

PROGRAM

nittinen

BITEN

FORTH spalten

REDAKTÖR: Lars-Erik Svahn

25-2

LT- SUPPORT

Innehåll

Redaktören, Ordföranden	2
Rita kurva i basic	3
Utmaningen	4-8
Litet amerikanskt skvaller	8
Personal record keeping-filer från basic	9-11
TI-59 Stefan Bolzmann...	12-13
Roten ur med 120,140 och 155 siffror	14-16
LT-support	17-19
FORTH-spalten	20
Oj-då!	20
Gameforth	21
Train	21
FORTH 'BG	22-23
Bokrescensioner	24-25
Brev	26
Tipsrad	26
TOTO-G	27
Programbanken	28

ISSN 0281-1146

Den nye redaktören har ordet.

För hela nummer 1 och det mesta av nummer 2 har Lennart Lindberg arbetat som redaktör. Från och med slutarbetet med nummer 2 kommer jag (Jan Alexandersson) att fungera som den redaktör som kommer att sammanställa och redigera dina insända bidrag. Om det kommer in mera bidrag från dig kan vi göra en ännu bättre tidning. Utskrift av allt 99-material i tidningen kommer att göras av Tony Rogvall. Liksom tidigare kommer den oundgängliga Claes Schibler att sköta monteringen av alla texterna och sköta kontakterna med tryckeriet m.m.

Ordföranden har ordet.

När det här skrivs har jag ganska nyss läst i Mikrodatorn ett par annonser, där nya användarföreningar för TI 99 söker sympatisörer. Jag avser ta kontakt så fort jag kan för att få till ett samarbete. Jag tror att vi alla får ut mest av att gå samman. Samtidigt finns det förstås särintressen. Jag hoppas vi kan ta hand om dem inom föreningen.

Ett sätt att ta hand om särintressen är SIG-arna. SIG FÖRTH har varit först ute med att formera sig och haft ett trevligt möte i Stockholm, där intressanta program bl a demonstrerades. Bo Carleö, t ex, hade skrivit ett mycket elegant grafikprogram som vi nog kommer att se mer av - bl a här i spaltarna.

Jag ser gärna att fler SIG-ar kommer igång. För att hjälpa till med detta kommer det allmänna medlemsmöte som vi arrangerar den 14 september i Stockholm på eftermiddagen - det är en lördag - att ägnas kontakter mellan personer med SIG-intressen. Vi avser se till att mötet ger möjligheter att träffas och knyta kontakter. Separat kallelse planeras till i augusti. Vik dagen!

Vi har ett bra tag nu haft en annons om en datavisionsmodul i tidningen. Programmet är nu klart och fungerar! Michael Öhman, som gjort det, har demonstrerat med framgång. Just nu finns programmet på disk. Bränningen av erforderligt PROM och annat tekniskt arbete kommer nu snart att ske, varefter modulen troligen kommer att finnas tillgänglig i handeln. Den går i vart fall att köpa genom föreningen. Vi tror att användbarheten av 99-orna på det här sättet på ett mycket intressant sätt utvidgas.

Jag noterade i nummer 85-1 att jag inte klarade av att tillfredsställande sköta redaktörssysslan fortsättningsvis. Raskt dök en redaktör upp, nämligen Jan Alexandersson. Jag hälsar honom mycket välkommen i detta värv och önskar honom lycka till. Han är dessutom mycket intresserad av Basic/Extended Basic som vi ju velat ägna mera utrymme åt och jag tror vi kan räkna med att han kommer sköta just den sidan av saken särskilt väl.

Låt mig avsluta med att berätta om en liten lækkerhet - er grann katalog över 99-tillbehör och hela 99-or från en säljare i USA. Jag tänker på Triton Products Company med adress P.O. Box 8123, San Fransisco, CA 94128, USA. Skriv efter katalogen om Du vill ha en fräsch förteckning över intressanta produkter. Färg, mycket bilder, prisuppgifter, noggranna produktbeskrivningar. Skicka med svarskuvert med rätt frankering (fråga på posten). Katalogen väger c:a 90 gram.

I samband med kallelsen till medlemsmötet i september planerar vi en enkät bland medlemmarna om maskininnehav och olika intressen. Svara på den! Vi kan göra mera för Dig om vi vet vad Du är intresserad av.

Lennart Lindberg, ordförande.

I REDAKTIONEN

Redaktör	Jan Alexandersson
Utmaningsredaktör	Anders Persson
Forth-redaktör	Lars-Erik Swahn
Programförmidlare	Göran Nygren
Allt-i-allo	Claes S

Ännu ett nummer av PB hand-assemblerat av Claes

Föreningens och redaktionens adress:

```

Föreningen Programbiten      ++++++
c/o Schibler                  +
Wahlbergsgatan 6 1 tr ned    + DATAINSPEKTIONENS +
121 46 Johanneshov           + LICENSNUMMER         +
                               +                           +
Postairo 19 83 00 - 6         + 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.

För kommersiellt bruk gäller följande:

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

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	400:-	
(600 för ICKE medlem)!!		
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:-
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:-

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

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 II-57, II-58, II-59		20:-
Föreningens programmeringsblanketter (II-59), olika typer, block om 50 blanketter (se PB 83-1 sidan 30) per block		11:-
Patenthandlingar II-59		25:-
40 st tomta magnetkort med plånbok		150:-
10 magnetkortslånbok		10:-

Rita kurva i Basic

av Jan Alexandersson

Det är nog många som saknar ett PLOT-kommando i BASIC. I följande program ritas en sinuskurva på skärmen genom att ett tecken (ASCII 34-159) placeras på aktuell RAD(1-24) och kolumn(1-32). Inom varje teckenruta skapas ett lokalt koordinatsystem X(1-8) och Y(1-8). Varje tecken definieras med en sträng om 16 hexadecimala siffror (0-F). En X-rad om 8 punkter kodas med två hexadecimala siffror (se vidare CALL CHAR i handboken). Om X<5 gäller den vänstra av dessa. $P=2*Y+(X<5)$ anger då vilken av dessa 16 hexadecimala siffror i strängen som är aktuell eftersom det logiska uttrycket är -1 när det är sant.

Programmet läser vilket tecken som finns på aktuell rad och kolumn och väljer ut rätt position i strängen. Det gäller sedan att överlagra den nya punkten på den tidigare grafiken. Detta kan göras med OR i Extended Basic. För X=1,2,3,4 blir en ny punkt 8,4,2,1. Detta ska sedan överlagras på den tidigare grafiken. Följande rader i Extended Basic gör detta:

```
110 FOR B=3 TO 0 STEP -1
120 CH=A OR (2^B)
130 PRINT TAB(20-B*3);CHR$(48+CH-7*(CH>9))
140 NEXT B
150 PRINT
160 NEXT A
```

För att kunna köra i vanlig BASIC har denna tabell lagrats i H(.). Eftersom BASIC kan ha CHAR upp till 159 medan Extended Basic har upp till 143 bör programmet självt testa om det körs i BASIC eller Extended Basic. Detta kan göras genom att RND läses efter RANDOMIZE(0). Man får olika svar i de två fallen.

Programbeskrivning

150-330 Initiering av variabler
340-420 Inmatning av data om kurvan
430-540 Huvudprogram för kurvritning
550-710 Underprogram för punktgrafik

Använda variabler

H\$(,)	omräkningstabell för punktöverlagring
T\$(,)	teckendefinition
I	loop-variabel
J	loop-variabel
HEX\$	hexadecimala siffror i valörordning
MACCHAR	högsta tecken som får användas
CHAR	högsta tecken som tidigare använts
XSTART	start i X-led (1-256)
XMAX	slut i X-led (1-256)
YSTART	start i Y-led (1-192)
AMPL	sinuskurvans amplitud (max 1)
PERIOD	antalet perioder av sinuskurvan
PKTAVST	avstånd efter X-axeln mellan ritade punkter som normalt är 1 men kan vara både större och mindre
DOTR	aktuell Y-koordinat för PLOT
DOTK	aktuell X-koordinat för PLOT
KEY	tangent vid CALL KEY
STA	status vid CALL KEY
RAD	rad i Y-led (1-24)
KOL	kolumn i X-led (1-32)
Y	lokal variabel i Y-led inom teckenruta
X	lokal variabel i X-led inom teckenruta
CH	tidigare tecken på RAD/KOL
GET\$	tidigare teckendefinition
P	position i den hexadecimala teckensträngen
N	tecken som skall skrivas av PLOT

Möjliga programvarianter

Om andra kurvor ska ritas kan X-axeln flyttas eller en Y-axel ritas på motsvarande sätt med tecken 34. Då måste även skärningen mellan X och Y ha ett eget tecken t.ex. 35.
I Extended Basic kan CALL CHARPAT användas i stället för T\$(,) och OR i stället för H\$(,).

```
100 REM RITA KURVA BA/XB
110 REM JAN ALEXANDERSSON
120 REM VERSION 1985-01-14
130 REM
140 DIM H$(15,3),T$(159)
150 REM INITIERA
160 DATA 8,4,2,1,9,5,3,1,A,6,2,3,B,7,3,3,C,4,6,5,D,5,
170 DATA 8,C,A,9,9,D,B,9,A,E,A,B,B,F,B,B,C,C,E,D,D,D,
180 FOR I=0 TO 15
190 FOR J=0 TO 3
200 READ H$(I,J)
210 NEXT J
220 NEXT I
230 HEX$="123456789ABCDEF"
240 RANDOMIZE (0)
250 MAXCHAR=143-16*(INT(RND*10)=8)
260 T$(31)="0000000000000000"
270 T$(32)=T$(31)
280 T$(33)="00000000000000FF"
290 CALL CHAR(33,T$(33))
300 CHAR=33
310 XSTART=17
320 XMAX=240
330 YSTART=96
340 REM DATAINMATNING
350 CALL CHAR(93,"00382838447C4444")
360 CALL CLEAR
370 PRINT "SINUS-KURVA *":
380 INPUT "AMPLITUD (MAX 1): ":AMPL
390 IF ABS(AMPL)>1 THEN 380
400 INPUT "PERIODER (ANTAL): ":PERIOD
410 INPUT "PUNKTAVSTAND: ":PKTAVST
420 IF PKTAVST<=0 THEN 410
430 REM HUVUDPROGRAM
440 CALL CLEAR
450 CALL HCHAR(YSTART/8,1+XSTART/8,33,(XMAX-XSTART)/8)
460 FOR I=XSTART TO XMAX STEP PKTAVST
470 DOTR=INT(YSTART-95*AMPL*SIN(PERIOD*8*ATN(1)*(I-XSTART)/(XMAX-XSTART)))
480 DOTK=INT(I)
490 GOSUB 550
500 NEXT I
510 CALL SOUND(50,440,0)
520 CALL KEY(3,KEY,STA)
530 IF STA<1 THEN 520
540 END
550 REM PLOT
560 IF CHAR>MAXCHAR-1 THEN 700
570 RAD=INT(1+(DOTR-1)/8)
580 KOL=INT(1+(DOTK-1)/8)
590 Y=DOTR-RAD*8+8
600 X=DOTK-KOL*8+8
610 CALL GCHAR(RAD,KOL,CH)
620 GET$=T$(CH)
630 P=2*Y+(X<5)
640 CHAR=CHAR-(CH<34)
650 N=CH+(CH<34)*(CH-CHAR)
660 T$(N)=SEG$(GET$,1,P-1)&H$(POS(HEX$,SEG$(GET$,P,1)),X-1+4*(X>4))&SEG$(GET$,P+1,16-P)
670 CALL CHAR(N,T$(N))
680 CALL HCHAR(RAD,KOL,N)
690 RETURN
700 I=XMAX
710 GOTO 690
```

UTMANINGEN!

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

Den här gången finns det både problem, lösningar, teoretiska halvlösningar och antydningar till framtida lösningar i Utmaningen. Blandad kompott, med andra ord.

Anledningen till att jag berättar om program som inte är färdiga än är att det kanske kan vara en tröst för ett tigerhjärta att veta att det som du vill ha åtminstone är påtänkt någonstans. Dessutom kan jag väl inte sticka under stol med att jag ju ska skriva om något under alla förhållanden.

Det handlar mest om assembler den här gången. Tyvärr hör de som har "basutrustningen" praktiskt taget aldrig av sig. Men det visar sig genast när någon faktiskt tar kontakt att vederbörande har massor med frågor som jag ibland kan besvara direkt. Om inte brukar jag sätta igång någon utredning, själv eller hos någon annan.

Slutsats: Du kan få hjälp med nästan vad som helst, men inte om du inte bekvämar dig till att fråga

Nog tjatat" Nu börjar det roliga.

55. MINI-ASSEMBLER

Om du har Extended BASIC och minnesexpansion till din 99:a, kan du utnyttja assemblerprogram. Detta gäller även om du bara har kassettbandsspelare för lagring av dina program. Ett gott exempel på detta är LT-Writer, som utnyttjar LT-Support för att få tillgång till ett effektivare lagringssätt än vad BASIC kan erbjuda. LT-Support är just ett assemblerprogram på kassett.

Problemet är att få in assemblerprogrammet i minnet. I LT-Writer finns det en massa CALL LOAD, som direkt laddar in koderna för assemblerprogrammet. Men att manuellt räkna ut vilka koderna ska vara, det är inte skojigt. (Det har jag inte gjort heller.) Särskilt inte för lite längre program. Då bör man i stället använda datorn till detta arbete. Det finns en utmärkt uppfinning, kallad assembler, som gör just detta. Den som hör till Editor/Assembler-modulen är ett exempel. Men den går ju inte att använda utan både modulen ifråga och disksystem.

Lars-Erik Svahn, och fler med honom, för den delen, tycker att man borde ha en assembler skriven i X-BASIC. Det finns några sådana på marknaden, men Lars-Erik har kommit på en lite annorlunda tanke. Att det går att skriva namngivna underprogram i X-BASIC torde väl inte ha undgått någon som har läst Utmaningen flitigt. Har du inte det kan den här spalten i numren 84-3 och 84-4 rekommenderas. Genom att skriva underprogram med namnen MOV, ANDI, BLWP etc. kan man få en procedur för varje instruktion som processorn har. Om man då hänger på några parametrar som anger adresser och/eller data, kan det se ut (nästan) så här:

