SOUND(50,440,0)
 520 CALL KEY(3,M,N):: IF N<1 THEN 520
 540 END
 550 REM
 560 IF D>C-1 THEN 700
 570 O=INT(1+(K-1)/8):: P=INT(1+(L-1)/8):: Q=K-O*8+8 :
     : R=L-P*8+8 :: CALL GCHAR(O,P,S):: D$=B$(S):: T=2
     *Q+(R<5):: D=D-(S<34):: U=S+(S<34)*(S-D)
 660 B$(U)=SEG$(D$,1,T-1)&A$(POS(C$,SEG$(D$,T,1),1),R-
     1+4*(R>4))&SEG$(D$,T+1,16-T):: CALL CHAR(U,B$(U))
     :: CALL HCHAR(O,P,U)
 690 RETURN
 700 A=F :: GOTO 690
```

Ett program som gör både PROGRAMING AIDS III och SMASH är COMPACTOR PLUS från DYNAMIC DATA. Jag har inte sett detta program men det finns att köpa för USD 25.95 från TENEX. ■

PROGRAMBANKEN

PROGRAMBANKEN

Här visas en lista på de tillgängliga programmen i PROGRAMBANKEN. Programmen kan köpas" Priset är uppdelat i två delar, dels en startkostnad som täcker kostnaden för porto och datamediet, dels en kopieringsavgift för varje program.

KASSETT	Startkostnad	35:-
	Kopieringsavgift	15:-
DISKETT	Startkostnad	55:-
	Kopieringsavgift	10:-

Som förut får DU tre program i utbyte när DU skickar in ett program DU SJÄLV HAR GJORT"
Ange om ditt program är skrivet i TI-Basic, Extended-Basic, Assembler, Pascal eller i Forth. Beskriv noggrant vilken utrustning som behövs (även i själva programmet), hur man startar upp och vad som behöver göras för att programmet skall fungera. Skriv även vilken modul som erfordras till programmet. Se KODFÖRKLARING"

Programkategori:
10 = Registerprogram
12 = Ordbehandling
14 = Ekonomi
20 = Regression, Kurvanpassning
21 = Statistik, Varians
29 = Statistik, Sannolikhet
30 = Linjär algebra
39 = Allmän matematik
65 = Elektronik
78 = Astronomi
90 = Programhjälpmedel
91 = Spel
92 = Utbildning
96 = Musik
97 = Demo
99 = Övrigt

10 ** REGISTERPROGRAM
01101002E --- Adressregister
11103069 --- Filläsning på Kassett
11103072 --K Snabbfil till Personal Record Keeping modulen
14103045 -DX Mini-Reg, Utman.84/3
15103092 -DX Tel-Ad, Telefonadresser

12 ** ORDBEHANDLING
10121010E --C Ordbehandling för TP-printer.
01121001E --P Word-processor
11123064 --- P:TEXTIN
11123065 --- P:TEXTUT, Lista P:TEXTIN filer
14123039 -PM Nova-Verba

14 ** EKONOMI
07141001H --X Annuities

20 ** REGRESSION, KURVANPASSNING
01201001E --- Uträkning av en linjes ekvation efter ett antal punkter fördelade kring en linje
01201002E --- Uträkning av riktningskoeff.

21 ** STATISTIK , VARIANS
06211001E --C Beräkning av standardavvikelser

29 ** STATISTIK, SANNOLIKHET
09291001H --- Statistiska kombinationer etc.

30 ** LINJÄR ALGEBRA
06301001E --- Matris invertering/multiplikation
14303050 --X Lösning av linjära ekvationssystem med Gauss-eliminationsmetod

39 ** ALLMÄN MATEMATIK
01391001E --- Integral, derivata, andragradsekv. reella rötter, cp analys, definiera chars, 3-D plot, fakultet och primtalstest
09391003H --- Differential eq.
11393071 --X Rita sinuskurva
14393051 --X Math-Pac, Integration, Newton-Raphson, Intervallhalvering, Romberg-integration

65 ** ELEKTRONIK
01651001E --- Beräkning av komponentvärden för resistiv parallellkoppling, kond-densator i serie, resonans för spole och kondensator, omvandling frekvens - våglängd, ohm's lag, uträkning av antennlängd

78 ** ASTRONOMI
01781001E --- Beräkning av geostationära satelliters positioner.