```
CALL MOV(RO,'1234,A)
```

Raden ovan motsvarar alltså MOV RO,'1234

A är adressen där instruktionen ska lagras. Den skall räknas upp av underprogrammen, beroende på hur stor plats instruktionen behöver. Vid körning av BASIC programmet kommer assemblerprogrammet att laddas i minnet på angivna adresser. En klar fördel med den här varianten, jämfört med den radassembler som finns till Mini Memory, är att det är mycket lätt att ändra programmet om man har glömt något. Sätt in några nya BASIC rader bara, och du har lagt till några instruktioner.

Dessutom är det inte katastrof om det inte fungerade som det skulle. Genom att lagra BASIC programmet på kassett innan assembleringen och provkörningen startar har du ju källkoden i behåll, även om det gick galet när assemblerprogrammet startades. In med BASIC programmet från kassetten igen, och vips: Återuppståndet från de döda programmens domäner

I vanliga fall hade jag avslutat det problemet här och övergått till nästa. Men tack vare det telefonerande som jag nämde i PB 85-1 har jag redan fått en sådan här assemblerator skriven, och kan alltså presentera även lösningen till problemet med en gång" Ibland är vi snabba...

Det är Lennart Thelander i Helsingborg som tagit sig an Lars-Eriks Utmaning. Det faktum att Lennart uttryckligen bad om en hoppas jag visar för er att det faktiskt finns folk som är intresserade av att tackla andras problem.

Mini-assemblern, som i vanlig ordning finns i anslutning till den här artikeln, innehåller ett SUB program för varje instruktion, och dessutom en del andra. Det är det "subigaste" program jag har sett hittills...

Observera att det här programmet fungerar bäst om assemblerkoden lagras i 8-k delen av minnesexpansionen. Det är ju också det normala med program som laddas in från disk med CALL LOAD i Extended BASIC.

Förutom de SUB program som utgör själva instruktionerna finns det ett antal andra som är bra att ha. De fungerar så här:

VALUE TO HEX: Omvandlar ett tal till ett fyrsiffrigt hexadecimalt tal. Resultatet blir alltid rätt så länge -32768 <= indata <= 65535.

HEX TO VALUE: Motsatsen till ovanstående.

BYTE: Laddar in en byte i minnet och räknar upp laddadressen. Du kan även använda den vanliga CALL LOAD. Se exemplet i början av programmet.

STARTADR: Ger adressen till första lediga byte i 8-k delen av minnet.

INITIALIZE: Som STARTADR, men gör först CALL INIT för att rensa minnet från eventuella gamla program.

END: Markerar hur mycket minne som använts hittills. Anropas efter programmet för att datorn ska veta hur mycket minne som använts.

DEF: Lagrar en så kallad extern definition i REF/DEF tabellen. Därigenom kan X-BASIC hitta assemblerprogrammet med en vanlig CALL LINK.

Intressant är att den här lösningen med STARTADR, END och DEF gör att det går utmärkt att ladda in ett stort program genom att skriva in delar av det till Mini-assemblern, och köra dessa delar efter varandra. Genom att avsluta varje del med CALL END, och börja nästa med ett anrop av STARTADR, talar delarna om för varandra hur mycket minne som hittills är använt. Det finns också en del procedurer som används vid kodningen av programmet.

REG: Ger ett tal som motsvarar den registerbeteckning som skickas in. REG fungerar så att register valfritt kan anges med eller utan bokstaven "R".

GENADR: Omvandlar en generell adress till rätt opkod.

SPLIT: Delar ett 16-bitars tal i två bytes.

DLOAD: Dubbel LOAD. Laddar ett 16-bitars värde och räknar upp laddadressen.

FORMAT1, FORMAT2, FORMAT3, FORMAT4, FORMAT5, FORMAT6, FORMAT8 och FORMAT8B assembler instruktioner med dessa format. Vissa instruktioner har så likvärdiga format att de slagits samman till en procedur.

Hur skriver du då ett program med hjälp av Mini-assemblern? Eftersom det är ganska svårt att förklara har jag valt att visa programmet med ett exempel inlagt i början. Den egentliga assemblern börjar inte förrän på rad 30000. De inledande raderna visar hur programmet Chime (E/A manualen, sid 321) kan matas in. Titta på de olika instruktioner som finns där och på de subprogram som definierar sådana som inte används i Chime, så kan du nog lista ut hur programmet skall användas.

OBS" På grund av tidsbrist är inte alla tänkbara varianter av det här BASIC programmet testade. Ingen garanti för att det är helt felfritt, alltså.

Om du matar in programmet från listan är det lämpligt att göra så här:

Börja med rad 30000 och fortsätt till slutet. Lagra detta på kassett. Det blir då den assembler som du kan använda till dina egna program. Fortsätt med de första raderna ner till 550. Lagra allt detta på kassett också. Det är viktigt, eftersom Chime måste avbrytas genom att datorn stängs av. Om du vill kunna "plinga" fler gånger kan det ju vara bra att ha källkoden kvar.

Skriv sen CALL LINK("CHIME") och lyssna.

NÄRKONTAKT MED KASSETTBANDSPELAREN

Som alla vet finns det en färdig rutin för kassetthanteringen som används praktiskt taget överallt. Datorn skriver ledtext på skärmen och förväntar sig att du trycker på ENTER då och då. Men ibland vill man t.ex. lagra flera filer efter varandra på kassetten utan att behöva ingripa manuellt mellan dem. Det finns ett flertal tillämpningar på detta. En är inom föreningen, när flera program ska överföras från diskett till kassett.

Problemet är alltså: Hur komma åt kassettrutinen utan att få med den vanliga texten osv?

Efter att ha diskuterat det här med några olika personer bestämde jag mig för att börja forska i hur det skulle gå till. Jag visste redan innan att det skulle gå, eftersom p-systemet gör det.

Den vanliga kassettrutinen är skriven i GPL. GPL tar hand om utskriften av ledtext, startar och stannar motorn, väntar på att ledarbandet ska ta slut etc. Men själva in- och utmatningen är skriven i assembler. Från GPL anropas den med en instruktion som heter I/O. Det är en av de speciella GPL instruktioner som kan användas till flera olika saker. Den klarar av ljud, manipulering av CRU bitar och, just, kassettagring. Genom att ge I/O lämpliga parametrar kan man anropa rutinerna WRITE, READ och VERIFY.

Rutinerna för kassett I/O finns alltså i ROM, i GPL interpretatorn. Det gällde bara att hitta dem, och komma på hur de skulle anropas. För att klara ut det satte jag igång med att kartlägga praktiskt taget hela GPL interpretatorn. Det mesta av den är tämligen enkelt, men det finns några delar som är ordentligt tillkrånglade.

Nu är det så att det finns ett antal mer eller mindre olika operativsystem till 99:an. Någon tidning påstod sig ha hittat sex olika versioner. Därför kan det tänkas att adresserna som jag har letat reda på inte är lika i din maskin. Tursamt nog är GPL interpretatorn i många och mycket baserad på tabeller, där adresserna till olika delar finns. Förhoppningsvis ligger tabellerna på samma ställe i minnet, oberoende av version.

I min maskin avkodas instruktioner med GPL format 1 och 8 (I/O har 8) på adressen 00C4. En tabell på 0C9E utnyttjas. 0CC8 ger adressen till I/O instruktionen. I min maskin är den 05C8. Där används ytterligare en tabell för att avkoda olika typer av I/O. Tabellen för det finns vid 0CEC. Från 0CF4 anges adressen för WRITE, READ och VERIFY till 1346, 142E och 1426 i min maskin.

Sammanfattningsvis så här:

WRITE: 1346
READ: 142E
VERIFY: 1426

Vid anrop av de här rutinerna måste följande iakttas:

GPL workspace (83E0) ska användas.

Det ska finnas två words med beskrivning på bufferten någonstans i minnet. Först ska antalet bytes att lagra eller läsa komma, sedan en pekare till bufferten.

R1 i GPLWS ska peka på den här beskrivningen.

Eventuella fel anges i GPLST (837C). Galenskapen i allmänhet indikeras med Condition biten, medan biten längst till höger anger typen av fel. En nolla där innebär "NO DATA FOUND", en etta "ERROR DETECTED IN DATA".

Återhoppet är trassligt. När kassettrutinen är färdig hoppar den nämligen till läsning av nästa GPL instruktion. Det är inte alls konstigt, utan beror bara på att den i normala fall kom därifrån. Men för att det ska gå att anropa READ etc. från ett assemblerprogram måste man också återvända dit. Enklarest är då att utnyttja det faktum att en XML instruktion exekverades när datorn började köra assemblerprogrammet. Den kan man ju använda igen. Det gäller bara att ändra adressen, eftersom XML normalt endast kan anropa en externt definierad (DEF) label.

Om E/A modulen används finns adressen vid 2002. För X-BASIC gäller 2000. Det här är alltså adressen till den rutin som letar upp en label i REF/DEF tabellen och anropar den som önskades. Genom att tillfälligt ändra på den adressen kan man få XML att hoppa var som helst.

För att nu XML instruktionen ska exekveras en gång till måste GPL adressen backas två steg. XML tar nämligen två bytes.

Efter allt detta skulle jag misstänka att ett exempel kan vara på sin plats.

Nedan visar jag hur ett anrop av lagringsrutinen kan göras. Det här är tänkt att ingå i ett program som ska lagra en del av minnet på kassett. Variablerna START och STOP anger mellan vilka adresser data som ska lagras finns. (Kanhända Tony Rogvall känner igen det här?) När dessa har fått sina värden behövs bara en BL 'SAVE för att lagringen ska ske.

REF	VMW,GRMRA,GRMWA	
STATUS EQU	>837C	GPL status byte
GPLWS EQU	>83E0	GPL workspace
IOWRIT EQU	>1346	Adresser till kassettrutiner
IOREAD EQU	>142E	
IOVRFY EQU	>1426	
XMLADR EQU	>2002	Adress som XML använder
VDPBUF EQU	>1000	Buffert i VDP RAM
USERWS EQU	>20BA	Workspace med E/A modulen
MOTOR1 EQU	22	Motorkontroll för CS1
START DATA	>0000	Adressintervall att lagra i exemplet
STOP DATA	>0000	I/O argument: Antal bytes
SAVARG DATA	>0000	Buffert pekare
	DATA VDPBUF	

Här kommer den subrutin som sköter om själva lagringen. Den räknar ut hur många bytes det handlar om, flyttar dessa till bufferten och gör de manövrer som behövs för att lagringen ska vara möjlig. Men den väntar inte på att ledarbandet ska ta slut. Avsikten är ju bara att visa hur man kan anropa GPL:s I/O, inte hur resten av programmet lämpligen ska ordnas.

SAVE	CLR R12	För att få rätt CRU adress	
	SBO MOTOR1	Starta motorn CS1	
	MOV 'STOP',SAVARG		
	S 'START',SAVARG	Räkna ut antalet bytes	
	LI R0,VDPBUF	Flytta data till VDP buffert	
	MOV 'START,R1		
	MOV 'SAVARG,R2		
	BLWP 'VMW		
	LI R0,SAVARG	Lagra en pekare till buffert-	
	MOV R0,'GPLWS+2	beskrivningen i R1 (GPLWS)	
	MOVB 'GRMRA,R0	Backa GROM adressen två bytes	
	SWPB R0		
	MOVB 'GRMRA,R0		
	SWPB R0		
	AI R0,-3	Minska med ett extra för att	
	MOVB R0,'GRMWA	korrigera felet som blir vid	
	SWPB R0	läsningen. Lagra sedan nya	
	MOVB R0,'GRMWA	adressen.	
	MOV 'XMLADR,R3	Ändra XML tabellen för att	
	LI R0,SAVE1	fixa ett återhopp till SAVE1	
	MOV R0,'XMLADR	Spara gamla adressen i R3	
	LWPI GPLWS		
	B 'IOWRIT	Anropa I/O write	
SAVE1	LWPI USERWS	Kommer tillbaka hit	
	MOV R3,'XMLADR	Återställ XML adressen	
	SBZ MOTOR1	Stanna motorn CS1	
	B *R11		

XML ändrar inte GPL status. Därför finns eventuella felindikeringar kvar i GPL status byte (837C).

När du ska anropa READ fungerar det på samma sätt. Bufferten innehåller ju ingenting vettigt före anropet. Antalet bytes anger hur stor bufferten är. Om en alltför liten buffert finns får man ett fel av typen "ERROR DETECTED IN DATA".

För VERIFY gäller att bufferten ska innehålla det som förutsättes finnas på bandet. Enklarest är det naturligtvis om du anropar VERIFY omedelbart efter WRITE, men det är inte nödvändigt.

Om du skriver något program som utnyttjar det här kan du väl skicka det till mig. Om det är något intressant kommer det med i Utmaningen.

TECKEN PÅ EGET SATT

Flera av de mer avancerade programmen till 99:an har teckendefinitionerna separerat i en fil på diskett. I och med detta är det lätt att ändra utseendet på tecknen efter eget behag. Ett exempel på detta är de nya versioner av TI-WRITER och MULTIPLAN som föreningen distribuerar (se PB 85-1).

Folke Andersson ville ha svenska tecken i Multiplan. Jag ordnade det, varvid Folke tyckte att jag kunde beskriva hur man gör i Utmaningen. Men det exakta förfarandet beror på vilket program man använder för att ändra på filen. Eftersom det finns flera olika program som går att använda till teckenändringen har jag valt att beskriva principen i stället.

Om du själv vill ändra på tecknen, anser jag att det är enklast att gå in på disketten och ändra där. Då måste du förstas ha tillgång till något program som klarar att läsa disketten, ändra innehållet och skriva tillbaka det igen. Det finns t.ex. ett program, kallat Disk-Patch, som går bra att använda. De som har Pascal kan utnyttja Patch på Utility disketten. Det går att använda Forths editor också, men det är inte lika enkelt. Det går nämligen lättast om man har någon hexadecimal editor.

Det finns fler program som duger, men jag kan inte räkna upp alla här. Låt oss anta att du har något lämpligt. Vi kan också säga att du vill ordna med svenska Å, ä och ö i Multiplan. (Jag gjorde inte det från början, eftersom hakparenteserna används till formler i Multiplan. Men man kan ju tänka sig att utveckla kalkylen med ett teckenset och använda den med ett annat.)

Börja med att formatera en diskett och kopiera över filen MPCHAR till den. I och med att det bara finns en fil på disketten är den lätt att hitta. Leta sedan upp definitionen av vänster hakparentes. Det kan vara lite besvärligt att orientera sig i filen, vilken ju ser ut som en jättelång CALL CHAR. Det är inte alltid så lätt att inse vad en massa siffror och bokstäver ger för tecken, men du kan rita upp det mönster som definieras på ett papper. Efter några rader ser du vad det är för ett tecken. Nu är det lätt att räkna ut var Å, Ä och Ö finns. Varje definition tar upp åtta bytes. Glöm inte att ändra även de små bokstäverna.

Just för Multiplan är det mer komplicerat än vanligt. Multiplan visar ju tecknen inverterat här och där. För att klara det finns hela alfabetet i två versioner, en normal där tecknet är "tätt" och bakgrunden "släckt" samt en omvänd. Därför måste du ändra på två ställen för varje tecken. Å andra sidan kan du utnyttja det här om du bara ändrar det vanliga alfabetet. Då får du svenska tecken för det mesta, men amerikanska om du placerar Multiplans markör över texten.

När ändringarna är genomförda måste filen MPCHAR tillbaka till Multiplan disketten förstas. Annars gör det inte mycket nytta.

Dessa tecken på skrivare!

I det här numret blandas ganska friskt utskrifter från flera olika skrivare. Det här, t ex är skrivet med en Sense pot. Hela Utmaningen, FORTH-spalten och annat är skrivet med Diablo, utrustad med det typhjul som finns redovisat i nr 84-4 sid 16 under beteckningen Scandia Elit 12 (vilket i och för sig är fel beteckning - det är ett specialtypsnitt för optisk läsning). Den stilen ger emellertid en del problem med tecken, som är viktiga i en sådan här tidskrift:

- "number sign" (bräddgården) ser ut som ett spanskt n (ett n med en "kråka" ovanpå)
- alfa-slang ser ut som ett enkelt citationstecken - en "blip"
- utropstecknet har också ändrat skepnad och ser ut som ett litet, svagt citationstecken

Det är faktiskt ett elände med den här förbistringen!
/Red.

DISKETTRUTINER

I Utmaningen har tidigare efterfrågats assemblerprogram som visar en diskettkatalog, kopierar disketter osv. Jag har fått en prototyp på ett katalogprogram, som eventuellt kommer senare. Det är inte så flexibelt än, men modifiering lär pågå.

När det gäller kopiering kan jag rekommendera ett program från Quality 99 Software som heter "the DUPLICATOR". Det kan kopiera en enkelsidig diskett i endast två pass, vilket alltså är mycket bra om man bara har en disk drive. Dessutom kopierar det vad som helst, eftersom det inte läser filer på disketten utan sektorvis. Det enda det inte kopierar är sig självt! Originalskivan kan inte kopieras vare sig med Disk Manager, Forth eller Pascal. Och om man lyckas flytta "the Duplicator" till en annan diskett vägrar den att fungera där. Ganska fiffigt faktiskt.

Nu går det naturligtvis att komma runt det här också, men hur det går till har jag lovat Maurice Swinnen att inte tala om. Det är inte lätt, det kan försäkras.

Herr Lindberg undrade också hur man ska göra det som kallas för "Scratch Record", dvs borttagning av en post mitt i en fil. Det finns en sådan operation, men den klarar inget nu levande diskkontrollkort av.

Om man vill ta bort en post mitt i en fil får man alltså själv, dvs i BASIC programmet eller vad det nu är, flytta alla posterna efter den här ett steg. På så sätt krymper filen ihop. Men det tar ju en förskräcklig tid om filen fr läng.

En annan variant är att utnyttja det faktum att filsystemet tillåter att filer är spridda över flera olika ställen på disketten. Genom att ändra i filkatalogen kan man få datorn att tro att filen slutar där man ville ta bort en post, och sedan börjar igen på något annat ställe. I själva verket fortsätter den omedelbart efter den raderade posten, men det gör ingenting. Nackdelen med det här är att det inte går att dela en fil i hur många småbitar som helst. Därigenom finns det också en gräns för hur många poster som kan tas bort på det sättet.

Det effektivaste torde vara att ha ett fält med pekare till varje post. Då kan man, till synes, radera en post genom att ta bort pekaren och flytta ihop resten av pekarna i fältet. Det går ju mycket fortare, om alla pekarna får plats i minnet. Jämför med Miniregistret i Utmaningen 84-4, där pekare användes för att snabba upp hanteringen av en matris med strängar.

Om pekare användes kan man också lämna "höl" i pekarfältet för borttagna poster. Om exempelvis en nolla betraktas som "ingen post", hoppar man bara över sådana pekare. Men det här fungerar inte bra annat än om man bara letar igenom filen sekventiellt, antingen framåt eller bakåt. Ofta räcker det ju inte med den metoden.

Tänk på att hur man än gör måste pekarna också lagras på disketten.

TERMINALPROGRAM

Det här är också en kår gammal bekant. Efterfrågades redan i 84-2. Tony Rogvall har gjort ett program som fungerar, men det ska få lite fler möjligheter innan det anses som färdigt. Tonys program använder bit-map mode för att få 64 tecken per rad på skärmen. En variant med valbar hastighet osv säges vara under utveckling när det här skrives.

På tal om det här med terminalprogram kan jag nämna att jag och Sven Lundgren, bara för att testa om det gick, har provat att använda en 99:a som terminal till en annan. Svens maskin körde med Terminal Emulator, och genom att koppla ihop den med min kunde vi köra Pascal på Svens dator utan att flytta p-kodskortet från min maskin till hans.

Vi använde alltså inte något modem, utan kopplade direkt med kabel, men inget annat än telefonräkningen hindrar att modem används.

PROGRAMMLISTNINGAR

Lennart Lindberg ville ha ett program som tillät listning av BASIC program på papper med exakt samma utseende som man får på skärmen. Tanken var att detta skulle underlätta inmatningen av program från tidningen. Själv anser jag att det är bättre med den metoden som använts hittills, dvs med radnumren för sig, men jag vet att det finns andra åsikter. Vad tycker du? Skriv och berätta"

Nåväl, det finns en och annan lösning på det här, men de förutsätter i regel att programmet listas på diskett först, i DIS/VAR 80 format. Elegantare hade det ju varit om listningen hade gjorts direkt från minnet, med hjälp av något passande assemblerprogram. Nu har jag fått reda på att något i den stilen är under utveckling, men jag har inget färdigt att visa än. Kanske det kan komma senare.

TIDSFÖRDRIV

Slutligen kan jag inte undanhålla er det här programmet. Det är avsett för E/A modulen, men det går att köra även med exempelvis Mini Memory. Jag kom på det här när jag höll på med att forska i den här kassetthistorien.

Ladda in programmet, och starta sedan med programnamnet MAG. En massa ettor (vanligtvis) dyker upp på skärmen. Sätt sedan igång kassettspelaren, gärna med lite musik på bandet, och titta.

```

*****
*
* Den här delen flyttas till RAM PAD och exekveras där
*
DEF MAG
REF VDPWA,VDPWD,VWTR

MAGIN EQU >1B           Magnetic tape input
GPLWS EQU >83E0
USERWS EQU >20BA        Workspace med E/A

MAG
LI R0,>8300             Flytta en del av programmet
LI R1,RESTART           till RAM PAD. ökar hastigheten
MOVLOP CI R1,LEAVE
JHE MOVED
MOV *R1+,*R0+
JMP MOVLOP

MOVED LWPI GPLWS        ökar farten
CLR R12                 För att få rätt CRU adress

LI R0,>01D0             Text mode
BLWP 'VWTR
SWPB R0
MOVB R0,>83D4

LI R0,>0717             Svart/cyan
BLWP 'VWTR

B >8300                 Hoppa till kod i snabbt minne

*****
*
* Den här delen flyttas till RAM PAD och exekveras där
*

RESTART LI R0,>0040      VDP adress till >0000 (för
MOVB R0,'VDPWA          skrivning)
SWPB R0
MOVB R0,'VDPWA

LI R2,960               Antal tecken på skärmen

INLOP
LI R1,>3020
TB MAGIN                Läs av magnetic in
JNE ISZERO
AI R1,>0100              Visa '1' om biten var hög
ISZERO MOVB R1,'VDPWD
DEC R2                  Räkna tecken
JNE INLOP

JMP RESTART
LEAVE

```

Programmet stannar aldrig självmant, utan du får stänga av datorn för att bli kvitt det. För att få upp farten utslöt jag en avkänning av tangentbordet. Som det är nu hinner det med en "sampling rate" på drygt 33 kHz, dvs det kollar magnetic input mer än 33000 gånger per sekund, eller ungefär två miljoner gånger i minuten om man så vill. Vadå 99:an långsam?

Riktigt intressant vore det att läsa av exempelvis radion på magnetic in och sedan låta datorn driva en utbit till samma värden som den läser in. Jag undrar hur pass väl det som kommer ut, förstärkt och skickat genom en högtalare, liknar det som kommer in?

Slut i rutan"

MINIASSEMBLER av Lennart Thelander

```

100 CALL INITIALIZE(ADR):: PRINT ADR
110 CALL DEF("CHIME",ADR)
112 CALL VALUE TO HEX(ADR,H01$):: H01$="RHO1$ :: PR
INT H01$
115 CALL BYTE(1,ADR):: CALL EVEN(ADR)
120 CALL LI("R0",4096,ADR):: CALL LI("R1",ADR+50,ADR)
:: CALL LI("R2",118,ADR)
130 CALL BLWP("2024",ADR):: LOOP=ADR :: CALL LIM(0,
ADR):: CALL LI("R10",4096,ADR)
140 CALL MOV("R10","83CC",ADR):: CALL SOCB(H01$,"83
FD",ADR):: CALL MOVB(H01$,"83CE",ADR)
150 CALL LIM(2,ADR):: LOOP2=ADR :: CALL MOVB("83CE"
,"83CE",ADR)
160 CALL JEQ((LOOP-ADR-2)/2,ADR):: CALL JMP((LOOP2-AD
R-2)/2,ADR)
170 CALL BYTE(5,ADR)
180 CALL BYTE(159,ADR)
190 CALL BYTE(191,ADR)
200 CALL BYTE(223,ADR)
210 CALL BYTE(255,ADR)
220 CALL BYTE(227,ADR)
230 CALL BYTE(001,ADR)
240 CALL BYTE(009,ADR)
250 CALL BYTE(142,ADR)
260 CALL BYTE(001,ADR)
270 CALL BYTE(164,ADR)
280 CALL BYTE(002,ADR)
290 CALL BYTE(197,ADR)
300 CALL BYTE(001,ADR)
310 CALL BYTE(144,ADR)
320 CALL BYTE(182,ADR)
330 CALL BYTE(211,ADR)
340 CALL BYTE(006,ADR)
350 CALL BYTE(003,ADR)
360 CALL LOAD(ADR,145,183,212,5):: ADR=ADR+4
370 CALL LOAD(ADR,003,146,184,213,4):: ADR=ADR+5
380 CALL LOAD(ADR,005,167,004,147,176,214,5):: ADR=AD
R+7
390 CALL LOAD(ADR,003,148,177,215,6):: ADR=ADR+5
400 CALL LOAD(ADR,003,149,178,216,7):: ADR=ADR+5
410 CALL LOAD(ADR,005,202,002,150,179,208,6):: ADR=AD
R+7
420 CALL LOAD(ADR,003,151,180,209,5):: ADR=ADR+5
430 CALL LOAD(ADR,003,152,181,210,4):: ADR=ADR+5
440 CALL LOAD(ADR,005,133,003,144,182,211,5):: ADR=AD
R+7
450 CALL LOAD(ADR,003,145,183,212,6):: ADR=ADR+5
460 CALL LOAD(ADR,003,146,184,213,7):: ADR=ADR+5
470 CALL LOAD(ADR,005,164,002,147,176,214,6):: ADR=AD
R+7
480 CALL LOAD(ADR,003,148,177,215,5):: ADR=ADR+5
490 CALL LOAD(ADR,003,149,178,216,4):: ADR=ADR+5
500 CALL LOAD(ADR,005,197,001,150,179,208,5):: ADR=AD
R+7
510 CALL LOAD(ADR,003,151,180,209,6):: ADR=ADR+5
520 CALL LOAD(ADR,003,152,181,210,7):: ADR=ADR+5
530 CALL LOAD(ADR,003,159,191,223,0):: ADR=ADR+5
550 CALL END(ADR)
30000 SUB INITIALIZE(ADR):: CALL INIT :: CALL PEEK(8194
,ADR,B):: ADR=ADR+256+B :: SUBEND
30010 SUB STARTADR(ADR):: CALL PEEK(8194,ADR,B):: ADR=A
DR+256+B :: SUBEND
30020 SUB END(ADR):: CALL DLOAD(ADR,8194):: SUBEND
30030 SUB BYTE(V,ADR):: CALL LOAD(ADR,V):: ADR=ADR+1 ::
SUBEND
30040 SUB WORD(V,ADR):: CALL DLOAD(V,ADR):: SUBEND
30050 SUB EVEN(ADR):: IF INT(ADR/2)*2<>ADR THEN ADR=ADR
+1
30051 SUBEND
30060 SUB DEF(NAMES,ADR):: CALL PEEK(8194,A,B):: A=A+25
6+B-8 :: CALL DLOAD(A,8196):: NAMES=NAMES$8"
30070 FOR I=1 TO 6 :: CALL LOAD(A+I-1,ASC(SEGS(NAMES,I,
1))):: NEXT I :: CALL DLOAD(ADR,A+6):: SUBEND
30080 SUB VALUE TO HEX(IN1,OUT$):: IN=IN1
30090 IN=IN/16 :: A=(IN-INT(IN))*16 :: IN=INT(IN)/16 ::
B=(IN-INT(IN))*16 :: IN=INT(IN)/16 :: C=(IN-INT(
IN))*16 :: IN=INT(IN)/16 :: D=(IN-INT(IN))*16
30100 IF D>9 THEN D$=CHR$(D+55)ELSE D$=CHR$(D+48)
30110 IF C>9 THEN C$=CHR$(C+55)ELSE C$=CHR$(C+48)
30120 IF B>9 THEN B$=CHR$(B+55)ELSE B$=CHR$(B+48)
30130 IF A>9 THEN A$=CHR$(A+55)ELSE A$=CHR$(A+48)
30140 OUT$=D$C$B$A$ :: SUBEND
30150 SUB HEX TO VALUE(INS,OUT):: DIM A(3):: INS=RPTS("
0",MAX(0,4-LEN(INS)))&INS
30160 FOR I=0 TO 3 :: A(I)=ASC(SEGS(INS,I+1,1)):: IF A(
I)>60 THEN A(I)=A(I)-55 ELSE A(I)=A(I)-48
30170 NEXT I :: OUT=A(0)*4096+A(1)*256+A(2)*16+A(3):: S
UBEND
30180 SUB REG(INS,OUT):: IF SEGS(INS,1,1)="R" THEN INS=
SEGS(INS,2,255)
30190 OUT=VAL(INS):: SUBEND
30200 SUB GENADR(INS,OUT1,OUT2):: OUT2=-1 :: IF POS(INS
,"+",1)=LEN(INS)THEN CALL REG(SEGS(INS,2,LEN(INS)
-2),OUT1):: OUT1=OUT1+48 :: SUBEXIT
30210 AS=SEGS(INS,1,1):: IF AS="+" THEN CALL REG(SEGS(I
NS,2,255),OUT1):: OUT1=OUT1+16 :: SUBEXIT
30220 IF AS="!" THEN 30230 ELSE CALL REG(INS,OUT1):: SU
BEXIT

```

Litet Amerikanskt Skvaller

av Lennart Lindberg

Det handlar om 99-ans framtid. Jag har skaffat initierade upplysningar om hur "snacket går" bland intresserade i USA. För det första är det alldeles klart att TI själva aldrig tar upp 99-an igen. De har nämligen gjort en överenskommelse med de amerikanska skattemyndigheterna, som innebär att de fick dra av sina förluster på 99-an om de lovade att aldrig ta upp den igen. De har alltså inte råd.

För det andra är det lika klart att det finns kompetenta företag som har ett tydligt intresse av att arbeta vidare med 99-an - f d TI-anställda som öppnat eget, t ex Triton, men också flera av de välkända, t ex Myarc och Navarone. Vi har tidigare skrivit om CorComp som har så fina tillbehör - Expansionsbox, Disc Controller mm - men som varit i finansiella svårigheter. De är nu på väg ut ur dessa och tjänar pengar igen efter ett ingripande av de myndigheter som övervakar företag i USA. Man såg till att de gjorde sig av med en del försäljare och skaffade fler ingenjörer, och det var tydligen vad som behövdes.

En firma uppges sälja Extended Basic-moduler för 59,95 dollar - mindre än hälften av TI:s gamla pris - och det är nygjorda grejor. Samma prisrelationer sägs gälla Mini Memory, Editor/Assembler och TI Writer bl a. Man misstänker att dessa grejor är gjorda på TI:s egna specifikationer, som letat sig ut på marknaden.

Bästa ryktet just nu sägs Myarc ha. De gjorde förut en hårddisk - Winchester drive - för 99-an, som ger maskinen fantastiska egenskaper. Nu har de kommit med även andra saker såsom alla slags kort för expansionsboxen - 32K-minne, RS232, Disk-kontrollkort. De planeras snart komma med p-code kort för Pascal och med en hel expansionsbox med RS232, 32K-minne och disk-kontrollkort för fyra st tätpackade dubbelsidiga diskstationer.

Men det kommer mera. Det ryktas mycket bestämt att Myarc kommer med en TI 99/128 - en storebror till 4A. Den finns tydligen i prototyp och uppges bli offentligt presenterad på CES - den stora datormässan i USA - i juni. Den sägs ha inbyggd X-Basic, C-Basic, Pascal, RS232 och disk-kontrollkort och kanske ett modem. 48K RAM i basversionen. Och priset? 400 dollar, säger ryktet. Mors! säger väl vi som gav nästan det för våra nakna 4A före prissänkningarna. Leverans till julen -85, sägs det.

Jag har också hört talas om en ny tidning med gott rykte, kallad Mini-Mag. Vi skall försöka skaffa den för att kunna fortsätta vidarebefordra mera intressanta informationer till dig, kära läsare. För den som vill skaffa den själv - adressen är S.O.S Publishers, 21777 Ventura Boulevard, Woodland Hills, California 91364.

16024 00000100,0010,0000