15783000 --- Trippelpunkten på 99:an, Tyngdpunkten till en sfärisk triangel

90 ** PRAKTISKA PROGRAMHJÄLPMEDEL
06901003E --X Sortering och utskrift från flera disketter av upp till 300 program
06901004E --- Diskkatalog med utskrift
08901006E -XE Utskrift av titelsida för ex. listningar
10901008E --- Very large charachters
11901011S --- Stapeldiagram
07902002 --P Banner, stor text utskriven på printer
11903063 DPX Lista Basicprogram med Basicprogram
11903066 --X PGM LIST59, Lista TI59 program på TI99/4A
11903067 --- Screen-Saver, spara skärmen
11903068 --- Testbild för Din Dator
11903075 -DX Lista DIS/VAR Filer
12901054 -JX Color-Draw, Ritprogram med färger
12903055 --- Screen-Ram
13903024 --X Grafik Exklusive, Kassettbaserat
13903022 -DX Fil-Data-Editor, för DIS/VAR80
13901057 --X Cursor, ändra utseendet på cursorn
13901058 --- Ensa-komma
13903060 -DX List28, Lista program 28tkn/bredd
13903061 -PX ASCII-koder för olika tecken-uppsättningar
13903062 -DX P:TILLCOMP, överför P:TEXTIN filer till Companion format
14903034 --- Teckendefinierare
14903042 --- Kassettinnehåll
14902052 --X Färg-Editor
14901053 --X Talkonvertering, Hex-Dec-Binärt
15903005 --- Baskonvertering, Hex-Dec-Bin
15903098 -JX Tommy Eriksson's Sprite-Maker
15903094 -JX Seppo Huikuris Sprite-Maker
15903076 DPX Listprogram, Listar Basicprogram
15903077 -SX Trunkering, skapa egna ord med trunkering
15903088 -EX List-Size, listar storleken på dina program
15901089 -DX Merge/Read, listar DIS/VAR163 filer
15901090 -EX Mg/Peeker, listar minnes-adresser
15903093 -DX Char-Adapt, Teckenändring, lagras som DIS/VAR80 filer
15903095 DEX Load-Assembler-Bas, omformar Assembler program till Extended-Basic program
15903001 EAD Disassembler, en fullständig disassemblerare
15903002 EAD K:INPUT, subprogram för inmatning av data från tangentbord
16903006 -DM Diskkatalog i Textmode

91 ** SPEL
02911002E --X Prickskytte med kanon mot en kyckling
02911003E --- Startrek, textadventure
02911006E --- Othello.
02911007E --- Robotjakt.
02911008E --- 15-spel.
02911009E --- Tärningsspel.
02911011E --- Keno
03911013E -+D Starguard
03911014E --- Miner
03911015E --C Yahtzee
03911016E --X 3D Tic-tac-toe
03911017E --- Black-Jack II
03911018E --X Not one
03911019E --X Datorpoker
03911024E --- Fyra i rad (luffarschack)
04911025E --- Startrek
04911026E --X Hjälp kycklingen över vägen
04911027E --- Katapult
01911028H --- Car driver
01911029H --- Animals
06911032E --- Vem skjuter den sista roboten
06911034H --X L-game
07911035H --X Wari
07911036H --- Ta dig fram i en osynlig labyrint.
07911037 --X Eliza
07911039H --X Teckenjakt
07911040H --X Black jack
07911041 --X Space battle
07911043 --- Enarmad bandit
08911039E --X Sprite-jakt
08911046E --X Damspel.
08911049E --X Swords & sorcery, textadventure
08911051E --X Ägg-fångst
09911052E --- Deep space
09911054H --- Planetary lander.
09911056H --X Lunar lander.
09911057F --- Towers of Hanoi.
09911058E --X Breakout, bollspel squashtyp.
09911059H --J Ritprogram i olika färger.
09911060H -JX Treasure hunt, labyrintspel.
10911061H --X Kermit
10911062H -JX Find the gun
10911063E --- Matematikspel.
10911065E --- Camel, adventureritt i öknen
10911008E --- Colorfraktions, matematikspel
06912004E --- Slalom
06912005E --- Killer
09912008F --X Asteriod (tal).