```

30880 SUB MOV(I$,J$,A):: CALL FORMAT1(49152,I$,J$,A)::
SUBEND
30890 SUB MOV(B(I$,J$,A):: CALL FORMAT1(53248,I$,J$,A)::
SUBEND
30900 SUB STST(I$,A):: CALL FORMAT8(704,I$,A):: SUBEND
30910 SUB STWP(I$,A):: CALL FORMAT8(672,I$,A):: SUBEND
30920 SUB SWPB(I$,A):: CALL FORMAT6(1728,I$,A):: SUBEND
30930 SUB ANDI(I$,V,A):: CALL FORMAT8(576,I$,V,A):: SUB
END
30940 SUB ORI(I$,V,A):: CALL FORMAT8(608,I$,V,A):: SUBE
ND
30950 SUB XOR(I$,J$,A):: CALL FORMAT3(10240,I$,J$,A)::
SUBEND
30960 SUB INV(I$,A):: CALL FORMAT6(1344,I$,A):: SUBEND
30970 SUB CLR(I$,A):: CALL FORMAT6(1216,I$,A):: SUBEND
30980 SUB SETO(I$,A):: CALL FORMAT6(1792,I$,A):: SUBEND
30990 SUB SOC(I$,J$,A):: CALL FORMAT1(57344,I$,J$,A)::
SUBEND
31000 SUB SOCB(I$,J$,A):: CALL FORMAT1(61440,I$,J$,A)::
SUBEND
31010 SUB SZC(I$,J$,A):: CALL FORMAT1(16384,I$,J$,A)::
SUBEND
31020 SUB SZCB(I$,J$,A):: CALL FORMAT1(20480,I$,J$,A)::
SUBEND
31030 SUB SRA(I$,C,A):: CALL FORMAT5(2048,I$,C,A):: SUB
END
31040 SUB SRL(I$,C,A):: CALL FORMAT5(2304,I$,C,A):: SUB
END
31050 SUB SLA(I$,C,A):: CALL FORMAT5(2560,I$,C,A):: SUB
END
31060 SUB SRC(I$,C,A):: CALL FORMAT5(2816,I$,C,A):: SUB
END
31070 SUB NOP(A):: CALL JMP(0,A):: SUBEND
31080 SUB RT(A):: CALL B("11",A):: SUBEND

```

```

30230 OUT2=POS(INS,"("): IF OUT2=0 THEN OUT1=32 :: G
OTO 30250
30240 CALL REG(SEGS(INS,OUT2+1,LEN(INS)-1-OUT2),OUT1)::
OUT1=OUT1+32 :: INS=SEGS(INS,1,OUT2-1)
30250 CALL HEX TO VALUE(SEGS(INS,2,255),OUT2):: SUBEND
30260 SUB DLOAD(V,A):: CALL SPLIT(V,B,C):: CALL LOAD(A,
B,C):: A=A+2 :: SUBEND
30270 SUB SPLIT(V,H,L):: H=INT(V/256):: L=V-H*256 :: SU
BEND
30280 SUB FORMAT1(OP,IN1$,IN2$,A):: CALL GENADR(IN1$,CO
1,A1):: CALL GENADR(IN2$,CO2,A2):: OP=OP+CO2*64+C
O1 :: CALL DLOAD(OP,A)
30290 IF A1<>-1 THEN CALL DLOAD(A1,A)
30300 IF A2<>-1 THEN CALL DLOAD(A2,A)
30310 SUBEND
30320 SUB FORMAT2(OP,D,A):: CALL LOAD(A,OP,D):: A=A+2 ::
SUBEND
30330 SUB FORMAT3(OP,IN1$,IN2$,A):: CALL GENADR(IN1$,CO
1,A1):: CALL REG(IN2$,CO2):: OP=OP+CO2*64+CO1 ::
CALL DLOAD(OP,A)
30340 IF A1<>-1 THEN CALL DLOAD(A1,A)
30350 SUBEND
30360 SUB FORMAT4(OP,INS,C,A):: CALL GENADR(INS,CO,A1)::
OP=OP+C*64+CO :: CALL DLOAD(OP,A):: IF A1<>-1 T
HEN CALL DLOAD(A1,A)
30370 SUBEND
30380 SUB FORMAT5(OP,INS,C,A):: CALL REG(INS,CO):: OP=O
P+C*16+CO :: CALL DLOAD(OP,A):: SUBEND
30390 SUB FORMAT6(OP,INS,A):: CALL GENADR(INS,CO,A1)::
OP=OP+CO :: CALL DLOAD(OP,A):: IF A1<>-1 THEN CAL
L DLOAD(A1,A)
30400 SUBEND
30410 SUB FORMAT8(OP,INS,V,A):: CALL REG(INS,B):: OP=OP
+B :: CALL DLOAD(OP,A):: CALL DLOAD(V,A):: SUBEND
30420 SUB FORMAT8B(OP,INS,A):: CALL REG(INS,B):: OP=OP+
B :: CALL DLOAD(OP,A):: SUBEND
30430 SUB A(I$,J$,A):: CALL FORMAT1(40960,I$,J$,A):: SU
BEND
30440 SUB AB(I$,J$,A):: CALL FORMAT1(45056,I$,J$,A):: S
UBEND
30450 SUB ABS(I$,A):: CALL FORMAT6(1856,I$,A):: SUBEND
30460 SUB AI(I$,V,A):: CALL FORMAT8(544,I$,V,A):: SUBEN
D
30470 SUB DEC(I$,A):: CALL FORMAT6(1536,I$,A):: SUBEND
30480 SUB DECT(I$,A):: CALL FORMAT6(1600,I$,A):: SUBEND
30490 SUB DIV(I$,J$,A):: CALL FORMAT3(15360,I$,J$,A)::
SUBEND
30500 SUB INC(I$,A):: CALL FORMAT6(1408,I$,A):: SUBEND
30510 SUB INCT(I$,A):: CALL FORMAT6(1472,I$,A):: SUBEND
30520 SUB MPY(I$,J$,A):: CALL FORMAT3(14336,I$,J$,A)::
SUBEND
30530 SUB NEG(I$,A):: CALL FORMAT6(1280,I$,A):: SUBEND
30540 SUB S(I$,J$,A):: CALL FORMAT1(24576,I$,J$,A):: SU
BEND
30550 SUB SB(I$,J$,A):: CALL FORMAT1(28672,I$,J$,A):: S
UBEND
30560 SUB B(I$,A):: CALL FORMAT6(1088,I$,A):: SUBEND
30570 SUB BL(I$,A):: CALL FORMAT6(1664,I$,A):: SUBEND
30580 SUB BLWP(I$,A):: CALL FORMAT6(1024,I$,A):: SUBEND
30590 SUB JEQ(V,A):: CALL FORMAT2(19,V,A):: SUBEND
30600 SUB JGT(V,A):: CALL FORMAT2(21,V,A):: SUBEND
30610 SUB JHE(V,A):: CALL FORMAT2(20,V,A):: SUBEND
30620 SUB JH(V,A):: CALL FORMAT2(27,V,A):: SUBEND
30630 SUB JL(V,A):: CALL FORMAT2(26,V,A):: SUBEND
30640 SUB JLE(V,A):: CALL FORMAT2(18,V,A):: SUBEND
30650 SUB JLT(V,A):: CALL FORMAT2(17,V,A):: SUBEND
30660 SUB JMP(V,A):: CALL FORMAT2(16,V,A):: SUBEND
30670 SUB JNC(V,A):: CALL FORMAT2(23,V,A):: SUBEND
30680 SUB JNE(V,A):: CALL FORMAT2(22,V,A):: SUBEND
30690 SUB JNO(V,A):: CALL FORMAT2(25,V,A):: SUBEND
30700 SUB JOP(V,A):: CALL FORMAT2(28,V,A):: SUBEND
30710 SUB JOC(V,A):: CALL FORMAT2(24,V,A):: SUBEND
30720 SUB RTWP(A):: CALL DLOAD(896,A):: SUBEND
30730 SUB X(I$,A):: CALL FORMAT6(1152,I$,A):: SUBEND
30740 SUB XOP(I$,J$,A):: CALL FORMAT3(11264,I$,J$,A)::
SUBEND
30750 SUB C(I$,J$,A):: CALL FORMAT1(32768,I$,J$,A):: SU
BEND
30760 SUB CB(I$,J$,A):: CALL FORMAT1(36864,I$,J$,A):: S
UBEND
30770 SUB CI(I$,V,A):: CALL FORMAT8(640,I$,V,A):: SUBEN
D
30780 SUB COC(I$,J$,A):: CALL FORMAT3(8192,I$,J$,A):: S
UBEND
30790 SUB CZC(I$,J$,A):: CALL FORMAT3(9216,I$,J$,A):: S
UBEND
30800 SUB LDCR(I$,V,A):: CALL FORMAT4(12288,I$,V,A):: S
UBEND
30810 SUB SBO(V,A):: CALL FORMAT2(29,V,A):: SUBEND
30820 SUB SBZ(V,A):: CALL FORMAT2(30,V,A):: SUBEND
30830 SUB STCR(I$,V,A):: CALL FORMAT4(13312,I$,V,A):: S
UBEND
30840 SUB TB(V,A):: CALL FORMAT2(31,V,A):: SUBEND
30850 SUB LI(I$,V,A):: CALL FORMAT8(512,I$,V,A):: SUBEN
D
30860 SUB LIM(V,A):: CALL FORMAT8(768,"0",V,A):: SUBEN
D
30870 SUB LWPI(V,A):: CALL FORMAT8(736,"0",V,A):: SUBEN
D

```

Personal Record Keeping-filer från Basic

av Jan Alexandersson

I Programbiten 84-03 fanns på sid 14 en artikel om Record Keeping från BASIC. Programmet fungerar ej helt bra om data saknas. Man måste testa om Z=1 vilket betyder att data saknas. Man kan sedan anropa ett PRINT-kommando med en tom sträng.

Detta inspirerade mig att leta efter flera CALL-kommandon och ta reda på syntaxen litet noggrannare. Det finns sammanlagt 7 kommandon:

- CALL P "prepare"
- CALL L "load"
- CALL S "save"
- CALL D "display"
- CALL A "accept"
- CALL G "get"
- CALL H "head"

CALL P

Detta reserverar minnesutrymme för RK-filen. Resterande minnesutrymme kan användas för BASIC. Om t.ex. 10 kbytes reserveras för RK-fil kan 4 kbytes användas för BASIC. Detta måste skrivas i kommandomod utan radnummer och följas av NEW. Om flexskiva används bör denna först begränsas till en fil med CALL FILES(1) följt av NEW. Därefter skrivs CALL P(9000) följt av NEW. Expansionsminne 32k kan ej användas. RAM-minnet får följande uppdelning:

VDP-RAM med kassett		VDP-RAM med flexskiva	
SKÄRM	2k	SKÄRM	2k
BASIC	4k	BASIC	4k
RK-FIL	10k	RK-FIL	9k
		DISK	1k
-----		-----	
CALL P(10000)		CALL FILES(1)	
NEW		NEW	
		CALL P(9000)	
		NEW	

CALL L

Laddar in RK-fil i reserverat minne från kassett eller flexskiva.

CALL S

Sparar RK-fil från RAM-minne till CS eller DSK. RK-filen sparas i PROGRAM-format dvs det går mycket snabbare att överföra den till kassett eller flexskiva. För kassett gör detta att 10 kbytes kan lagras på 2,5 min jämfört med 10 min med normala datafiler öppnade med OPEN och CLOSE.

CALL D

Detta kan även användas utan RK-fil och fungerar ungefär som DISPLAY AT SIZE i Extended Basic.

CALL A

Detta kan även användas utan RK-fil och fungerar ungefär som ACCEPT AT SIZE i Extended Basic. Det går ej att bryta programmet vid CALL A vilket gör att det bör sparas innan det provas. Även VALIDATE finns delvis med eftersom CALL A upprepas om man väljer fel på sträng och numerisk.

CALL G

Skriver, läser och raderar data.

CALL H

Skriver och läser uppgifter om RK-filens struktur. Antalet fält och poster behöver ej skrivas eftersom de stegas fram automatiskt med övriga kommandon.

SYNTAX FÖR PERSONAL RECORD KEEPING ELLER STATISTICS

CALL P(minnesutrymme) PREPARE

CALL L(filnamn,test) LOAD
test=0 vid misslyckat försök
test=1 vid lyckat försök

CALL S(filnamn,test) SAVE
test=0 vid misslyckat försök
test=1 vid lyckat försök

CALL D(rad,kolumn,längd,data) DISPLAY
rad=1-24
kolumn=1-28
längd=1-28 (kan ej nå nästa rad)
data= sträng eller numerisk

CALL A(rad,kolumn,längd,test,data) ACCEPT
rad=1-24
kolumn=1-28
längd=1-28 (kan ej nå nästa rad)
test=1 om data finns
test=2 om data ej finns
data= sträng eller numerisk

CALL G(0,post,fält,data) skriv GET
CALL G(1,post,fält,test,data) läs
CALL G(2,post,fält,data) radera
test=0 om data finns
test=1 om data ej finns
data= sträng eller numerisk

CALL H(operation,information,fält,data) HEAD
operation=0 vid skriv
1 vid läs
information= 1 data=filnamn
2 dag (0-99)
3 månad (0-99)
4 år (0-99)
5 fält (items)
6 poster (pages)
7 huvudlängd (ej skriv)
8 postlängd (ej skriv)
9 fältrubrik (item name)
10 fälttyp (item_type)
1 för CHAR
2 för INT
3 för DEC
4 för SCI
11 editeringsbredd (width)
12 decimaler
13 lagringsbredd
14 relativ fältposition

fält=0 för övergripande information 1-8
fält=1-poster för information 9-14
data= sträng eller numerisk

SNABBFIL

För att testa de nya BASIC-kommandona har jag gjort ett program som i stort sett gör samma saker som det inbyggda programmet dock med flera förbättringar. Följande funktioner har tagits med:

- N - initiera ny fil
- L - ladda äldre fil
- A - addera data
- D - visa data (display)
- C - ändra data (change)
- F - ändra fältnamn (saknas i TEXAS)
- S - spara fil
- ändra programnamn (saknas i TEXAS)
- svenska bokstäver (saknas i TEXAS)

Följande funktioner saknas:

- radera hela poster
- sortering
- matematiska transformationer

Jag har försökt ha så mycket data samtidigt som möjligt på skärmen och inte radera skärmen i onödan som TEXAS gör. De två nedersta raderna används som meny med all information om vad som skall väljas. Programmet kan köras med någon av modulerna PERSONAL RECORD KEEPING eller STATISTICS.

Programbeskrivning

- 100- 150 initiering med A,X och Ö
- 160- 210 START: val av ny eller äldre fil
- 220- 500 LADDA: läser in äldre fil och skriver ut namn, datum och struktur (fältets bredd anges hexadecimalt för att spara utrymme på skärmen)
- 510- 560 MENY
- 570- 650 HUVUD: uppdaterar antalet poster och antalet bytes som används av RK-filen.
- 660- 730 FÄLT: ändrar fältrubrik (item name)
- 740-1030 NY: initiera ny RK-fil med fältnamn, fälttyp, fältbredd och antalet decimaler
- 1040-1280 VISA DATA: skriver önskad post på skärm
- 1290-1440 NY DATA: ändrar data i önskad post
- 1440-1480 ADD DATA: lägger till en ny post
- 1490-1690 SPARA: sparar RK-fil till CS eller DSK. Om samma programnamn och filnamn som tidigare fanns önskas tryck endast på ENTER för dessa
- 1700 slut
- 1710-1770 SUDDA DATA: raderar meny och data
- 1780-1800 SUDDA MENY
- 1810-1870 MAX: räknar ut maximal bredd för data
- 1880-1900 skriver ut rubrik som används på två olika ställen i programmet

Skärmdisponering

1234567890123456789012345678

- 1 ADRESSER 1985-01-19
- 2
- 3 FÄLT=8 POST=35 BYTE=3738
- 4
- 5 2
- 6
- 7 EFTERNAMN PETTERSSON
- 8 FÖRNAMN BERTIL
- 9 GATA/BOX DATAVÄGEN 5,3TR
- 0 POSTNR 12345
- 1 POSTADR STOCKHOLM
- 2 TELEFON 08-7101010
- 3 FÖDD:ÅR 1950
- 4 FÖDD:TID 12-31
- 5 osv
- 6
- 7
- 8
- 9
- 0
- 1 (plats för sista fält 15)
- 2
- 3 F(ÄLT) A(DD) D(ISP) S(PAR)
- 4 E(XIT) L(ADDA)

Använda variabler

- K tangentens ASCII-kod vid CALL KEY
- S status vid CALL KEY
- F aktuellt fält
- FLT maximalt antal fält
- P aktuellt post
- PST maximalt antal poster
- I aktuell information
- D temporär lagring av numerisk data
- D\$ temporär lagring av sträng data
- T test av hur kommandot fungerar
- HL huvudlängd
- PL postlängd
- TYP fälttyp (1-4)
- BRED bredden av aktuellt fält
- MAX största fältbredden
- NAMN\$ programnamn som skrivs i RK-filen
- FIL\$ filnamn som används vid LADDA och SPARA

Möjliga programvarianter

Genom att köra RK-filen från BASIC får man full kontroll över skärmens utseende. Det är möjligt att skriva ut data på andra lednen.

NAMN	POSTADR	TELEFON
ADAM	STOCKHOLM	08-7101010
BERTIL	GÖTEBORG	031-101010
CESAR	MALMÖ	040-101010
ovs		

På detta sätt kan c:a 20 poster visas samtidigt på skärmen. RK-filen kan innehålla fler fält än dessa tre men programmet behöver ej läsa av dessa. Man kan låta flera olika BASIC-program efter varandra använda eller bearbeta RK-filen. NEW, OLD eller SAVE kommer ej att sudda RK-filen utan den finns kvar ungefär som en flexskiva. Det krävs BYE eller CALL L av en annan fil för att RK-filen skall försvinna.

- 100 REM SNABBFIL RK/ST
- 110 REM JAN ALEXANDERSSON
- 120 REM 1985-01-16
- 130 CALL CHAR(91,"00280038447C4444")
- 140 CALL CHAR(92,"0028007C4444447C")
- 150 CALL CHAR(93,"00382838447C4444")
- 160 REM START
- 170 CALL CLEAR
- 180 CALL D(23,1,10,"ÖNSKAD FIL")
- 190 CALL D(24,1,12,"N(Y) L(ADDA)")
- 200 CALL KEY(3,K,S)
- 210 ON 1-(K=78)-2*(K=76)GOTO 200,740,220
- 220 REM LADDA
- 230 CALL HCHAR(24,3,32,28)
- 240 CALL A(23,12,15,T,FIL\$)
- 250 IF T=2 THEN 280
- 260 CALL L(FIL\$,T)
- 270 IF T=0 THEN 240
- 280 CALL CLEAR
- 290 CALL H(1,1,0,NAMN\$)
- 300 CALL D(1,1,9,NAMN\$)
- 310 D\$="19"
- 320 FOR I=4 TO 2 STEP -1
- 330 CALL H(1,I,0,D)
- 340 D\$=D\$&SEG\$("-0",1-(I=4),2+(D>9)+(I=4))&STR\$(D)
- 350 NEXT I
- 360 CALL D(1,17,10,D\$)
- 370 GOSUB 1880
- 380 GOSUB 570
- 390 GOSUB 1810

```

400 FOR F=1 TO FLT
410 CALL H(1,9,F,D$)
420 CALL D(6+F,1,9,D$)
430 CALL H(1,10,F,D)
440 CALL D(6+F,12+MAX,4,SEG$( "CHARINT DEC SCI ",1+(D-1)*4,4))
450 CALL H(1,11,F,D)
460 CALL HCHAR(6+F,31+(MAX<15),48+D-7*(D>9))
470 CALL H(1,12,F,D)
480 CALL HCHAR(6+F,32,48+D+16*(D=0))
490 NEXT F
500 GOTO 510
510 REM MENY
520 CALL D(23,1,28,"F(ÄLT) A(DD) D(ISP) S(PAR)")
530 CALL D(24,1,28,"E(XIT) L(ADDA)")
540 CALL KEY(3,K,S)
550 IF S<1 THEN 540
560 ON 1-(K=70)-2*(K=65)-3*(K=68)-4*(K=83)-5*(K=69)-6*(K=76)GOTO 540,660,1440,1040,1490,1700,160
570 REM HUVUD
580 CALL H(1,5,0,FLT)
590 CALL D(3,6,2,STR$(FLT))
600 CALL H(1,6,0,PST)
610 CALL D(3,14,3,STR$(PST))
620 CALL H(1,7,0,HL)
630 CALL H(1,8,0,PL)
640 CALL D(3,23,5,STR$(512+HL+PST*PL))
650 RETURN
660 REM FÄLT
670 GOSUB 1780
680 FOR F=1 TO FLT
690 CALL A(6+F,1,9,T,D$)
700 IF T=2 THEN 720
710 CALL H(0,9,F,D$)
720 NEXT F
730 GOTO 510
740 REM NY
750 GOSUB 1780
760 CALL H(0,5,0,0)
770 CALL H(0,6,0,0)
780 CALL D(5,1,28,"FÄLTNAMN TYP BREDD DEC")
790 CALL D(23,1,28,"1=CHAR 2=INT 3=DEC 4=SCI")
800 FOR F=1 TO 15
810 CALL A(6+F,1,9,T,D$)
820 IF T=2 THEN 970
830 CALL H(0,9,F,D$)
840 CALL A(6+F,13,1,T,TYP)
850 IF (T=2)+(TYP>4)THEN 840
860 CALL H(0,10,F,TYP)
870 IF TYP=4 THEN 920
880 CALL A(6+F,18,2,T,D)
890 IF (D>15)+(T=2)THEN 880
900 CALL H(0,11,F,D)
910 IF TYP<3 THEN 960
920 CALL A(6+F,24,1,T,D)
930 IF T=2 THEN 920
940 CALL H(0,12,F,D)
950 IF F=15 THEN 970
960 NEXT F
970 FLT=F
980 GOSUB 1880
990 GOSUB 570
1000 MAX=15
1010 GOSUB 1710
1020 GOSUB 1810
1030 GOTO 510
1040 REM VISA DATA
1050 GOSUB 1710
1060 P=1
1070 CALL D(5,13,3,P)
1080 CALL A(5,14,3,T,P)
1090 IF P>PST THEN 1080
1100 FOR F=1 TO FLT
1110 CALL H(1,10,F,TYP)
1120 IF TYP=1 THEN 1170
1130 CALL G(1,P,F,T,D)
1140 IF T=1 THEN 1210
1150 CALL D(6+F,11,MAX,STR$(D))
1160 GOTO 1220
1170 CALL G(1,P,F,T,D$)
1180 IF T=1 THEN 1210
1190 CALL D(6+F,11,MAX,D$)
1200 GOTO 1220
1210 CALL HCHAR(6+F,13,32,MAX)
1220 NEXT F

```