09912009 --- Battlestar
11911055S --X Hängning
11917001S -JX Inkräktarna
11917002S -JX Päron
12911001S --- One Check, brädspel
14911037E --X One Check II
12913002 --J Jockes Hostility
12913004 --- Giljotinen, ordlek
12913007 --- Tjugoett
12913008 --- Totto
12913009 --- Octopus
12913010 --- Tipsservice
12913012 --- The Hunt of Aladin
12913013 --- Bowling
12913014 --- Bilrally i Norge
12913015 --- Masken
12913016 --- Djungelfight
12913017 JXB Snake, Ormen
12913018 JXB Zombie
13913019 --J Kami-Kaze
13913020 --X Ubåt
13913023 --- Jumping Jack
13913024 --- Astrostorm
13913025 --- Goblin
13913026 --- Identify the States
14913035 --X Skroting
14913036 -JX Starport 99
14913040 --- Blockman
14913044 --X Mastermind
14911046E --X Pricka flygplan
14912047H --X Rymdslaget
14911048E --X Spacegame
15913079/1 --X Adventurespel, Gör dina egna Adventure-spel, (1-9)
15913080/2 --X Adventurespel
15913081/3 --X Adventurespel
15913082/4 --X Adventurespel
15913083/5 --X Adventurespel
15913084/6 --X Adventurespel
15913085/7 --X Adventurespel
15913086/8 --X Adventurespel
15913087/9 --X Adventurespel

92 ** UTHILDNING
04921001E --- Enkla räkneövn. med de fyra räknesätten
04921002E --- Kemifrågor
04921003E --X Relative IQ-test
04921004E --- Algebra
04921005E --- Projectile problems, beräkn.-spel
01921006H --X Arvsanlag
07921007E --- Träning i dåtid (imperfekt)
10921008E --- Räkneträning med bråkdelar
11923074 --X Fransk glosträning
13923031 --X Byggnadsmekanik
13923029 --X Huvudräkning
13923030 --X Glosträning, kassettbaserad

96 ** MUSIK
08961001E --X Killing me softly
05962001 --C A Strauss walz
05962002 --- Never on a sunday
05962003 --X Berceuse
05962004 -XC Beethoven ñ
05962005 --X Serenade
05962006 +DX Amazing graze
05962007 --- Beethoven opus 27
05962008 +DX Time in a bottle
05962009 +DX Light up my life
05962010 --N Bummble-boogie
06962011 -+D Fiddler on the roof
07962013 --- Looking through you
08962014 --X Chopin op 11 ñ
06962001 --- The pink panther
06962006 --- Let me call you sweetheart
12963056 --- Tremolo
14963033 --- TI-Keyboard
14963043 --- When the Saints

97 ** DEMO
07971003H --- Demonstration av olika tecken-storlekar
11973070 -EX Demoladdning av Assembler program
13973059 --- Korsord
14973038 --X Mönstergrafik
15973097 --- Snabbare grafik med Ti-Basic
15973099 --X Doppler-effekt
15971091 -EX Mg/Text, Textmode
15973003 --M TextMode i TI-Basic
15973004 --M Auto-Sprite i TI-Basic

99 ** ÖVRIGT
01991001E --X Släktforskningsregister
01991002H --- Utskrift av månadskalender
09991003F --- Biorythm.
01992001 --- Månkalender
11993073 -PX Tipsrad
13993028 --X Biorytm II
13993021 --P Etiketter
14993032 --X Biorytm III
14993041 --- Resultatlista, Slalom
14993049 --X Tips-system
15993078 -EX Automatisk uppringning
15993096 --- Dagräkning med almanacka
07991005H --- Morse
10991007H --- Lotto

KODFÖRKLARING

De fyra första siffrorna är interna beteckningar (diskettnummer, programkategori).
Femte siffran anger ursprungsland:

1 = England, USA	2 = Holland
3 = Sverige	4 = Frankrike

De tre sista siffrorna är ett internt löpnummer. Bokstaven som finns sist anger vilket språk som används i programmet.

E = Engelska	F = Franska
H = Holländska	S = Svenska

Eventuella bokstäver efter programnumret och bokstaven talar om vilken utrustning som krävs för att kunna använda programmen som de är:

C = CALL FILES (1) måste användas om man har diskettenhet.	
D = Diskettenhet	E = Minnesexpansion
F = Forth	A = Editor/Assembler
M = MiniMemory	X = Extended-Basic
P = Printer	S = Speech Synthesizer
J = Joystick	(pratorn)
T = Terminal Emulator II	K = Personal Record Keeping

Ett + anger att programmet är uppdelat i flera delar.