```

1230 CALL D(23,1,28,"C(HANGE) M(ENY) N(XSTA)")
1240 CALL KEY(3,K,S)
1250 ON 1-(K=67)-2*(K=77)-3*(K=78)GOTO 1240,1290,510,1260
1260 P=1-P*(P<PST)
1270 CALL D(5,13,4,P)
1280 GOTO 1100
1290 REM NY DATA
1300 GOSUB 1780
1310 CALL D(5,13,4,P)
1320 FOR F=1 TO FLT
1330 CALL H(1,11,F,BRED)
1340 CALL H(1,10,F,TYP)
1350 IF TYP=1 THEN 1390
1360 CALL A(6+F,11,BRED,T,D)
1370 CALL G(-2*(T=2),P,F,D)
1380 GOTO 1410
1390 CALL A(6+F,11,BRED,T,D$)
1400 CALL G(-2*(T=2),P,F,D$)
1410 NEXT F
1420 GOSUB 570
1430 GOTO 510
1440 REM ADD DATA
1450 PST=PST+1
1460 P=PST
1470 GOSUB 1710
1480 GOTO 1290
1490 REM SPARA
1500 CALL CLEAR
1510 CALL D(22,1,14,"NAMN "&NAMN$)
1520 CALL A(22,6,9,T,NAMN$)
1530 IF T=2 THEN 1550
1540 CALL H(0,1,0,NAMN$)
1550 CALL D(23,1,8,"DATUM 19")
1560 CALL A(23,9,2,T,D)
1570 IF T=2 THEN 1560
1580 CALL H(0,4,0,D)
1590 CALL A(23,12,2,T,D)
1600 IF T=2 THEN 1590
1610 CALL H(0,3,0,D)
1620 CALL A(23,15,2,T,D)
1630 IF T=2 THEN 1620
1640 CALL H(0,2,0,D)
1650 CALL D(24,1,23,"FILNAMN "&FIL$)
1660 CALL A(24,9,15,T,FIL$)
1670 CALL S(FIL$,T)
1680 IF T=0 THEN 1660
1690 GOTO 160
1700 END
1710 REM SUDDA DATA
1720 GOSUB 1780
1730 CALL HCHAR(5,3,32,28)
1740 FOR F=1 TO FLT
1750 CALL HCHAR(6+F,13,32,MAX)
1760 NEXT F
1770 RETURN
1780 REM SUDDA MENY
1790 CALL HCHAR(23,1,32,64)
1800 RETURN
1810 REM MAX
1820 MAX=0
1830 FOR F=1 TO FLT
1840 CALL H(1,11,F,D)
1850 MAX=-MAX*(D<MAX)-D*(D>=MAX)
1860 NEXT F
1870 RETURN
1880 REM
1890 CALL D(3,1,22,"FÄLT= POST= BYTE=")
1900 RETURN

```

SÄLJES

TI 99/4A
Speech Synthesizer, Mini Memory, Extended Basic,
Terminal Emulator II, SST Compiler System.

Spel:
Tunnels of Doom, Adventure, Donkey Kong, Parcec,
Car Wars, Fathom, A-Maze-Ing, Microsurgeon, Alpiner
Tombstone City, Munch Man, Driving Demon.

Paketpris: 4500:-

Mikael Lundberg Telefon 090-12 94 47
Basstråket 7 C
902 57 Umeå

TI-59

Stefan Boltzmann...

Lysekil 28/12 1984

Hej Claes.

Detta program beräknar en gråkroppas utstrålningsemit-
tans i ett godtyckligt våglängdsintervall. Och vad är
nu detta då?

Om man har riktigt gott minne och läste N eller I på
gymnasiet kanske man kommer ihåg att det fanns en lag
som hette Stefan-Boltzmanns strålningslag och som - via
tabellttittande - lyder $M = e * s * T^4$,

där e är emissiviteten (alltid mellan noll och ett),

s är en konstant: $s = 5.6697 * 10^{-8}$

och T är den utstrålade kroppens temperatur i Kelvin.

Stefan-Boltzmanns lag är mycket enkel att hantera. Till
den behövs näppeligen några räknedosprogram. Den är
mycket användbar när man vill räkna ut emittansen M

över alla våglängder. Emittansen är ingenting krånglig-
ligare än utstrålad effekt per ytenhet och vill man
för skojs skull räkna ut solens effekt (!) görs detta

med: $P = M * A * e * s * T^4$,
där $e = 1$ (idealt svartkroppsstrålning),

$T = 5800 \text{ K}$ (medeltemperatur vid solytan)

och $A = 6 * 10^{18} \text{ m}^2$ (solens area).

Solens utstrålningseffekt fås då till cirka $4 * 10^{26}$
Watt.

Man kan naturligtvis också räkna på mer hemnära exempel
såsom solfångare, glödlampor, varma cigarettfimpar,
askan i spisen, spisplattors strålning pga. eftervärme,
m.m. Det enda man behöver veta är materialets tempera-
tur och dess emissivitet. Här följer en lista på några
materials emissivitet (för nytta och nöje):

Material	Emissivitet e
Polerat guld	0.02
Polerat stål	0.07
Wolframtråd	0.3
Oxiderat stål	0.79
Betong	0.92
Rött tegel	0.93
Is	0.96
Matt svart färg	0.97
Hud	0.98
Aska	0.99

Om man emellertid bara vill beräkna emittansen i ett
avgränsat våglängdsområde, såsom t.ex. det synliga
området eller det infraröda området, så duger inte
Stefan-Boltzmanns strålningslag. Istället måste man inte-
grera Planks Strålningslag över det önskade våglängds-
området. Detta är dock ganska så lurigt men har natur-
ligtvis en given lösning:

Vi tar fram våra 59:or och kör följande program!

Inmatningsinstruktioner

1. Stäng av och sätt på räknaren.
2. Knappa in steg 000-239.
3. Tryck 5 Op 17.
4. Ladda register 30-49 enligt registerlistning.
5. Knappa in steg 480-507.
6. Tryck 6 Op 17.
7. Spela in på magnetkort med 1 2nd Write, 3 2nd Write.
8. Provkör.

Körinstruktioner

1. Läs in sida 1 och 2.
2. Tryck START. (= "E").
3. Följ instruktionerna.

Programmet använder sig av samma typ av "prompting" som
i programmet Reglerteknik 1 i PB 84-4, dvs:

label: A = YES

B = NO

C = CONTINUE (behåller tidigare inmatade data)

D = ENTER (inmatning av data)

E' = START

Någon YES/NO-fråga förekommer inte i detta program.

Den integralalgoritm som används är Simpsons formel i
ML-09, varför standardmodulen måste sitta i.

Frågan "N?" frågar efter antalet integrationssteg. Det
är lämpligt att, såsom i körexemplet, göra två kör-
ningar med olika stegantal för att kolla att körningen
konvergerat. Om inte? Giv större N!

Våglängden ges i mikrometer och temperaturen i Kelvin.

Skall man t.ex. integrera över det synliga våglängds-
området (ljus) matar man in: lower lambda = 0.38 μm
och upper lambda = 0.76 μm .

I körexemplet beräknas emittansen från en rumstempe-
rerad kropp i våglängdsintervallet 8 till 12 mikro-
meter. Materialet är svartkroppsstrålning, dvs. emis-
siviteten är = 1.

Lycka till!

Martin Harnevie

Mölnådal

K-TRYCK AB

TI-59

000	76	LBL	060	00	00	120	71	SBR	180	03	03	480	98	ADV
001	16	A'	061	52	52	121	99	PRT	181	18	C'	481	04	4
002	42	STD	062	92	RTN	122	04	4	182	01	1	482	17	B'
003	10	10	063	76	LBL	123	17	B'	183	08	8	483	18	C'
004	53	(	064	18	C'	124	18	C'	184	09	9	484	69	DP
005	43	RCL	065	69	DP	125	01	1	185	42	STD	485	00	00
006	15	15	066	05	05	126	03	3	186	06	06	486	43	RCL
007	65	X	067	92	RTN	127	02	2	187	00	0	487	47	47
008	03	3	068	76	LBL	128	42	STD	188	92	RTN	488	69	DP
009	07	7	069	91	R/S	129	06	06	189	42	STD	489	01	01
010	04	4	070	22	INV	130	00	0	190	15	15	490	06	6
011	01	1	071	58	FIX	131	92	RTN	191	99	PRT	491	04	4
012	05	5	072	19	D'	132	42	STD	192	69	DP	492	52	EE
013	55	-	073	05	5	133	01	01	193	00	00	493	06	6
014	53	(	074	69	DP	134	99	PRT	194	03	3	494	19	D'
015	43	RCL	075	17	17	135	43	RCL	195	01	1	495	69	DP
016	10	10	076	92	RTN	136	49	49	196	07	7	496	02	02
017	65	X	077	76	LBL	137	69	DP	197	01	1	497	18	C'
018	33	X²	078	99	PRT	138	01	01	198	69	DP	498	32	X¹¹
019	33	X²	079	06	6	139	18	C'	199	02	02	499	99	PRT
020	54	)	080	06	6	140	01	1	200	18	C'	500	71	SBR
021	55	-	081	35	1/X	141	04	4	201	02	2	501	99	PRT
022	53	(	082	82	HIR	142	07	7	202	00	0	502	61	GTD
023	53	(	083	05	05	143	42	STD	203	08	8	503	01	01
024	01	1	084	82	HIR	144	06	06	204	42	STD	504	92	92
025	04	4	085	06	06	145	00	0	205	06	06	505	00	0
026	03	3	086	82	HIR	146	92	RTN	206	02	2	506	00	0
027	08	8	087	07	07	147	42	STD	207	92	RTN	507	00	0
028	08	8	088	82	HIR	148	02	02	208	99	PRT			
029	55	-	089	08	08	149	99	PRT	209	68	NOP			
030	43	RCL	090	18	C'	150	03	3	210	76	LBL			
031	10	10	091	92	RTN	151	17	B'	211	92	RTN			
032	55	-	092	76	LBL	152	07	7	212	53	(			
033	43	RCL	093	13	C	153	01	1	213	59	INT			
034	16	16	094	24	CE	154	52	EE	214	65	X			
035	54	)	095	03	3	155	08	8	215	02	2			
036	22	INV	096	44	SUM	156	19	D'	216	54	)			
037	23	LN X	097	06	06	157	69	DP	217	36	PGM			
038	75	-	098	76	LBL	158	04	04	218	09	09			
039	01	1	099	15	E	159	18	C'	219	13	C			
040	54	)	100	19	D'	160	01	1	220	36	PGM			
041	54	)	101	83	GD*	161	06	6	221	09	09			
042	92	RTN	102	06	06	162	07	7	222	14	D			
043	76	LBL	103	76	LBL	163	42	STD	223	32	X¹¹			
044	19	D'	104	10	E'	164	06	06	224	03	3			
045	22	INV	105	71	SBR	165	00	0	225	00	0			
046	57	ENG	106	91	R/S	166	92	RTN	226	42	STD			
047	92	RTN	107	03	3	167	42	STD	227	08	08			
048	76	LBL	108	00	0	168	16	16	228	61	GTD			
049	17	B'	109	42	STD	169	99	PRT	229	04	04			
050	42	STD	110	08	08	170	69	DP	230	80	80			
051	07	07	111	98	ADV	171	00	00	231	76	LBL			
052	73	RC*	112	71	SBR	172	02	2	232	81	RST			
053	08	08	113	99	PRT	173	17	B'	233	06	6			
054	69	DP	114	04	4	174	07	7	234	69	DP			
055	28	28	115	17	B'	175	01	1	235	17	17			
056	84	DP*	116	18	C'	176	52	EE	236	92	RTN			
057	07	07	117	04	4	177	08	8	237	68	NOP			
058	97	DSZ	118	17	B'	178	19	D'	238	68	NOP			
059	07	07	119	18	C'	179	69	DP	239	68	NOP			

1517002431. 30
 2437371331. 31
 3137001730. 32
 3513162413. 33
 3542132700. 34
 24313717. 35
 1731223723. 36
 4313421727. 37
 3056710000. 38
 1613005541. 39
 27133014. 40
 2732431735. 41
 1700552656. 42
 3513374135. 43
 3717303317. 44
 2442243745. 45
 1730243636. 46
 4363153070. 47
 0. 48
 4133331735. 49

 RADIANT EMITTANCE IN
 WAVELENGTH INTERVAL

LOWER CANADA (UN)?
 8.
 UPPER CANADA (UN)?
 12.
 TEMPERATURE (K)?
 300.
 EMISSIVITY?
 1.
 N?
 5.

RADIANT EMITTANCE IN
 W/CM² =
 .0120934323

 N?
 10.

RADIANT EMITTANCE IN
 W/CM² =
 .0120934323

 N?

TI-59

ROTEN UR- MED 120 SIFFROR!

HEJ !

Har tillbringat tiden mellan och under jul-nyårshelger med att förbättra mitt program för beräkning av kvadratrot med 120 siffrors precision.

Jag tycker det var alltför långsamt, efter en tids gnugande kom jag på KVADRERINGSREGLN !!

$$(a + b)^2 = a^2 + 2ab + b^2 \text{ och}$$

$$(a - b)^2 = a^2 - 2ab + b^2.$$

Alltså istället för att räkna ut kvadraten varje gång, läggs till eller dras ifrån den gamla kvadraten och fås en ny kvadrat.

Sagt och gjort sägs det! (Try and Error) Efter en vecka tycktes allt stämma. Satte in resultatet i mitt gamla program, resultat körtid 3.5 timmat (jfr 9).

Efter ytterligare omdisponeringar av register kunde jag få ut 125 siffror. Körtid cirka 4 timmar.

Program 1

Spelas in på normaluppdelning.

Startas med tryck: A RST STF IND 7 EE (Input) R/S

(**)Input = det tal Du vill ha roten ur.

(Tryck ej in parenteserna runt input!)

Bifogar även program 2 som beräknar kvadratroten med 155 siffrors precision.

På grund av utrymmesbrist blir det en del knappande för att starta programmet.

Startsekvens: tryck 10 OP 17 CMS ADV Input PRT ADV X 8

INV LOG) STO 95 . 1 SQR X² 1/X RST STF IND 7 EE R/S

(Input se ovan (**))

Exekveringstid för detta program cirka 7 timmar.

Mera TI-59 i Programbiten !

Sven-Arne Wallin

343 00 Älmhult

PS. Bifogar utskrift av program 1 roten ur 2, bifogar även utskrift av program 2 roten ur 3. (För att få ut kvadraten knappa in följande efter exekveringen:

RST LRN LBL A 61 STO 98 RC* 99 PRT 1 SUM 99 DSZ ADV

0 6 ADV ADV ADV R/S LRN A

CHECKSUM:

CHECKSUM:

BANK 1
6347184703.

BANK 1
6724721805.

PRDG
6347184703.

BANK 2
4725538200.

PRDG
1450260005.

2.	000	92	RTH	080	82	HIR
	001	91	R/S	081	17	17
14142.	002	98	ADV	082	94	+/-
13562.	003	99	PRT	083	82	HIR
37309.	004	98	ADV	084	06	06
50488.	005	65	X	085	22	INV
1688.	006	08	8	086	87	IFF
72420.	007	22	INV	087	02	02
96980.	008	28	LOG	088	00	00
78569.	009	54	)	089	97	97
67187.	010	82	HIR	090	02	2
53769.	011	05	05	091	08	8
48073.	012	08	8	092	82	HIR
17667.	013	69	DP	093	34	34
97379.	014	17	17	094	61	GTD
90732.	015	47	CMS	095	01	01
47846.	016	02	2	096	54	54
21070.	017	05	5	097	86	STF
38850.	018	42	STO	098	01	01
38753.	019	78	78	099	86	STF
43276.	020	05	5	100	02	02
41572.	021	22	INV	101	61	GTD
73501.	022	28	LOG	102	01	01
38462.	023	82	HIR	103	54	54
30912.	024	08	08	104	22	INV
29702.	025	04	4	105	86	STF
49248.	026	22	INV	106	01	01
	027	28	LOG	107	22	INV
	028	82	HIR	108	86	STF
199999999.	029	07	07	109	02	02
99999.	030	05	5	110	01	1
99999.	031	42	STO	111	00	0
99999.	032	75	75	112	82	HIR
99999.	033	05	5	113	67	67
99999.	034	65	X	114	97	DSZ
99999.	035	82	HIR	115	75	75
99999.	036	17	17	116	00	00
99999.	037	54	)	117	33	33
99999.	038	82	HIR	118	73	RC*
99999.	039	06	06	119	76	76
99999.	040	04	4	120	99	PRT
99999.	041	07	7	121	01	1
99999.	042	82	HIR	122	44	SUM
99999.	043	04	04	123	76	76
99999.	044	61	GTD	124	97	DSZ
99999.	045	01	01	125	78	78
99999.	046	54	54	126	00	00
99999.	047	22	INV	127	25	25
99999.	048	77	GE	128	02	2
99999.	049	00	00	129	49	PRD
99999.	050	62	62	130	76	76
99999.	051	03	3	131	97	DSZ
99999.	052	94	+/-	132	76	76
89801.	053	61	GTD	133	51	BST
86537.	054	00	00	134	02	2
69067.	055	63	63	135	05	5
84243.	056	87	IFF	136	42	STO
2249.	057	01	01	137	78	78
39409.	058	01	01	138	98	ADV
9940.	059	04	04	139	73	RC*
14784.	060	86	STF	140	78	78
84823.	061	01	01	141	99	PRT
63090.	062	02	2	142	01	1
23287.	063	65	X	143	44	SUM
81678.	064	82	HIR	144	78	78
78456.	065	17	17	145	97	DSZ
14558.	066	54	)	146	76	76
31997.	067	82	HIR	147	01	01
25805.	068	06	06	148	39	39
75885.	069	07	7	149	98	ADV
48800.	070	06	6	150	98	ADV
15641.	071	82	HIR	151	98	ADV
8603.	072	04	04	152	25	CLR
59134.	073	61	GTD	153	81	RST
88869.	074	01	01	154	43	RCL
46411.	075	54	54	155	76	76
52445.	076	22	INV	156	42	STO
65504.	077	77	GE	157	74	74
	078	00	00	158	85	+
	079	56	56	159	01	1

160	85	+	240	94	+/-
161	42	STD	241	44	SUM
162	77	77	242	74	74
163	85	+	243	44	SUM
164	02	2	244	79	79
165	03	3	245	97	DSZ
166	54	)	246	77	77
167	42	STD	247	02	02
168	79	79	248	15	15
169	82	HIR	249	82	HIR
170	16	16	250	15	15
171	33	X ²	251	32	X:T
172	85	+	252	82	HIR
173	82	HIR	253	14	14
174	16	16	254	42	STD
175	65	X	255	77	77
176	02	2	256	43	RCL
177	65	X	257	25	25
178	73	RC*	258	83	GD*
179	74	74	259	77	77
180	54	)	260	76	LBL
181	74	SM*	261	11	R
182	79	79	262	25	CLR
183	01	1	263	93	.
184	94	+/-	264	01	1
185	44	SUM	265	34	FX
186	74	74	266	33	X ²
187	44	SUM	267	35	1/X
188	79	79	268	91	R/S
189	97	DSZ	269	00	0
190	77	77	270	00	0
191	01	01			
192	73	73			
193	82	HIR			
194	16	16			
195	74	SM*			
196	76	76			
197	43	RCL			
198	76	76			
199	85	+			
200	24	CE			
201	85	+			
202	01	1			
203	85	+			
204	42	STD			
205	77	77			
206	02	2			
207	04	4			
208	85	+			
209	42	STD			
210	79	79			
211	01	1			
212	54	)			
213	42	STD			
214	74	74			
215	73	RC*			
216	74	74			
217	55	+			
218	82	HIR			
219	18	18			
220	75	-			
221	22	INV			
222	59	INT			
223	29	CP			
224	77	GE			
225	02	02			
226	29	29			
227	75	-			
228	01	1			
229	54	)			
230	74	SM*			
231	79	79			
232	65	X			
233	82	HIR			
234	18	18			
235	54	)			
236	22	INV			
237	74	SM*			
238	74	74			
239	01	1			

3.

17320.
50807.
56887.
72935.
27446.
34150.
58723.
66942.
80525.
38103.
80628.
5580.
69794.
51933.
1690.
88000.
37081.
14618.
67572.
48575.
67562.
61414.
15406.
70302.
99699.
45094.
99895.
24788.
11655.
51209.
43736.

299999999.	000	92	RTN	080	85	+
99999.	001	91	R/S	081	42	STD
99999.	002	03	3	082	97	97
99999.	003	01	1	083	01	1
99999.	004	42	STD	084	54	)
99999.	005	94	94	085	42	STD
99999.	006	05	5	086	96	96
99999.	007	22	INV	087	73	RC*
99999.	008	28	LOG	088	96	96
99999.	009	82	HIR	089	55	+
99999.	010	08	08	090	82	HIR
99999.	011	04	4	091	18	18
99999.	012	22	INV	092	54	)
99999.	013	28	LOG	093	75	-
99999.	014	82	HIR	094	22	INV
99999.	015	07	07	095	59	INT
99999.	016	05	5	096	29	CP
99999.	017	42	STD	097	77	GE
99999.	018	93	93	098	01	01
99999.	019	09	9	099	02	02
99999.	020	65	X	100	75	-
99999.	021	82	HIR	101	01	1
99999.	022	17	17	102	54	)
99999.	023	54	)	103	74	SM*
99999.	024	82	HIR	104	97	97
99999.	025	06	06	105	65	X
99999.	026	43	RCL	106	82	HIR
99999.	027	99	99	107	18	18
99999.	028	42	STD	108	54	)
99999.	029	98	98	109	22	INV
83189.	030	85	+	110	74	SM*
37538.	031	01	1	111	96	96
55811.	032	85	+	112	01	1
68626.	033	42	STD	113	94	+/-
72127.	034	97	97	114	44	SUM
78817.	035	85	+	115	96	96
15972.	036	02	2	116	44	SUM
13488.	037	09	9	117	97	97
6766.	038	54	)	118	97	DSZ
67336.	039	42	STD	119	98	98
72099.	040	96	96	120	00	00
77339.	041	82	HIR	121	87	87
5513.	042	16	16	122	43	RCL
34427.	043	33	X ²	123	95	95
44270.	044	85	+	124	32	X:T
5568.	045	82	HIR	125	43	RCL
71267.	046	16	16	126	31	31
57975.	047	65	X	127	22	INV
59150.	048	02	2	128	77	GE
15470.	049	65	X	129	01	01
45582.	050	73	RC*	130	39	39
35485.	051	98	98	131	82	HIR
82368.	052	54	)	132	17	17
27210.	053	74	SM*	133	94	+/-
53148.	054	96	96	134	82	HIR
30884.	055	01	1	135	06	06
25468.	056	94	+/-	136	61	GTD
74144.	057	44	SUM	137	00	00
92634.	058	96	96	138	26	26
72776.	059	44	SUM	139	01	1
37696.	060	98	98	140	00	0
	061	97	DSZ	141	82	HIR
	062	97	97	142	67	67
	063	00	00	143	97	DSZ
	064	45	45	144	93	93
	065	82	HIR	145	00	00
	066	16	16	146	19	19
	067	74	SM*	147	73	RC*
	068	99	99	148	99	99
	069	43	RCL	149	99	PRT
	070	99	99	150	01	1
	071	85	+	151	44	SUM
	072	24	CE	152	99	99
	073	85	+	153	97	DSZ
	074	01	1	154	94	94
	075	85	+	155	00	00
	076	42	STD	156	11	11
	077	98	98	157	98	ADV
	078	03	3	158	25	CLR
	079	00	0	159	81	RST

Älmhult 850415

Finputsning av mitt tidigare insända program för beräkning av kvadratroten med 140 siffrors precision. Exekveringstiden i sekunder är cirka med X siffrors precision:

$$T = \frac{X}{4} (3X + 25)$$

140 siffrors precision ger $T = 15575$ sekunder, ungefär
= 4.20 timmar.

Precisionen ändras på steg 16-17

(28 i programmet mult. 5 = 140)

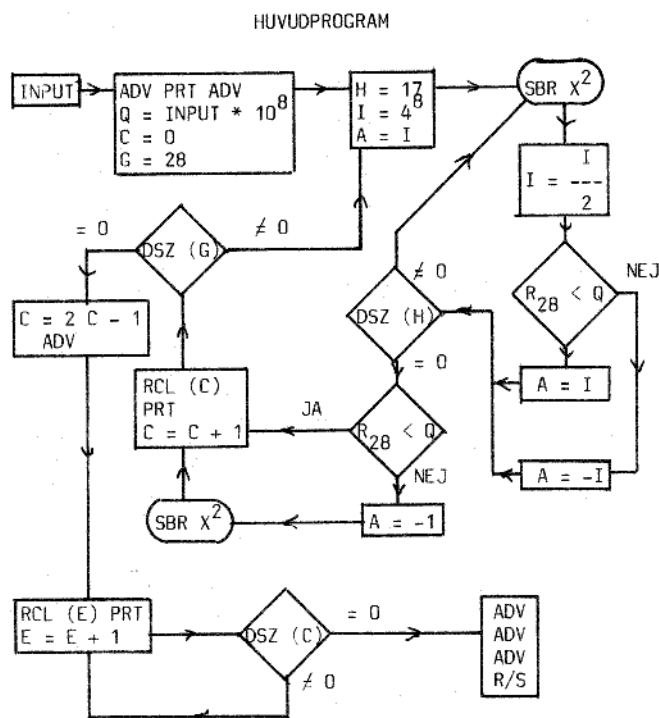
Bifogar flödesdiagram i två delar, dels huvudprogram samt underprogram X2.

Programmet startas med

A RST STF IND 7 EE

Knappa-in talet du vill ha beräknat med formen
 $1 < X < 100$ max 8 decimaler.

MVH Sven-Arne Wallin (P2PG)



CHECKSUM:

BANK 1

559885905.

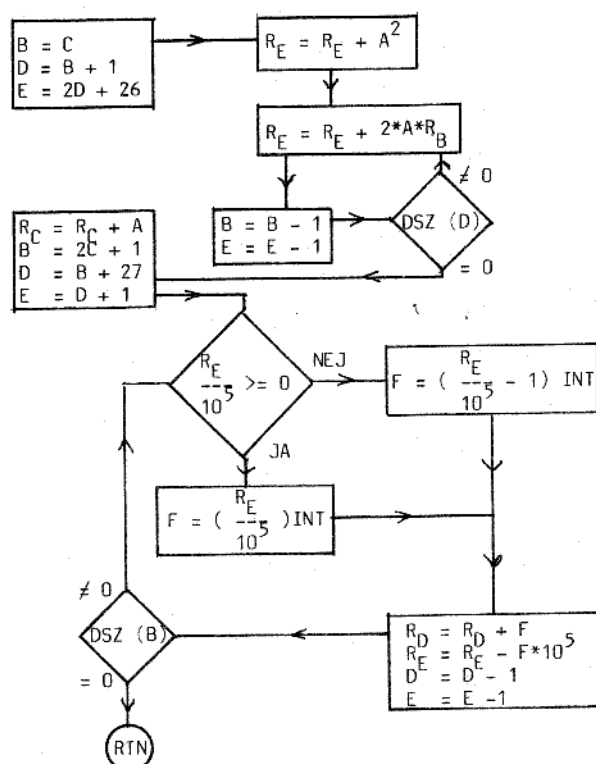
PROG

559685905.

000	92	RTN
001	91	R/S
002	98	ADV
003	99	PRT
004	98	ADV
005	65	X
006	08	8
007	22	INV
008	28	LOG
009	54	)
010	82	HIR
011	05	05
012	09	9
013	69	DP
014	17	17
015	47	CM9

016	02	2	044	42	STD
017	08	8	045	87	87
018	42	STD	046	85	+
019	88	88	047	02	2
020	05	5	048	06	6
021	22	INV	049	54	)
022	28	LDG	050	42	STD
023	82	HIR	051	89	89
024	08	08	052	82	HIR
025	01	1	053	16	16
026	07	7	054	33	X ²
027	42	STD	055	85	+
028	85	85	056	82	HIR
029	04	4	057	16	16
030	33	X ²	058	65	X
031	33	X ²	059	02	2
032	33	X ²	060	65	X
033	82	HIR	061	73	RC+
034	07	07	062	84	84
035	82	HIR	063	54	)
036	06	06	064	74	SM+
037	43	RCL	065	89	89
038	86	86	066	01	1
039	42	STD	067	94	+/-
040	84	84	068	44	SUM
041	85	+	069	84	84
042	01	1	070	44	SUM
043	85	+	071	89	89
			072	97	DSD
			073	87	87

PROGRAMBITEN 85-2

$$\text{SBR } x^2$$


074	00	00	122	01	1	169	61	GTO
075	56	56	123	94	+/-	170	00	00
076	82	HIR	124	44	SUM	171	35	35
077	16	16	125	84	84	172	22	INV
078	74	SM*	126	44	SUM	173	86	STF
079	86	86	127	89	89	174	02	02
080	43	RCL	128	97	DSZ	175	73	RC*
081	86	86	129	87	87	176	86	86
082	65	x	130	00	00	177	99	PR
083	02	2	131	98	98	178	01	1
084	85	+	132	82	HIR	179	44	SUM
085	01	1	133	15	15	180	86	86
086	85	+	134	32	XIT	181	97	DSZ
087	42	STD	135	43	RCL	182	88	88
088	87	87	136	28	28	183	00	00
089	02	2	137	87	IFF	184	25	25
090	07	7	138	02	02	185	02	2
091	85	+	139	01	01	186	49	PRD
092	42	STD	140	72	72	187	86	86
093	89	89	141	22	INV	188	97	DSZ
094	01	1	142	77	GE	189	86	86
095	54	)	143	01	01	190	51	BST
096	42	STD	144	49	49	191	98	ADV
097	84	84	145	01	1	192	73	RC*
098	73	RC*	146	94	+/-	193	84	84
099	84	84	147	65	x	194	99	PRT
100	55	+	148	53	(	195	01	1
101	82	HIR	149	02	2	196	44	SUM
102	18	18	150	82	HIR	197	84	84
103	75	-	151	67	67	198	97	DSZ
104	22	INV	152	82	HIR	199	86	86
105	59	INT	153	17	17	200	01	01
106	29	CP	154	95	=	201	92	92
107	77	GE	155	97	DSZ	202	98	ADV
108	01	01	156	85	85	203	98	ADV
109	12	12	157	00	00	204	98	ADV
110	75	-	158	35	35	205	25	CLR
111	01	1	159	43	RCL	206	81	RST
112	54	)	160	28	28	207	76	LBL
113	74	SM*	161	22	INV	208	11	A
114	89	89	162	77	GE	209	25	CLR
115	65	x	163	01	01	210	93	
116	82	HIR	164	75	75	211	01	1
117	18	18	165	86	STF	212	34	FX
118	54	)	166	02	02	213	33	X ²
119	22	INV	167	01	1	214	35	1/X
120	74	SM*	168	94	+/-	215	91	R/S
121	84	84				216	00	0

LT-SUPPORT

av Anders Persson

LT-Writer har funktioner som är omöjliga att programmera i BASIC. LT-Support ger tillgång till ett effektivare sätt att lagra text på kassett än BASIC kan erbjuda. Dessutom åstadkommer det den speciella cursorn som LT-Writer har. Jag ska försöka förklara hur LT-Support fungerar, och i viss mån varför det ser ut som det gör. Det kan kanske vara till hjälp för den skara mer eller mindre förtvivlade personer som börjar ge upp hoppet om att det överhuvudtaget är möjligt att lära sig assemblerprogrammering.

STRÄNGHANTERING

För att det ska vara lättare att förstå varför LT-Support fungerar och ser ut som det gör, börjar jag med dess bakgrund. Eftersom LT-Writer skapades med avsikt att ge ett så bra ordbehandlingsprogram som möjligt i ett minimalt system, ingick det i förutsättningarna att både programmet och den text som skrevs skulle lagras på kassett. Om man inte kan gå utanför de ramar som BASIC ger, är man hänvisad till den ordinarie filhanteringen. Det är den som används i miniversionen av LT-Writer, en version som verkligen klarar sig med ett minimalt system. Tyvärr blir lagringen mycket ineffektiv, eftersom bandspelaren måste stanna och starta mellan varje rad som lagras. Dessutom måste alla rader vara lika långa på kassetten, vilket ytterligare begränsar överföringshastigheten. Korta rader fylls ut med blanka. De olika problemen bidrar till att tiden för lagring och inläsning förlängs flera gånger om. Exakt hur mycket beror på hur texten ser ut.

Det var inte speciellt svårt att inse att användandet av LT-Writer skulle bli avsevärt trevligare om vi kunde utnyttja samma lagringsmetod som datorn använder för BASIC-program. Genom att dessutom tillämpa variabel postlängd för raderna som lagras på kassetten, skulle vi kunna minimera tiden så mycket som maskinen, inte programspråket, medger. I grund och botten är det naturligtvis fortfarande det program som utgör datorns kassett-DSR (DSR = Device Service Routine, dvs ett program som kommunicerar med någon typ av yttre enhet) som sätter gränsen, men sådana härklyverier kan man glatt lämna därhän just nu. Att det inte gick att åstadkomma detta utan ett assemblerprogram var uppenbart. Tyvärr tillåter 99:ans synnerligen säregna konstruktion inte assemblerprogram utan minnesexpansionen. Det är alltså priset för den ökade snabbheten. Man kan vända på resonemanget och säga att om man inte har extra minne kan man inte skriva så mycket, och alltså tar det inte så lång tid att lagra. Hur som helst, att programmet uppför sig bättre med extra minne är väl inte någon direkt överraskning?

Nu är det, i vanlig ordning, inte slut på krånglet här. Även om det förvisso går att köra assemblerprogram med X-BASIC när datorn har mer minne, är det däremot inte möjligt att SKRIVA dem. För att undvika att få telefonen blockerad är det väl bäst att jag tillstår att det trots allt faktiskt går, åtminstone i teorin. Det här verkar väl snurrigare än vanligt, är jag rädd. Förklaringen finns i Programbiten 84-1, sidan 10. Det är alltså fullt möjligt att skriva assemblerprogrammet på papper, assemblera för hand, göra om hexadecimala koder till decimala och skriva ett BASIC-program som laddar in koderna i minnet, men den sysselsättningen torde i självsöddande omfattning vara fullt jämförbar med att räkna grässtrån på golfbanor.

I praktiken behöver man Editor/Assembler-modulen, skrivminne och helst skrivare. Jag har utnyttjat den utrustningen på min maskin, medan Lennart har skrivit BASIC-delen på sin, utan att behöva mer än Extended BASIC och extra minne. Jag nämner detta bara för att ni som aldrig har skrivit assemblerprogram men ändå, förhoppningsvis, läser detta ska veta vad som behövs.

Nåväl, tillsammans hade vi all nödvändig utrustning. Det "enda" som återstod var att fundera ut ett lämpligt sätt att hantera texten på, så att vi skulle kunna göra kassettagringen så snabb som möjligt. Den uppgiften överlät Lennart med varm hand till mig...

Praktiskt nog ger den befintliga filhanteringen en möjlighet att lagra en del av minnet, precis som det ser ut, på kassett. Det är den funktionen som används när datorn lagrar BASIC-program. Dunt nog gäller detta bara för VDP RAM, dvs den del av minnet som videoprocessorn bestämmer över. BASIC har visserligen strängarna i VDP RAM från början, men de kan vara utspridda i stort sett hur som helst. För att lagringen ska vara effektiv måste de finnas intill varandra. Dessutom kan man inte på assemblernivå härja hur som helst i VDP RAM utan att förlora möjligheten att återvända till BASIC.

Av dessa anledningar bestämde jag mig för att utnyttja 8k-delen av minnesexpansionen till att packa strängarna i, innan de flyttades till VDP RAM för lagring på kassetten. Då X-BASIC mycket väl kan utnyttja praktiskt taget hela VDP RAM när LT-Writer används, är det inte säkert att det finns plats över för bufferten som behövs för filhanteringen. Därför lät jag helt enkelt bufferten ta den plats den behövde, oberoende av vad som eventuellt fanns där innan. För att kunna återvända till Extended BASIC blir det, med den metoden, nödvändigt att spara det som fanns i VDP RAM, så att man kan flytta tillbaks det igen innan återhopp till BASIC sker. Gör man inte det går det åt pepparn till ögonblickligen. Alltså gällde det att leta upp någon vrå i minnet där jag tillfälligt kunde stuva undan det som fanns i VDP RAM innan jag flyttade dit texten. Nu är det nog dags att titta på en minneskarta innan jag fortsätter det här!

2000 Assemblerprogram som laddas när man gör CALL INIT i BASIC.

24F4 Buffert som utnyttjas av stränghanteringsrutinerna i LT-Support. Omedelbart efter starten laddas istället LT-Loadfile här. Det programmet läser in LT-Support från kassetten.

3FB8 DEF-tabellen för alla externdeklarerade labels i LT-Support.

4000 DSR, Extended BASIC ROM mm.

A000 Själva programmet LT-Support.

LAST Efter assemblerprogrammet finns en buffert som utnyttjas vid lagring respektive inläsning av text. Den verkliga adressen beror på hur stort LT-Support är.

Var bufferten slutar beror i sin tur på hur stort BASIC-programmet är. LT-Support räknar ut hur stor plats bufferten får uppta. Större buffert ger snabbare program.

???? BASIC-programmet LT-Writer. Först numeriska värden, sedan radnummertabellen och programraderna.

FFFF

DEN CIRKULERANDE BUFFERTEN eller RUNT, RUNT, RUNT...

Som synes finns det ett utrymme mellan LT-Support och LT-Writer som kan användas som buffert. Om det är tillräckligt stort för att rymma allt som behöver sparas är problemet löst. Men eftersom jag hade, och har, funderingar på att utnyttja LT-Supports funktioner i en del andra program, kunde jag inte på förhand veta hur stort utrymme som skulle vara tillgängligt. Om det är mindre än vad man inbillar sig kan man skriva över och förstöra BASIC-programmet. Därför bestämde jag mig för att göra programmet tillräckligt flexibelt för att klara olika situationer, men också intelligent nog för att alltid utnyttja det aktuella läget så bra som möjligt. Skulle det tas i, så skulle det!

Programmet måste alltså kontrollera hur stor buffert som är tillgänglig. På adressen >8386 ska finnas uppgift om var BASIC börjar använda minnet, dvs var frågetecknen på minneskartan är belägna i verkligheten. Så påstår i alla fall tidningen "The Smart Programmer". Det visade sig vara lite si och så med det, varför jag tog till en säkerhetsmarginal om 100 bytes. Det har fungerat vid alla prov hittills. Var LT-Support slutar är lättare att avgöra. Jag satte helt enkelt etiketten (labeln) LAST efter den sista instruktionen. Bufferten börjar omedelbart efter programmet.

Om bufferten rymmer allt som behöver sparas, är det enkelt ordnat. Det enda som behövs är att flytta VDP data till bufferten, utföra filhanteringen och flytta tillbaka det som finns i bufferten till VDP RAM. Men lagra 7000 bytes i ett 2000 bytes stort utrymme är inte lätt. För att klara sådana fall fick jag tillgripa ett slags cirkulerande buffertsystem. Allt flyttas i omgångar om tre steg. Ju mindre bufferten är, desto fler sådana omgångar behövs. Själva flyttningen går till så här:

Flytta så mycket som möjligt från VDP RAM till bufferten.

Flytta en del av texten till VDP RAM.

Flytta innehållet i bufferten till det ställe som texten kommer ifrån.

Med siffrorna ovan kan man flytta högst 2000 bytes i taget. Den manövern får man göra om tre gånger, och som avslutning flytta runt 1000 bytes. Det är ju lätt att räkna ut hur många flyttningar som behövs, och hur många bytes som blir över till sist.

När allt flyttande är färdigt har texten och data i VDP RAM bytt plats. Genom att göra om cirkulationen kommer man tillbaka i samma läge som rädde när man började.

Den här metoden klarar av kravet på flexibilitet. Även fullt utnyttjat fungerar programmet oavsett om bufferten råkar rymma 20000 bytes eller bara en enda. Det skiljer naturligtvis på exekveringstiden, men bara från omedelbart till 5-6 sekunder. Lagringen på kassett tar ner än en och en halv minut...

Nu var det ju meningen att flyttandet inte ska vara krångligare än vad som behövs i varje enskilt fall. Det blir alltså nödvändigt att varje gång räkna ut vad som behövs för tillfället, men hur det går till återkommer jag till när jag beskriver lagring och läsning på kassett.

DATASTRUKTUR eller HUR SER DET UT I MINNET?

Innan strängarna lagras på kassetten ska de packas ihop intill varandra. Detta göres alltså i minnesexpansionens 8K-del. Den utnyttjas också av Extended BASIC när man gör CALL INIT, men det finns två pekare som talar om hur stort utrymme som är ledigt. LT-Support använder dessa för att räkna ut hur mycket plats det finns över till texten. I och med att pekarna utnyttjas till detta kan man låta Extended BASICs LOAD-funktion ladda assemblerprogram från diskett till 8K-delen först. Dessa kommer att lämnas orörda av LT-Support. Dessutom ändrar LT-Support på pekarna för att få BASIC att tro att hela minnet redan är använt. Därigenom kan LOAD inte ladda in assemblerprogram som förstör strängarna.

Lennart beskriver hur rutinerna i LT-Support ska anropas. STSIO används för att stoppa in strängar i minnet, STRCL för att hämta ut dem igen. Varje sträng lagras med en byte som talar om hur många tecken den innehåller, följt av texten själv. När STRCL hämtar ut strängen tittar den efter hur många tecken som ingår, och hämtar ut dessa. Men båda rutinerna måste veta var i minnet de ska titta. Därför har båda två en parameter som ska ange strängens adress. För att det ska vara lätt att lagra eller läsa flera strängar i följd, ändras pekaren efter varje anrop till att peka på den första lediga adressen efter den sträng som just lästes (eller lagrades). På så sätt kommer parametern automatiskt att peka på nästa strängs längdindikator.

Nå, hur ska man då veta vad pekarens värde ska vara första gången? Jo, innan man kan utnyttja strängfunktionerna, måste STINIT anropas. Den talar då om var man ska börja. STINIT i sin tur tar fram adressen genom att kopiera BASICs pekare till den första oanvända byten.

PÅ KASSETT

Möjligheten att spara strängarna är den näst mest komplicerade funktionen. Den överträffas bara av möjligheten att läsa in dem igen. Det finns två problem som kräver en del eftertanke.

Det första är det faktum att det inte finns någon färdig rutin för filhantering från assemblerprogram. Inte när de är avsedda att anropas av Extended BASIC, vill säga. Det finns det om man skriver program som ska anropas med hjälp av Mini Memory eller Editor/Assembler-modulerna. Å andra sidan har dessa färdiga rutiner svagheten att antingen inte klara av kassettpandspelaren eller också bara den. LT-Support innehåller en DSRLNK som klarar vilken sorts filhantering som helst. Det är för övrigt en modifierad version av den som Björn Gustavsson använt för inladdning av föreningens Forth med Extended BASIC. Tack vare att alla sorters filer kan användas går det lika bra att lagra texten på diskett som på kassett.

Det andra problemet är kravet på filbuffert i VDP RAM. Ovan beskrev jag principen med den cirkulerande bufferten, men frågan har fler sidor än så.

Antag, som tidigare, att man maximalt kan behöva lagra 7000 bytes på kassett. Bufferten rymmer 2000 bytes. Om man bara har 600 bytes strängar i minnet, finns det ingen anledning att tillgripa cirkulationsmetoden. Då är bufferten stor nog för att rymma allt som behöver sparas. Men om 4500 bytes ska lagras måste den andra metoden tillgripas. Vad man måste tänka på är att ett visst sätt att spara VDP data på, kräver ett motsvarande sätt för att återställa det som sparats. Gjorde man på det ena sättet före filhanteringen, måste man göra på motsvarande sätt efteråt.

Det är väl inte så farligt vid lagring av texten, eftersom man då vet hur många bytes det rör sig om innan lagringen påbörjas. Men vid inläsning vet man inte förrän läsningen är klar hur många bytes det handlar om. Dessutom är STLOAD, som sköter om lagringen, så konstruerad att om man har några strängar i minnet och sedan läser in flera, kommer de nya att placeras efter de första i minnet. Detta förutsätter att det finns plats. Det antal bytes som kan läsas in beror alltså på hur mycket som redan finns i minnet när inläsningen göres.

Om bufferten är större än det antal bytes som för tillfället är lediga för strängarna, är allt gott och väl. Då sparar man VDP data i bufferten, läser in från kassett, flyttar lästa strängar från filbufferten till textminnet och återställer VDP RAM från textminnet. Men om det finns plats för 4400 bytes, men bufferten bara är 2000 bytes stor, kan man inte göra så.

Eftersom det är ointressant vad som finns i den lediga delen av strängminnet, kan man flytta VDP data direkt dit. Då finns det på plats innan karusellen ska igång igen. Efter inläsningen finns det två olika alternativ. Enklast blir det om bara 1100 bytes lästes in. Då kan bufferten ta alltihop på en gång. Det som måste göras är att flytta filbufferten till den andra bufferten, återställa VDP RAM från textminnet och flytta den inlästa texten till just textminnet. Men om 3800 bytes lästes in, måste cirkulationsmetoden tillgripas igen.

En ytterligare komplikation är att det blir 600 bytes över, som sparats från VDP RAM, men inte har någon motsvarighet i inlästa tecken.

För att STLOAD ska kunna veta hur många tecken en fil innehåller, placerar STSAVE antalet tecken först i filen.

FLIPPEN eller DEN ANPASSNINGSBARA CURSORN

Den speciella cursor som LT-Writer har kan kanske betraktas som en plog. Vi kom på den en kväll när vi satt några stycken hos Lennart och diskuterade datorer i allmänhet och LT-Writer i synnerhet. Någon tyckte att man skulle ha en cursor vars utseende avspeglade det tecken den för tillfället täckte. Genom att skriva en avbrottsdriven rutin borde det gå att ordna. Avbrott (interrupt) genereras 50 gånger i sekunden, och vi tyckte att det borde vara tillräckligt för att cursorn skulle hinna ändra utseende när man flyttar den mellan olika tecken.

Jag gick hem och försökte senare samma kväll, och lyckades ganska fort, tack vare lite tur. När jag hade fått det att fungera skrev Lennart om LT-Writer, för att utnyttja den nya funktionen. Eftersom datorn ska avbryta det den håller på med var 20:e millisekund för att kolla hur cursorn ska se ut, minskar kapaciteten lite grand. Jag mätte upp en minskning på ungefär 5%, men jag har snabbat upp rutinen lite grand sen dess. Då 99:ans BASIC inte är snabb så det stör utan extra påhäng, har Lennart konsekvent stängt av specialcursorn så fort den inte är på skärmen. Arbetsintensiva uppgifter som justering av texten drabbas alltså inte av förluster i datorns kapacitet. Det bara tar tid ändå...

Efter att ha testskrivit med LT-Writer anser jag att id:n faktiskt har en praktisk biverkning. Det är nämligen mycket lätt att se om man håller på att skriva in något nytt, eller om man har backat tillbaka in i redan skriven text.

När Du nu känner till bakgrunden till LT-Support, kan jag beskriva källkoden. Jag tar den uppifrån och ner.

Till att börja med finns ett antal EQU- och DATA-direktiv, vilka används i de olika delarna av LT-Support.

Därefter finns en beskrivning av hur registren användes.

Den första rutinen är DSRLNK. Den fungerar enligt beskrivningen i Editor/Assembler-manualen. Rutinen innehåller en speciell GPLLNK, vilken anropar LINK-programmet i GPL-delen av operativsystemet. Eventuella fel meddelas i processorns statusregister.

GETNUM fungerar som den färdiga rutinen NUMREF, men gör även en omvandling från flyttal till heltal (CFI). Genom att anropa omvandlingen med XMLNK slipper man att själv spara R11 i GPLWS.

PUTNUM innebär motsatsen till GETNUM. Omvandlingen gäller naturligtvis heltal till flyttal. Där använder jag det omvandlingsprogram som finns i Extended BASIC-modulen. Även här är det enklast att utnyttja XMLNK för anropet.

FINSIZ räknar ut hur stor buffert det finns plats för mellan LT-Support och BASIC-programmet. FINSIZ fungerar oberoende av om LT-Support finns i 8k- eller 24k-delen.

Om buffertstorleken blir negativ ges meddelandet "NO PROGRAM PRESENT", eftersom BASIC-programmet då har skrivit över en del av assemblerprogrammet. Labeln LAST finns omedelbart efter den sista instruktionen i LT-Support.

STINIT är den första rutinen som kan anropas från BASIC. Den tar reda på hur stort utrymme som finns för text och buffert. Efter PUSTRT ges parametern START sitt värde. Eftersom START alltid skall vara den första parametern, kan alla delar av LT-Support använda samma tilldelning.

STFREE räknar ut hur mycket av minnet som finns kvar till att lagra text i. Antalet bytes ges i parameter nummer två.

STCLR fungerar som STINIT med den skillnaden att den inte ändrar på pekarna som BASIC använder. STCLR kan alltså användas hur många gånger som helst, medan STINIT bara kan användas en gång.

OFLNAM läser in ett filnamn från den första parametern till den PAB som används för lagring och läsning av text på kassett. Genom att låta filnamnet vara en parameter klarar assemblerprogrammet av olika sorters utrustning. För att anpassa lagringen till diskett behöver bara BASIC-programmet ändras. Dessutom var det lätt att utveckla programmet på en maskin med diskettssystem.

SETPAB hanterar de delar av filhanteringen som är lika, oberoende av om man lagrar eller läser text.

MOVBUF är det lilla program som åstadkommer en cirkulation genom bufferten. MOVBUF flyttar dock bara en del av hela den text som skall till filbufferten.

CIRCUL heter rutinen som räknar ut hur många gånger MOVBUF måste anropas, samt vilka adresser som ska användas. Instruktionen DIV hos 9900 erbjuder ett mycket enkelt sätt att räkna ut både hur många gånger MOVBUF måste anropas, och hur många bytes som blir över när det inte går jämt ut. Observera att MOVBUF själv modifierar de pekare som utgör dess parametrar.

STSAVE anropas när man ska lagra texten på kassett. Först räknar jag ut hur många bytes som ska lagras. IF1 avgör om cirkulationsprincipen måste tillgripas, eller om det räcker med en enkel flyttning av data. R7 använder jag som en flagga för att efter lagringen veta vilken buffertmetod som använts.

STLOAD läser in texten. Som jag tidigare beskrivit finns det fler alternativ efter inläsningen, vilket gör STLOAD lite mer komplicerad än STSAVE.

STRCL hämtar ut en sträng från minnet. Längdbyten utnyttjas för att räkna ut var nästa sträng börjar.

STSTO är motsatsen, dvs den lagrar en sträng i minnet. Det blir lite krångligare än att hämta ut en sträng, eftersom man måste räkna ut hur stor plats som finns kvar också. Dessutom kan ingen sträng vara längre än 255 tecken, vilket också måste kollas.

FLIP används för att få igång den speciella cursorn. Först testas om den redan är igång. Om så är fallet ska man inte göra något alls. Annars lagras den nuvarande definitionen på cursorn undan, varefter interrupt pekaren laddas med adressen till ADAPT.

NOFLIP stänger av specialcursorn. Även här gäller att om den redan är avstängd ska man inte göra det en gång till. När interrupt pekaren nollställts ges cursorn samma definition som den hade när FLIP anropades.

ADAPT är den avbrottsdrivna rutinen. Jag börjar med att titta efter om cursorn verkligen är på skärmen. Om den inte är det, finns det ingen anledning att krångla med att ändra utseendet på den. Ett annat specialfall är om cursorn står ovanpå ett blanktecken. I så fall får den återta det utseende den hade innan FLIP anropades. Modifieringen av cursorn består av att definitionen för det tecken cursorn täcker läses in, ett byte mot nollor och tvärtom, och den nya definitionen lagras som cursordefinition.

FLIP, NOFLIP och ADAPT har egna processorregister, kallade FLIPUS.

Det var allt. Den som är intresserad av mer information om LT-Support, eller det program som åstadkommer den fil som LT-Writer läser in vid uppstart, rekommenderar jag att ta kontakt med mig eller Lennart Thelander.

Anders Persson
Kämnärsvägen 4 : 1078
222 45 Lund

Lennart Thelander
Dalhemsvägen 103 A (nya adressen)
252 65 Helsingborg

FORTH spalten

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

SIG-FORTH

Är du intresserad av FORTH, men ändå inte fått något brev från SIG-FORTH? I sådana fall vill jag att du tar kontakt med mig. Vi har nämligen bildat en speciell intresse grupp-för FORTH intresserade, vilken främst har kommit till för att vi som håller på med FORTH ska få bättre kontakt med varandra. Om du ringer mig så ska jag förklara.

SIG-FORTH medlemmarna i Stockholm har tom hunnit med att ha sin första träff. Enda felet med träffen var att tiden inte räckte till. Vi var 15-20 personer vilka samlats för att diskutera FORTH och titta på demonstrationer av FORTH program. Bo Carleö demonstrerade ett mycket proffsigt rit-program han gjort. Programmet är på 18 föredömligt skrivna FORTH-skärmar, och dess funktion är så enkel och omfattande att vi alla blev mycket imponerade.

Det har kommit ett demonstrations program från USA, vilket innehåller en hel del olika grafik demonstrationer, och ett i mitt tycke fantastiskt musik program, vilket spelar en fuga av Bach. Jag tror att detta program kan beställas på diskett från föreningen. Jag själv demonstrerade ett disk-baserat FORTH system, som jag kallar FORTH899. Detta kommer att ingå i program banken så småningom (jag fick en del ideer med hem från träffen, så det ska förbättras lite först). Kanske kan Bo Carleös program FORIT beställas från föreningen också? Det kom upp många ideer på träffen, och entusiasmen var inte att ta miste på, så SIG-FORTH kommer att låta höra tala om sig. Var så säkra!

BENCHMARK-TEST

Jag har gjort en Benchmark test (för att få reda på hur snabbt vårt FORTH är), och jämfört med de uppgifter som finns i tidningen BYTE (november -84). Den testen antyder att FORTH på IBM PC är lite drygt dubbelt så snabb som vårt FORTH.

Medan jag ändå var igång och testade, så måste jag en del andra exekveringstider också: skillnaderna mellan 2 / , U* och 2 * , samt skillnaderna mellan 2 / , U/ och / . Resultaten är helt klart värda att notera, ty multiplikation resp division med 2 är ofta förekommande i FORTH. För ett tag sedan lyckades jag halvera exekveringstiden för multi color plottnings rutinen MCHAR i TI-FORTH (vilken föresten har felaktigt stack diagram i TI-manualen, det ska vara (color column row --)), bara genom att sitta och skriva om saker som 8 * till 2* 2* etc.

Här följer nu testrutinerna och testkörningarna:

```
( Benchmark ) HEX 83D6 CONSTANT TIME
Ø En inbyggd timer som räknar upp från 0
Ø och anger hundradelar av en sekund
DECIMAL 214 CONSTANT TNOP
Ø 10000 st NOP tar 214 1/100-sekunder
Ø CORR ( t n -- t' ) TNOP 10000 * / - ;
Ø För att korrigera för anropstiden, vid uppreoning
: .T ( t' -- ) DUP 100 / Ø .R " " s ;
Ø Skriver ut tiden på ett lämpligt sätt
: TIMES ( n TIMES name -- ) Ø R TIME XCOMPILEA ' CFA
" " takes " R Ø 0-TIME "
DO DUP EXECUTE
LOOP DROP ' R Ø CORR .T ;
Ø Upprepar rutinen "name" n gånger
Ø "name" måste ha stackdiagrammet ( -- )
```

```
: TEST0 ;
: TEST1 1234 DROP ;
: TEST2 1234 2 * Ø 2 U/ DROP DROP ;
: TEST3 1234 2* 2/ DROP ;
: TEST4 1234 2 * 2 / DROP ;
: TEST5 1234 2 U* 2 U/ DROP DROP ;
```

```
10000 TIMES TEST0 takes 0.0 s ok
10000 TIMES TEST1 takes 0.80 s ok
10000 TIMES TEST2 takes 8.60 s ok
10000 TIMES TEST3 takes 1.74 s ok
10000 TIMES TEST4 takes 23.26 s ok
10000 TIMES TEST5 takes 6.14 s ok
```

Det är innehållet i testrutinerna som ska mätas, så CORR ser till att tiden för TEST0 blir 0. Innehållet i TEST1 finns med i alla övriga testrutiner, och dennas tid kan dras ifrån de andras:

```
2 * Ø 2 U/ DROP tar 780 mikrosekunder
2* 2/ tar 94 mikrosekunder
2 * 2 / tar 2246 mikrosekunder
2 U* 2 U/ DROP tar 534 mikrosekunder
```

Slutsats: är det bråttom så försök att undvika * och / , vilka är mer än 4 ggr långsammare än U* resp U/ . Obs stackdiagrammen: U* (u1 u2 -- ud) U/ (ud u1 -- u2 u3) , där u2 = resten , u står för tal utan tecken och d står för dubbelt heltal.

När 2 * 2 / är 24 ggr långsammare än 2* 2/ , så förstår man att det är 2* och 2/ som ska användas i tidskritiska rutiner.

I det här numret blev inte spalten på det sätt jag har tänkt mig att den ska bli (redaktören har ont om plats), men jag lovar att komma igen. Mina intentioner är att visa att FORTH kan vara mycket användar-vänligt, samt att ta upp sådant som du skickar in till mig.

GO FORTH

Oj-då

Rita i FORTH 84-4 sid 30

av Lennert Lindberg och Jan Alexandersson

I Michael Dahlquists program för grafik i FORTH nr 84-4 finns ett antal testord. Ett av dem kallat T6 fungerar endast om man skriver:

TO-B T6 (ENTER)

Om man i stället försöker att skriva TO-B (ENTER) och väntar på initieringen och sedan skriver T6 så kommer ej programmet att fungera. Detta beror på att programmet startar så snabbt att (ENTER)-tryckningen även hinner att läsas av T6.

Ordet ENDCASE kompilerar in ett DROP vilket man kan se i PB-FORTH(Föreningens FORTH) med SEE T6. Detta medför att 0 droppas och 13 kommer högst upp på stacken. Detta läses av UNTIL som avslutar BEGIN-UNTIL loopen. En lämplig lösning är att stoppa ett SWAP omedelbart före ENDCASE.

Sedan kan T6 ställas upp på ett mera läsbart sätt, t ex så här(vilket troligen gör det lättare att se hur det fungerar)

```
: T6 128 112
BEGIN ?KEY
CASE 2 OF 1 ENDØF
ASCII E OF 1- 2DUP PLOT Ø ENDØF
ASCII S OF SWAP 1- SWAP 2DUP PLOT Ø ENDØF
ASCII D OF SWAP 1+ SWAP 2DUP PLOT Ø ENDØF
ASCII X OF 1+ 2DUP PLOT Ø ENDØF
Ø SWAP ENDCASE
UNTIL DROP DROP ;
```

Det finns ytterligare en möjlighet till förbättringar. Om man på skärm 81 initierar FØ VARIABLE BCO så kommer plottningen att göras med vit kurva på transparent bakgrund. Det är då möjligt att ändra skärmfärgen till t ex svart eller rött. Om man väljer F4 så blir skärmtecknens bakgrunnsfärg alltid blå oberoende av skärmfärg.

Det är även något lurt med Michaels program om det laddas in i TI-FORTH med nödvändiga tillägg. Om någon har en förklaring till detta kanske vi tar upp det i ett kommande nummer.

GameFORTH

Train

GAMEFORTH -

en FORTH-version av Lars-Erik Svahn

presenterat av Lennart Lindberg

Vår nye - sedan nr 85-1 - spaltredaktör för FORTH-spalten, Lars-Erik Svahn, nöjer sig inte med att skriva text om FORTH, minsann. Högst egenhändigt har han tillverkat en egen FORTH-version, baserad på föreningens FORTH. Den rymmer på nitton tilläggsskärmar. Han kallar den GAMEFORTH därför att den är särskilt lämpad för den som vill skriva spelprogram. Den innehåller ord för sprites, grafik, prat med pratör, ljud, flyttal och trigonometri förutom en del allmänna ord som Lars-Erik tycker är bra att ha.

De här godsakerna finns på en skiva som du kan på vanligt sätt beställa genom föreningen till det vanliga priset, 60:- kr. Dokumentationen kommer att finnas på skivan för utskrift eller på papper utskriven - vilket är inte klart ännu. Här nedan antyder jag bara vad skivan innehåller.

Sprites, 2 skärmar
Det finns ord för att definiera, färga, lägesbetämma och röra sprites, för att skapa en vägg som sprites studsar emot, för att beräkna taxi-avståndet mellan sprites, för att gömma och radera sprites mm.

Grafik, 1 skärm
Här finns ord för textmode, för grafikmode, för att färga skärmen, för teckendefinitioner och viss plotting mm.

Prat, 2 skärmar
Du kan initiera prat, uttala ord och tysta pratör.

Ljud, 1 skärm
Du kan definiera, ladda och aktivera ljudvariabler.

Flyttal, 2 skärmar
Du kan definiera och ladda flyttalsvariabler och flyttalsmatriser (fält) samt hantera flyttal med de vanliga räknesätten plus trigonometriska funktioner.

Datastrukturer, 3 skärmar
Här finns ord för att hantera en särskild stack i VDP-minnet, ord för stränghantering och ord för sträng- och heltalsmatriser. Möjligheterna att lagra data i VDP-minnet ger ett effektivare minnesutnyttjande. Du hittar här också ord för hantering av mängder med mängdlogik (union, snitt, symmetrisk differens mm).

Miniassembler, 1 skärm
Det här är en liten flexibel, minnessnål assembler som utmärkt väl klarar de korta maskinkodsrutiner som kan bli aktuella i primitiva FORTH-ord. Den kan användas i stället för den assembler som finns i föreningens FORTH. Tar man bort den tillsammans med dis-assemblern tjänar man 4,5K minnesutrymme. Det finns ett ord CUT som kan göra det trots att den koden ligger i skyddat utrymme. Ordet fungerar som FORGET men hejdas ej av minnesskyddet (FENCE).

Trigonometri, 2 skärmar
Det handlar om ord för heltalstrigonometri - grader eller radianer, sinus, cosinus, tangens och arc-tangens.

Diverse, 5 skärmar
Olika nyttiga ord finns här, t ex
RPICK som beter sig som PICK men på returstacken
TOR som är ROT omvänt
BYE som beter sig som i Basic
JS som tar hand om joystickvärden
VAR: som fungerar som CREATE i polyFORTH (i Starting FORTH)
GN som hämtar tal från tg-bordet till stacken
S+ som adderar till tal nere i stacken

Som sagt - den som vill använda GAMEFORTH kan köpa det från föreningen för en kostnad som motsvarar disken, kopieringen och sändandet.

Det vore väl inte snyggt att inte tala om att Lars-Erik redan håller på och är i stort sett färdig med FORTH899, som är en FORTH-version som innehåller det bästa - enligt Lars-Eriks smak - ur föreningens FORTH, TI 3.0 och GAMEFORTH.

DOPPLER-EFFEKT

av Lennart Lindberg

Den som stått och sett ett tåg fara förbi vet, att om tåget samtidigt tjuver med visslan förändras ljudets höjd (frekvens) vid passagen. Medan tåget närmar sig hörs frekvensen hög men när det är på väg bort låter ljudet lägre.

Fenomenet kallas dopplereffekten.

Lars-Erik Svahn har satt ihop ett program som illustrerar det här med hjälp av 99-ans tongenerator. Han har just valt exemplet med tåget och kallar sitt program för TRAIN.

Jag har listat programmet här intill med en speciell listteknik som ger en lista med 28 teckens bredd - precis som när man listar Extended Basic-program på skärmen. Vitsen med det är att man lätt ser eventuella fel när man knappar in programmet på skärmen.

Jag skulle uppskatta att få reda på Dina synpunkter på den här listningen och när den kan vara bra jämfört med den vi vanligen använder - där radnumren kommer för sig till vänster och där hela kolumnbredden utnyttjas.

```
100 ! *****
110 ! * TRAIN *
120 ! * LARS-ERIK SVAHN *
130 ! * 08-742 61 07 *
140 ! *****
150 !
160 CALL CHAR(91,"0028003844
7C4444")
170 CALL CHAR(116,"3004E062F
FFF9393")
180 CALL CHAR(117,"1C1C083E8
A182424")
190 CALL CHAR(118,"00FFAAAA
AAAA")
200 CALL CHAR(119,"103070FF7
0301")
210 CALL CHAR(120,"080C0EFF0
E0C08")
220 CALL CHAR(121,"081C3E7F0
8080808")
230 CALL CHAR(122,"080808087
F3E1C08")
240 CALL CHAR(124,"000000FF")
)
250 OPTION BASE 1
260 DIM F(600),D(600)
270 CALL CLEAR :: PRINT " P
ROGRAMMET DEMONSTRERAR:"
DOPPLER-EFFekten." ::
: : : : :
280 CALL HCHAR(15,9,119):: C
ALL HCHAR(15,10,124,2):: CAL
L HCHAR(15,13,83):: CALL HCH
AR(15,15,124,2):: CALL HCHAR
(15,17,120)
290 CALL HCHAR(16,7,118,19)
300 CALL HCHAR(17,9,116):: C
ALL HCHAR(17,18,121)
310 CALL HCHAR(18,8,124,2)::
CALL HCHAR(18,10,120):: CAL
L HCHAR(18,18,65)
320 CALL HCHAR(19,9,86):: CA
LL HCHAR(19,18,122)
330 CALL HCHAR(20,18,117)
340 CALL SCREEN(12):: INPUT
"A (m): ":A
350 CALL SCREEN(11):: INPUT
"S (m), V (m/s): ":S0,V
360 PRINT "VANTA..."
370 DT=.18 :: N=0
380 F1=400 :: F2=500 :: F3=9
00
390 K=A2 :: X=9.32 :: VL=33
1 :: T=-DT*1200 :: DS=V*DT ::
Y=27.9
400 FOR S=-S0 TO S0 STEP DS
410 N=N+1 :: R=SQR(K+S2)::
F(N)=1+S*V/R/VL :: D(N)=MIN(
MAX(X*LOG(R)-Y,0),30):: NEXT
S
420 PRINT ::
430 PRINT " 1: ANDRA PARAME
TRARNA:" 2: LJUD-DEMONSTRA
TION:" 3: SLUT"
440 CALL SCREEN(10)
450 CALL KEY(3,Z,KEY):: IF N
OT KEY THEN 450 ELSE CALL SC
REEN(5):: Z=Z-48 :: ON MAX(M
IN(4,Z),1)GOTO 270,460,470
460 FOR I=1 TO N :: CALL SOU
ND(T,F1/F(I),D(I),F2/F(I),D(
I),F3/F(I),30,-8,D(I)):: NEX
T I :: GOTO 440
470 CALL CLEAR
480 END
```

FORTH

FORTH
Björn Gustavsson

Jag hade inte tänkt använda en 99:a igen, och hade därför lämnat alla mina FORTH-skivor till min bror så att han kunde sammanställa all källkod för föreningens FORTH. Men när jag besökte honom under en helg, insåg jag att ingen utom jag själv någonsin skulle kunna pussla ihop källkoden, eftersom den var utspridd på ett halvt dussin disketter. Därför sammanställde jag källkoden under den helgen.

Källkoden kommer att distribueras genom föreningen på samma sätt som FORTH. Den här artikeln är ett litet försök att dokumentera källkoden. Eftersom jag inte kan gå in på allting i detalj kommer jag främst att behandla de delar som är svårbegripliga.

Mer källkod kan komma att grävas fram; jag kommer att överlämna alla disketter med FORTH-källkod till Lars-Erik Svahn.

För närvarande är följande källkod tillgänglig:

Källkoden för programladdarna för Extended BASIC och Mini Memory skrivna i assembler (Editor/assembler krävs för att assemblera).

Två disketter med allt som behövs för att upgradera TI FORTH 3.0 till föreningens FORTH, inklusive editor, skrivarrutiner, disassembler, assembler etc. Detta är skrivet i FORTH med en liten del i assembler (som assembleras under FORTH). Assemblern och disassemblern är något ändrade jämfört med den variant som finns på systemskivan. (Jag har begått dödssynden att inte spara den gamla källkoden.) Eftersom innehållet skiljer sig något från den systemskiva som skickats ut har den versionen fått numret 2.1 i stället för 2.0.

En diskett med en GPL-disassembler och en Z80-disassembler, båda skrivna i FORTH. Den nya versionen av disassemblern som finns version 2.1 måste vara laddad för att det skall fungera.

HUR PB FORTH UTVECKLADES

På Mikrodatormassan i Sollentuna våren 1983 fick jag tag på TI FORTH, men helt "naken" utan alla skärmar med källkod. Skivan hade troligen kopierats på TI:s huvudkontor i Europa med hjälp av Disk Manager vilken endast kopierar vanliga filer.

Jag var alltså tvungen att göra mina egna hjälpord, och jag började med att mata in en liten editor från FORTH DIMENSIONS rakt av utan att kunna spara källkoden. Sedan använde jag den lilla editorn för att mata in källkoden för en bättre, men ändå radorienterad editor.

När jag hade en editor skrev jag andra hjälprutiner som jag behövde för att kunna utnyttja FORTH bättre. Det blev en skärmorienterad editor, en disassembler för att kunna lista ut hur TI hade implementerat FORTH, skrivarrutiner som kunde hantera två eller flera skrivare (jag hade en matris-skrivare och skönskrivare inkopplade samtidigt) och hel del annat.

Någon gång under hösten 1983 fick föreningen tillstånd av TI att distribuera FORTH med mina utökningar. Jag började arbeta på att göra FORTH körbar med andra moduler än Editor/assembler och kassettbandspelare. Det gav upphov till en rad problem.

På vilket sätt skall FORTH lagras på skiva eller kassett, så att det kan laddas med alla moduler och från kassettbandspelare? TI:s format var otänkbart eftersom det inte går att ladda från bandspelare. Det enda typen av filer som kan laddas snabbt från bandspelare är programfiler (det format som BASIC-program lagras i). Eftersom Editor/assembler ("5 Run program file") kan köra programfiler både från skiva och kassett var det naturligt att använda det programfilsformatet.

Program i det formatet består av en eller flera programfiler som är högst 8K långa. Sådana program laddas helt enkelt rakt in i minnet på en fast adress. Det är relativt enkelt att skriva en laddare för programfiler som kan köras med Mini Memory eller Extended BASIC.

Vad som återstod var att ändra FORTH så att det skulle kunna lagras i detta programfilsformat.

Först behövdes ordet SAVE-SYSTEM (CSAVE-SYSTEM) som sparar hela FORTH som det ligger i minnet på en skiva (kassett). Källkoden för dessa ord finns på skärm 27-29, del 1. SAVE-SYSTEM och CSAVE-SYSTEM sparar först hela ordlistan som finns i 24K-delen av minnesutbyggnaden i två eller tre programfiler. I den sista programfilen lagras assemblerkoden som finns i 8K-delen. I den filen lagras också definitionen av alla tecken. Detta görs för att den som använder bandspelare inte skall vara tvungen att ladda tecken från bandspelare; den som har skivenhet kan ju lätt ladda tecknen från en skärm.

När FORTH skall laddas in från en programfil behövs rutiner som bl a initierar VDP-register och startar interpreteringen av FORTH-kod. I TI:s version finns dessa rutiner i det assemblerprogram som finns i filen FORTH. Men dessa rutiner försvinner så fort FORTH är startat. För att göra det lätt för mig skrev jag bara den kod som startar interpreteringen av FORTH-kod i assembler (källkod del 2, skärm 54). Den finns i en ledig del av 8K-delen av minnet; den kopierar bara FORTHs sk inre interpretator till det snabbare CPU-minnet i datorns grundenhet och startar interpretatorn.

Det första ordet i högnivåkod som körs är COLD (källkod del 1, skärm 26). Det initierar alla VDP-register och tömmer skärmen. Teckendefinitionerna som sparades av SAVE-SYSTEM kopieras sedan till sin plats i VDP-minnet. Vissa FORTH-variabler, för bl a blockbuffert-hantering, initieras. WARNING testas sedan för att se om programfilen sparades med SAVE-SYSTEM eller CSAVE-SYSTEM.

Om det är ett kassettsystem exekveras CS (källkod del 1, skärm 25). CS ändrar CFA-pekaren i R/W så att CR/W utförs i stället. R/W utför normalt alla läsningar och skrivningar av skärmar på en skiva. CR/W läser och skriver skärmar på kassettspelare. Skärmar på kassettspelare lagras som programfiler.

Låt oss återvända till COLD, och det svåraste problemet som jag stötte på. Ett ord som heter PATCH anropas. Problemet jag hade var GPL, Graphics Programming Language, ett interpreterat, assemblerliknande språk som TI hittade på av främst två anledningar: Det är mycket lättare att hantera VDP-minnet i GPL än i assembler, eftersom instruktioner för att läsa och skriva i VDP-minnet har byggts in i direkt i språket. GPL-program kan endast köras om de ligger i GROM, ett av TI specialtillverkat ROM med en inbyggd programräknare. Eftersom endast TI kunde tillverka GROM, kunde endast TI göra moduler för TI-99/4A.

Tyvärr har vissa nödvändiga systemrutiner skrivits i GPL, däribland rutinerna för kassettspelare. Därför var det helt nödvändigt att jag kunde anropa GPL-rutiner oavsett vilken modul som var ansluten. Problemet ligger i att kunna återvända från en GPL-rutin. En förutsättning är att man vet adressen till en lämplig XML-instruktion i ett GROM. (XML-instruktionen fungerar på liknande sätt som XMLLNK.) Det finns XML-instruktioner i både Mini Memory, Extended BASIC och Editor/assembler, men de är av olika typ och ligger på olika platser.

Det finns några lämpliga XML-instruktioner i de GROM som finns i grundenheten, men man kan inte vara säker på att de ligger på samma adress i alla 99:or som finns ute.

Jag löste problemet genom att skriva ordet PATCH (källkod del 1, skärm 21-22) som söker efter en XML-instruktion varje gång FORTH startas upp. Jag vet att det finns åtminstone två olika varianter GROM i de TI-99/4A som har tillverkats. Föreningen skulle kanske kunde undersöka hur många olika typer av GROM det finns, och om det finns en lämplig XML-instruktion på samma plats i alla, i vilket fall PATCH skulle kunna tas bort.

GPL-problemet gör det nödvändigt att flytta på returstacken eftersom de XML-instruktioner som finns i grundenhetens GROM använder hoppvektorer vid >3FC0 och >3FEO. (Källkod del 2, skärm 55.)

När man har en fungerande GPLLNK kan man använda den för att definiera DSRLNK (källkod, del 1, skärm 20). Denna DSRLNK fungerar för alla typer av enheter, både skivenheter och kassettspelare. Den anropar en rutin i GPL, som söker igenom alla drivrutiner, oavsett om de är skrivna i assembler eller GPL.

Det finns en hoppinstruktion i början av GROMet (adress 10) som hoppar till den rutinen, men rutinen är skriven så att den endast kan anropas från GPL, eftersom den kräver en parameter som fysiskt skall vara placerad direkt efter anropet (vilket innebär att den måste ligga i GROM). Jag löste problemet genom att hämta adressen i hoppinstruktion och öka den med två för att hoppa över instruktionen som hämtar parametern. Parametern måste då i stället lagras före anropet på adress >836D.

HUR MAN SKAPAR PB FORTH FRÅN TI FORTH

Följande behövs:

TI FORTH version 3.0
Källkod för föreningens FORTH, del 1 och 2
En tom, formaterad diskett
Editor/Assembler

Starta TI FORTH på vanligt sätt. Sätt in källkodsskivan, del 2, i DRO. Skriv:

EMPTY-BUFFERS
0 FENCE !
FORGET TASK
45 LOAD

48 LOAD
REMOVE FLUSH

Tag bort alla högnivåord.
Ladda assemblern högst upp i minnet
så att den kan plockas bort spårlöst.
Ladda vissa praktiska ord skrivna i assembler.
Plocka bort assemblern.

Sätt in källkodsskivan, del 1, i DRO.

3 LOAD FLUSH

Ladda praktiska högnivåord, inklusive editorn.

Sätt in källkod, del 2, i DRO.

3 LOAD
53 LOAD

54 LOAD

FORGET ORG
4 BLOCK UPDATE
5 BLOCK UPDATE
7 BLOCK UPDATE
0 CHAR.ADR 6 VLOAD
128 CHAR.ADR 6 -VLOAD
UPPDATE

Ladda assemblern och disassemblern.
Modifiera assemblern så att den assemblerar till
valfri adress i minnet.
Ladda uppstartningskoden i assembler och gör litet
andra lappningar. Det huvudsakliga syftet
med dessa är att möjliggöra SAVE-SYSTEM och
återstarta med vilken modul som helst.
Tag bort modifieringarna av assemblern.
Förbered kopiering av felmeddelanden.

Hjälpskärm till editorn.
Ladda in den svenska teckenuppsättningen.
Ladda in den inverterade teckenuppsättningen.

Sätt in den tomma disketten i DRO.

FLUSH
SAVE-SYSTEM

Kopiera över block 4, 5, 6 och 7.
Spar hela FORTH-systemet.

Klart!

BOKRESCENSIONER

BOKRESCENSIONER Björn Gustavsson

Här kommer recensioner av Leo Brodies nya **Thinking FORTH** (1984) och Kernighans och Plaunders **The elements of Programming Style** (1978). Båda böckerna diskuterar strukturerad programmering, vilket är anledningen till att jag recenserar dem tillsammans.

Målet med strukturerad programmering är inte att undvika GOTO-instruktioner, utan att framställa program som man vet är felfria och kan uppdateras eller utökas. Detta mål kan uppnås genom att dela upp programmet i moduler så att man kan förstå en del i taget, och att koda varje modul tydligt och lättläst. Båda böckerna ger synpunkter på hur det görs praktiskt.

Thinking FORTH, av Leo Brodie, Prentice-Hall, Englewood Cliffs 1984, 300 sidor.

Leo Brodie har tidigare givit ut **Starting FORTH**, som fortfarande är en oöverträffad lärobok i grunderna i FORTH. Hans andra bok, **Thinking FORTH**, förutsätter åtminstone grundläggande kunskaper i FORTH (bilaga A ger dock en kort överblick). Boken visar hur man programmerar strukturerat i FORTH, men andra språk tas upp som jämförelse.

Boken första tre kapitel innehåller huvudsakligen diskussioner om hur man från en idé konstruerar ett program. Många synpunkter, ibland sinsemellan motstridiga, förs fram av Brodie och ett antal FORTH-programmerare som denne har intervjuat. Det gör det svårt att få någon riktig överblick.

Från och med fjärde kapitlet är boken intressantare. Nu koncentrerar sig Brodie mer på FORTH, med många exempel på hur man skriver kod i FORTH. Ett kapitel handlar bara om hur man skriver läslig FORTH-kod: Hur man organiserar sina skärmar, hur koden läggs upp på en skärm, hur mycket indrag det bör vara, hur mycket luft det bör vara mellan orden, hur man kommenterar och hur man väljer bra namn för sina ord. Jag önskar att de som utvecklade TI FORTH hade läst det kapitlet - deras källkod bryter mot en hel del läsbarhetsregler, bl.a. att man skall lämna luft mellan orden.

Vidare diskuteras hur man bryter ned ett program i små ord, och hur man bryter ut kod som gör en liknande uppgift. Organisation av data och datastrukturer behandlas i ett annat kapitel.

Det sista kapitlet handlar om hur man undviker kontrollstrukturer (if - then etc). Att de över huvud taget skall undvikas beror på att de "gör koden mer komplex. Ju komplexare koden är, desto svårare blir det att läsa och underhålla den".

Tyvärr gör vissa av Brodies trick koden mer svårsläst än motsvarande kontrollstruktur. Bland annat föreslår han några trick som bygger på att logiska värden behandlas som tal. Dessvärre fungerar hans exempel olika under FORTH-79 (eller föreningens FORTH) eller med den nya FORTH-83. Värdet för sant villkor är nämligen 1 i alla FORTH-dialekter, utom i FORTH-83, där värdet har ändrats till -1. Det är en utmärkt anledningen till att inte vara för smart.

Thinking FORTH avslutats med översikt av FORTH (för dem som inte har **Starting FORTH**), implementationen av två intressanta ord som presenteras i kapitel 7, implementationen av några andra användbara ord (t.ex. ASCII), svar på vissa övningsuppgifter samt en sammanfattning av stilreglerna i kapitel fem. Det finns även ett register.

Till sist några negativa synpunkter. Jag tycker att boken är intressant så länge Brodie skriver om det som han behärskar, nämligen FORTH. Däremot blir jag irriterad när han jämför FORTH med andra programspråk, vilka han uppenbarligen inte vet något alls om, utom att de är underlägsna FORTH.

Således behandlar första kapitlet programkonstruktion i allmänhet och inleds med en kort historik hur programspråken har utvecklats från maskinkod till skapelsens krona (den som gissar vad vinner en fotvandring till Sigtuna). För att riktigt visa hur bra FORTH är, skriver han om följande FORTH-

program:

```
: BREAKFAST
  HURRIED? IF CEREAL ELSE EGGS THEN CLEAN ;
```

till följande:

```
procedure make-breakfast
  var h: boolean (indicates hurried)
  test for hurried
  if h = true then
    call make-cereal
  else
    call make-eggs
  end
  call cleanup
end
```

Exemplet ser onekligen klumpigt ut i de här programspråket. Brodie passar på att påpeka att i FORTH behöver man bara skriva ett ords namn för att anropa det, i motsats till exemplet ovan där man måste skriva **call** vid varje anrop.

Nu är det inte så riktigt så illa med moderna programspråk. Endast i gamla programspråk som FORTRAN och PL/I finns det någon **call-sats**. Inget av de vanligaste moderna högnivåspråken, som C, Pascal, Modula-2 eller Ada har någon **call-sats**. Man behöver heller inte använda kryptiska namn. Låt oss titta på hur exemplet skrivs i Pascal (eftersom Brodies hemmagjorda språk mest liknar Pascal):

```
procedure breakfast;
begin
  if hurried then
    cereal
  else
    eggs;
  cleanup
end;
```

Jag förutsätter att Brodies "exempel" bygger på ren okunnighet och inte avsiktlig vilseledning.

Sammanfattning: Boken är en värdig uppföljare till **Starting FORTH**. Så länge Brodie håller sig till FORTH är boken mycket läsvärd. De jämförelser som görs med andra språk bör däremot tas med en stor nypa salt. Det gäller främst kapitel 1, 2, 3 och en liten del av kapitel 8.

Introduction to Assembly Language for the TI Home Computer, av Ralph Molesworth, Steve Davis Publishing, Dallas 1983, 139.

Jag vet inte om den här boken finns i Sverige; jag köpte mitt exemplar i San Francisco förra sommaren.

Boken vänder sig till den som har antingen Editor/Assembler eller Mini Memory. Jag tror inte att boken är lämplig för den direkte nybörjaren; däremot kan den lite mer avancerade 99-ägaren ha god nytta av alla programexempel som finns i boken.

Alla exempel är utskrivna med skrivare, varför det finns gott hopp om att de fungerar. Mängden kommentarer skiljer sig väsentligt: Vissa program har inga kommentarer alls, medan andra har för många kommentarer, av följande typ:

```
BLWP ÉDSRLNK      PERFORM DSRLNK
INC R3             ADD 1 TO R3
```

Något program har lite vettigare kommentarer.

Följande behandlas i boken: Binär och hexadecimal aritmetik, adresser, register, maskinspråks-instruktioner, visa tecken på skärmen, läsa av tangentbordet, filhantering, sortering, blanda BASIC och assembler. De flesta av de nämnda avsnitten innehåller programexempel, varav vissa tar upp några sidor. Dessutom finns två separata exempel som visar hur man definierar riktiga små bokstäver med under-slänger och ett BASIC-program med assemblersubrutiner som ritar stolpdiagram.

The Elements of Programming Style, av Brian W. Kernighan och P.J. Plauger, McGraw-Hill, New York 1978, 168 sidor.

The Elements of Programming Style är en ovanlig bok om strukturerad programmering. Författarna visar ett program, hämtat ur någon lärobok, och diskuterar vad som är bra och dåligt. (Läroböcker innehåller tyvärr alltför ofta exempel på hur man inte skall göra.) Sedan förbättrar de programmet och ger en allmän regel, som utan stöd av exemplet skulle verka abstrakt och svår att tillämpa i verkligheten.

Trots att alla exempel är skrivna i så gott som "döda" språk, FORTRAN och PL/1, är boken fortfarande mycket läsvärd. Lärdomarna kan nämligen tillämpas på andra språk, både ostrukturerade som BASIC och assembler, och strukturerade som Pascal och C. Man har ungefär samma möjligheter att skriva strukturerat i FORTRAN och Extended BASIC, varför jag tror att läsare av Programbiten kan ha nytta av denna bok.

Första kapitlet är introduktionen till boken. De följande tre kapitlen behandlar stil med början vid de enskilda satserna i språket, för att fortsätta med kontrollstruktur och till slut hur man delar sitt program i moduler (underprogram, subrutiner) på ett lämpligt sätt. Författarna strävar hela tiden efter att skriva tydlig kod, och de struntar i om det tar några bråkdelar av en sekund längre tid att exekvera programmet.

Därefter behandlas in- och utmatning. Författarna menar här att alla indata skall testas för otillåtna och orimliga värden. Kapitlet inleds med en tidningsartikel som berättar om hur en operatör av misstag matade in ett "P" i ett fält med siffror, med en förlust på 290000 dollar som resultat.

Nästa kapitel påvisar en rad vanliga misstag och fallor. Några exempel på sådana fallor är att glömma att ge variabler värden innan de används, loopar som går ett steg för långt eller för kort och flyttalsfallor (som att $10 * 0.1$ sällan blir exakt lika med 1).

Därefter behandlas effektivitet och optimering. Författarna föreslår att man skall få programmet att fungera innan man försöker få det snabbt. När det väl fungerar kanske man upptäcker att hastigheten är fullt tillfredsställande. Om programmet ändå är för långsamt brukar det sällan löna sig att "smartkoda" den använda lösningsmetoden. Det är bättre att hitta en bättre metod. Ett utmärkt exempel är sortering. "Bubble sort" är en enkel metod som fungerar bra för ett litet antal element. Med ett större antal element blir metoden mycket långsam, och det lönar sig inte att finputsa metoden; endast en bättre sorteringsmetod hjälper.

Det sista kapitlet tar upp dokumentation. Författarna menar att dåligt skriven kod inte blir bättre av mängder av kommentarer; skriv hellre om sådan kod. Bra skriven kod behöver inte så mycket kommentarer. Dessutom är kommentarer som säger samma sak som koden, eller felaktiga kommentarer, värdelösa eller värre än inga kommentarer alls.

Sist i boken finns en sammanfattning över alla regler som härletts i boken. Varje kapitel innehåller övningsuppgifter; ofta skall man lista ut vad ett program gör (den svåraste delen) och sedan förbättra det (det brukar vara lättare).

Sammanfattning: En bok som rekommenderas för den programmeringsintresserade. Det kan krävas viss ansträngning för att komma in i boken, men det är värt det. Det är för övrigt intressant att jämföra boken med Brodies Thinking FORTH, recenserad ovan.

INTRODUKTION TILL FORTRAN

För att göra det lättare att läsa The Elements of Programming Style förklaras här de största skillnaderna mellan FORTRAN och Extended BASIC.

```
C          SUMMABERÄKNING
          ISUM = 0
          DO 100 I = 1, 100, 2
          ISUM = ISUM + I
100        CONTINUE
```

motsvarar i BASIC

```
90 REM SUMMABERÄKNING
100 SUM = 0
110 FOR I = 1 TO 100 STEP 2
120 SUM = SUM + I
130 NEXT I
```

CONTINUE är en nollsats som inte gör någonting. Variabeln heter ISUM i stället för SUM, eftersom vi vill att den skall vara en heltalsvariabel. Alla odeklarerade variabler i FORTRAN får typen bestämd av första bokstaven: A-H, O-Z är flyttal (REAL) och I-N är heltal (INTEGER).

Villkor skrivs på följande sätt:

```
.LT.      mindre än
.LE.      mindre än eller lika med
.EQ.      lika med
.NE.      ej lika med
.GE.      större än eller lika med
.GT.      större än
.NOT.     inverterar ett villkor
```

Det finns flera sorters IF-satser:

IF (villkor) sats

Om villkoret är uppfyllt utförs satsen.

IF (C .LT. 0.0) C = -C

Om C är mindre än flyttalet 0, utförs C = -C (vilket alltså tar absolutbelopp).

Den aritmetiska IF-satsen

IF (uttryck) radnr1, radnr2, radnr3

hoppa till radnr1 om uttrycket är mindre 0, till radnr2 om uttrycket är lika med noll, eller till radnr3 om uttrycket är större än noll. Perfekt för att göra program helt oläsliga.

WRITE(enhet, radnr) variabellista
READ(enhet, radnr) variabellista

sköter utmatning respektive inmatning. enhet anger vart/varifrån man skriver/läser, medan radnr hänvisar till en FORMAT-sats som talar om hur överföringen skall göras.

SUBROUTINE name(parameterlista)

motsvaras av SUB i Extended BASIC.

FUNCTION name(parameterlista)

definierar ett underprogram som returnerar ett värde (tyvärr saknas motsvarighet i Extended BASIC).

DATA-satsen i FORTRAN används för att ge variabler ett initialvärde.

BREV

Från en medlem i Växjö fick vi en kassett - ett vanligt "postens ljudbrev" - med dels ett brev på, dels ett par program. Brevet följer nedan. Programmen finns på annan plats i tidningen under namnen TIPSRAD och TOTO-G./Red

Hej!

Som ni ser använder jag det nya ordbehandlingsprogrammet.

Tyvärr! Om jag skriver kommatecken så får jag inte ut texten efter det fast jag använder citations tecken vid varje rad.

Är det nå't fel i programmet P:TEXTUT eller har jag gjort något fel?

I P:TEXTUT-programmet saknas raderna med de svenska bokstäverna.

Jag skickar med två program i X-BASIC.

Det första är ett tipsprogram. Jag har matat in alla instruktioner med undantag för instruktioner till printer eftersom jag ingen har. Om ni vill publicera det vore det roligt. Jag har lämnat raderna 650 till 730 tomma till detta syfte att kunna få printerutskrifter.

Det andra programmet är till dem som tycker om att spela på hästar.

Om ni accepterar dessa program, vill jag gärna från banken ha(Red: Och det skall han få!)

Med vänliga hälsningar

Giuseppe Insalaco

Vintervägen 22 A

352 37 Växjö

Från Arne Wennberg i Landskrona (han som skrivit ett program TRIPPELPUNKTEN i X/BASIC) kom ett brev. Han har problem med programmen P:TEXTIN och P:TEXTUT. Det verkar faktiskt vara samma problem som Giuseppe anmält. Arne skriver (i sammandrag):

... Jag har undersökt detta och förstår inte varför. Om jag slår in

A\$="HEJ,KALLE" och sedan

PRINT A\$

skrivs det fullt korrekt HEJ,KALLE på skärmen. Om jag däremot spelar in strängen i en fil med definitionen OPEN\$1:"CS1", SEQUENTIAL, DISPLAY, OUTPUT, FIXED 128 och sedan spelar tillbaka detta med filen definierad exakt likadant med bara skillnaden att OUTPUT bytts mot INPUT, så blir det bara HEJ på skärmen när jag printar det inlästa. Det som står efter kommatecknet försvinner....

Men Arne har inte gett sig med detta utan klurat ut att han får korrekt resultat om han definierar om filen som INTERNAL i stället för DISPLAY. Han undrar nu om han har något fel på sina grejor - dator eller bandspelare (Philips Data Recorder D6628).

Jag tror nog att grejorna är OK och att det handlar om en systematisk egenskap hos operativsystem och bandspelar-rutiner i 99-an i kombination med egenskaper hos instruktionerna PRINT och INPUT för filer (se den svenska bruksanvisningen sidorna 121-131).Jag skall försöka få en av våra systematiskt-analytiska begåvningar med intresse och kunskaper som täcker det här att förklara saken i tidningen - helst förstås i det här numret./Red.

Tipsrad

av Giuseppe Insalaco

```

10 CALL CLEAR :: PRINT "OM DU VILL HA UTSKRIFT PÅ" :
: PRINT :: PRINT "PRINTER,MATA IN INSTRUKTION-" :
: PRINT
20 PRINT "ER TILL DEN I RADERNA" :: PRINT :: PRINT "
650 TILL 730."
30 FOR T=1 TO 1500 :: NEXT T
100 CALL CHAR(91,"00440038447C4444")
110 CALL CHAR(93,"00100038447C4444")
120 CALL CHAR(92,"0044003844444438")
130 CALL CLEAR
140 PRINT " $$$$$$$$$$$$$$$$$$$$$$$$" :: PRINT
150 PRINT " $ DATOR TIPPAR AT DIG $" :: PRINT
160 PRINT " $$$$$$$$$$$$$$$$$$$$$$$$"
170 PRINT :: PRINT
180 PRINT "DENNA PROGRAM BYGGER EN" :: PRINT
190 PRINT "EN SYSTEM AT DIG DU BESTÄM-" :: PRINT
200 PRINT "MER DEN LÄNGSIKTIGA FÖRDELN-" :: PRINT ::
PRINT "NING MELLAN 1,X OCH 2." :: PRINT
210 PRINT "VILL DU HA 40% '1' SKRIV .40"
220 PRINT :: PRINT "VILL DU HA SAMMA ELLER OLIKA" ::
PRINT :: PRINT "SANNOLIKHET FÖR MATCHERNA?" :: PR
INT :: PRINT "(S/O)",:: INPUT B$
230 IF B$="O" OR B$="o" THEN GOTO 760
240 DIM E(13),F(13)
250 PRINT
260 PRINT "ANGE '1'(%)" ";;: INPUT N :: PRINT
270 FOR X=1 TO 13 :: E(X)=N :: NEXT X :: X=0
280 PRINT "ANGE 'X'(%)" ";;: INPUT M :: PRINT
290 FOR X=1 TO 13 :: F(X)=M :: NEXT X :: X=0
300 PRINT :: PRINT "ANGE ANTAL RADER (MAX 145)" :: PR
INT :: INPUT G :: PRINT
310 PRINT "UTSKRIFT PÅ PRINTER (J/N)"::: INPUT A$
320 CALL CLEAR :: CALL SCREEN(16)
330 DISPLAY AT(2,4):"TIPSFÖRSLAG" :: DISPLAY AT(2,16)
:G : DISPLAY AT(2,21):"RADER"
340 IF B$="O" THEN 370
350 DISPLAY AT(4,2):"1=" :: DISPLAY AT(4,4):E(1):: DI
SPLAY AT(4,10):"X=" :: DISPLAY AT(4,12):F(1):: DI
SPLAY AT(4,20):"2="
360 DISPLAY AT(4,22):1-E(1)-F(1)
370 IF A$="J" THEN 420 ELSE 380
380 IF J=9 OR J=18 OR J=27 OR J=36 OR J=45 OR J=54 OR
J=63 THEN GOSUB 740
390 IF J=72 OR J=81 OR J=90 OR J=99 OR J=108 OR J=117
OR J=126 OR J=135 THEN GOSUB 740
400 GOTO 450
410 PRINT :: PRINT
420 IF J=9 OR J=18 OR J=27 OR J=36 OR J=45 OR J=54 OR
J=63 THEN GOSUB 650
430 IF J=72 OR J=81 OR J=90 OR J=99 OR J=108 OR J=117
OR J=126 OR J=135 THEN GOSUB 650
440 IF (A$="J")AND J=G THEN GOSUB 650
450 IF J=G THEN 820
460 J=J+1
470 B=B+1
480 X=0
490 C=0
500 RANDOMIZE
510 IF B=4 OR B=7 OR B=10 THEN D=D+2
520 FOR X=1 TO 13
530 T=RND
540 IF X=4 OR X=7 OR X=10 THEN C=C+1
550 IF T<E(X)THEN 600
560 IF T<E(X)+F(X)THEN 620
570 DISPLAY AT(6+X+C,(2*B)+D):"2"
580 NEXT X
590 GOTO 640
600 DISPLAY AT(6+X+C,(2*B)+D):"1"
610 GOTO 580
620 DISPLAY AT(6+X+C,(2*B)+D):"X"
630 GOTO 580
640 GOTO 370
650 ! I DE FÖLJANDE RADER KAN DU PROGRAMMERA INSTRUK
TIONER TILL PRINTER
660 !
670 !
680 !
690 !
700 !
710 !
720 !
730 !

```

```

740 DISPLAY AT(24,1)BEEP:"PRESS ANY KEY TO CONT."
750 CALL KEY(0,K,S):: IF S=0 THEN 750 :: B=0 :: D=0 :
: RETURN
760 DIM E$(13),F$(13):: FOR X=1 TO 13
770 CALL CLEAR :: PRINT "ANGE P('1') MATCH N.":X:: I
NPUT E(X):: PRINT
780 PRINT :: PRINT "ANGE P('X') MATCH N.":X:: INPUT
F(X)
790 NEXT X
800 X=0
810 GOTO 300
820 DISPLAY AT(2,2):" "J :: DISPLAY AT(2,10):"RADE
RS FORSLAG"
830 CALL KEY(0,K,S):: IF S=0 THEN 830 :: END

```

TOTO-G

av Guiseppe Insalaco

```

100 REM EXTENDED BASIC
110 REM GIUSEPPE INSALACO VINTERVÄGEN 22 A
352 37 VÄXJÖ
120 CALL CHAR(91,"00280038447C4444")
130 CALL CHAR(92,"0028003844444438")
140 CALL CHAR(93,"00100038447C4444")
150 DIM R(5):: DIM C(5):: !
160 FOR X=1 TO 5 :: READ R(X):: NEXT X
170 DATA 4,6,8,10,12
180 CALL CHAR(95,"040EFF7C7C644224")
190 CALL CHAR(94,"000000FF10101010")
200 CALL CLEAR
210 P=1000 :: !SPELARENS PENGAR
220 DIM H$(5)
230 FOR X=1 TO 5 :: READ H$(X):: NEXT X
240 DATA "NORTH-STAR","RIBBOT","BLACK-ARROW","DAKOTA-
PRIDE","BLUE-RIBBON"
250 GOSUB 340 :: ! TITEL
260 GOSUB 540 :: ! VADSLAGNING
270 GOSUB 710 :: ! RITA BANAN
280 GOSUB 770 :: ! LOPNING
290 GOSUB 1020 :: ! KOLLA RESULTAT
300 IF P=0 THEN GOTO 330
310 PRINT "VILL DU SPELA MER(J/N)>:: INPUT R$
320 IF R$="J" THEN CALL CLEAR :: GOTO 260
330 GOTO 1260 :: ! SLUT
340 ! TITEL
350 CALL CLEAR
360 GOSUB 390 :: ! BLINKANDE TITEL
370 GOSUB 450 :: !FÖRKLARING
380 RETURN
390 ! TITEL
400 CALL SCREEN(16)
410 DISPLAY AT(12,11):"TOTO-GALOP"
420 FOR I=1 TO 1500 :: NEXT I
430 CALL CLEAR
440 RETURN
450 ! REGLER
460 DISPLAY AT(4,3):"VÄLKOMMEN TILL TOTO-GALOPP"
470 PRINT
480 DISPLAY AT(7,3):"DU VILL PROVA LYCKAN." :: DISPLA
Y AT(9,3):"DU HAR MED DIG 1000 KR"
490 PRINT
500 DISPLAY AT(10,3):"ATT SPELA MED." :: DISPLAY AT(1
3,3):"SATSA PÅ EN HÄST I VARJE" :: DISPLAY AT(15,
3):"LOPP."
510 DISPLAY AT(22,3):"LYCKA TILL"
520 FOR D=1 TO 3000 :: NEXT D
530 CALL CLEAR :: RETURN
540 ! VADSLAGNING
550 ! SKRIV HÄSTARNAS NAMN
560 FOR X=1 TO 5
570 PRINT :: PRINT X,H$(X)
580 NEXT X
590 FOR X=1 TO 28 :: PRINT "-":X:: NEXT X
600 PRINT
610 PRINT "ODDS ÄR 5 TILL 1" :: PRINT "-----"
620 PRINT
630 PRINT "VILKEN HÄST SATSAR DU PÅ" :: PRINT "(SKRIV
HÄSTENS NUMMER)" :: INPUT H
640 PRINT

```

```

650 PRINT "DU HAR KR:";P
660 PRINT "HUR MYCKET SATSAR DU?" :: INPUT S
670 IF S>P THEN GOTO 650
680 IF S>P THEN GOTO 32767
690 LET P=P-S
700 CALL CLEAR :: RETURN
710 CALL CLEAR
720 CALL SCREEN(3)
730 FOR X=1 TO 29 :: CALL HCHAR(3,X,94):: NEXT X
740 FOR X=1 TO 30 :: CALL HCHAR(13,X,94):: NEXT X
750 FOR F=3 TO 13 :: CALL VCHAR(F,30,58):: NEXT F
760 RETURN
770 RANDOMIZE
780 W=0 :: !
790 !
800 FOR X=1 TO 5 :: C(X)=0 :: NEXT X
810 !
820 FOR X=1 TO 5
830 DISPLAY AT(R(X),C(X))BEEP:CHR$(95)
840 NEXT X
850 FOR D=1 TO 50 :: NEXT D
860 FOR X=1 TO 5
870 DISPLAY AT(R(X),C(X)):CHR$(32)
880 NEXT X
890 FOR X=1 TO 5
900 C(X)=C(X)+1+INT(RND*3)
910 IF C(X)>27 THEN C(X)=28 :: CALL SCREEN(11):: W=1
920 NEXT X
930 IF W=1 THEN GOSUB 950 :: RETURN
940 GOTO 810
950 ! VINST
960 FOR X=1 TO 5
970 DISPLAY AT(R(X),C(X)):CHR$(95)
980 NEXT X
990 FOR D=1 TO 500 :: NEXT D
1000 CALL CLEAR
1010 RETURN
1020 !
1030 !
1040 W=0
1050 FOR X=1 TO 5
1060 IF C(X)=28 THEN IF H=X THEN LET W=1
1070 NEXT X
1080 IF W=1 THEN GOSUB 1120
1090 IF W=0 THEN GOSUB 1200
1100 P=P+S
1110 RETURN
1120 ! OM VINST
1130 S=S*5
1140 DISPLAY AT(8,10):"GRATTIS !!!" :: DISPLAY AT(9,10
):"-----"
1150 DISPLAY AT(14,3):H$(H):" VANN LOPPET"
1160 DISPLAY AT(18,3):"DU HAR VUNNIT KR:";S
1170 FOR D=1 TO 1000 :: NEXT D
1180 CALL CLEAR :: CALL SCREEN(16)
1190 RETURN
1200 ! FÖRLUST
1210 S=0
1220 DISPLAY AT(9,13)ERASE ALL BEEP:"TYVÄR"
1230 DISPLAY AT(15,3):H$(H):" VANN INTE"
1240 FOR D=1 TO 1000 :: NEXT D
1250 CALL CLEAR :: CALL SCREEN(16):: RETURN
1260 ! SLUT
1270 IF P=0 THEN GOTO 1310
1280 IF P<1000 THEN GOTO 1350
1290 IF P=1000 THEN GOTO 1440
1300 IF P>1000 THEN GOTO 1390
1310 ! PANK
1320 DISPLAY AT(6,10)ERASE ALL BEEP:"AJ AJ AJ."
1330 DISPLAY AT(10,3):"DU ÄR ALLDELES PANK." :: DISPLA
Y AT(14,3):"DU FÄR LOV OCH LIFTA HEM"
1340 FOR D=1 TO 1000 :: NEXT D :: END
1350 ! FÖRLUST
1360 DISPLAY AT(6,10)ERASE ALL BEEP:"AJ AJ"
1370 DISPLAY AT(10,3):"DIN KASSA ÄR NU:";P;" KR."
1380 FOR D=1 TO 1000 :: NEXT D :: END
1390 ! SPELAREN VANN
1400 DISPLAY AT(6,10)ERASE ALL BEEP:"FINT !!!"
1410 DISPLAY AT(10,3):"KLOKT ATT SLUTA MEDAN DU" :: DI
SPLAY AT(12,3):"ÄR I VINST"
1420 DISPLAY AT(16,3):"NU GÄR VI OCH FESTAR" :: DISPLA
Y AT(20,3):"DU HAR";P;"KRONOR"
1430 FOR D=1 TO 1500 :: NEXT D :: END
1440 ! JÄMT
1450 DISPLAY AT(6,10)ERASE ALL BEEP:"BRA !"
1460 DISPLAY AT(12,3):"DET GICK JÄMT UT."
1470 FOR D=1 TO 1500 :: NEXT D :: END

```

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 mediet, dels en kopieringsavgift för varje program.

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

Som Föret får Du tre program i utbyte när Du skickar in ett program DU SJÄLV HAR GJORT!
Ange om Ditt program är i TI-basic, X-basic eller i FORTH. Beskriv vilken utrustning som behövs enligt vidstående kodförklaring.

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

39 ** ALLMAN MATEMATIK
01391001E --- Integral, derivata, andragradsekv.
reella rötter, cp analys, definier
era chars, 3-D plot, fakultet och
primtalstest
09391003H --- Differential eq.
78 ** ASTRONOMI
01781001E --- Beräkning av geostationära sate-
litters positioner.
14 ** EKONOMI
07141001H ---X Annuities
65 ** ELEKTRONIK
01651001E --- Beräkning av komponentvärden för
resistiv parallellkoppling, kond-
ensator i serie, resonans för
spole och kondensator, omvandling
frekvens - våglängd, ohm's lag,
uträkning av antennlängd
30 ** LINJÄR ALGEBRA
06301001E --- Matris invertering/multiplikation
96 ** MUSIK
08961001E ---X Killing me softly
05962001 ---C A Strauss waltz
05962002 --- Never on a sunday
05962003 ---X Berceuse
05962004 ---X Beethoven # 5
05962005 ---X Serenade
05962006 ---X Amazing graze
05962007 --- Beethoven opus 27
05962008 ---X Time in a bottle
05962009 ---X Light up my life
05962010 ---N Bumble-boogie
06962011 ---D Fiddler on the roof
07962013 --- Looking through you
08962014 ---X Chopin op 11 # 3
06972001 --- The pink panther
06972006 --- Let me call you sweetheart

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 --- Backgammon
03911017E ---X 3D Tic-tac-toe
03911018E ---X Not one
03911019E ---X Datorpoker
03911024E --- Fyra i rad (luffarschack)
04911025E --- Startrek 2
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

KODFÖRKLARING

De fyra första siffrorna är interna beteckningar.
Femte siffran anger utspringsland:
1 = England, USA 2 = Holland
3 = Sverige
De tre sista siffrorna är ett löpnummer.
Bokstaven som finns sist anger vilket språk som används
i programmet:
E = Engelska F = Franska
H = Holländska S = Svenska
Eventuella bokstäver efter numret talar om vilken
utrustning som krävs:
C = CALL FILES (1) måste användas om man har disk.
D = Diskettenhet E = Minnesexpansion
F = FORTH J = Joystick
P = Printer X = Extended Basic
Ett + anger att programmet är delat i flera delar.

90 ** PRAKTISKA PROGRAMHJÄLPMEDEL
01901001E ---P Word-processor I
01901002E --- Adressregister
06901003E ---X Sortering och utskrift från flera
disketter av upp till 300 program
06901004E --- Diskkatalog med utskrift
07901005H --- Morse
08901006E ---X Utskrift av titelsida för ex.
listningar
10901007H --- Lotto
10901008E --- Very large characters
10901010E ---C Ordbehandling för TP-printer.
11901011S --- Stapeldiagram
07902002 ---P Banner, stor text utskriven på
printer
07971003H --- Demonstration av olika tecken-
storlekar
20 ** REGRESSION, KURVANPASSNING
01201001E --- Uträkning av en linjes ekvation
efter ett antal punkter fördelade
kring en linje
01201002E --- Uträkning av riktningskoeff.
29 ** STATISTIK, SANNOLIKHET
09291001H --- Statistiska kombinationer etc.
21 ** STATISTIK, VARIANS
06211001E ---C Beräkning av standardavvikelser
92 ** UTBILDNING
04921001E --- Enkla räknesev. med de fyra
räknesev.
04921002E --- Kemifrågor
04921003E ---X Relative IQ-test
04921004E --- Algebra
04921005E --- Projectile problems, beräkn.-spel
01921006H ---X Arvsanlag
07921007E --- Träning i dätid (imperfekt)
10921008E --- Räkneträning med bråkdelar
99 ** ÖVRIGT
01991001E ---X Släktforskningsregister
01991002H --- Utskrift av månadskalender
09991003F --- Biorythm.
01992001 --- Månkalender

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 ---X Treasure hunt, labyrintspel.
10911061H ---X Kermit
10911062H ---X Find the gun
10911063E --- Matematikspel.
10911065E --- Camel, adventureritt i öken
10911008E --- Colprfraktions, matematikspel
06912004E --- Sialom
06912005E --- Killer
09912008F ---X Asteriod (tal).
09912009 --- Battlestar
11911055S ---X Hängning
11917001S ---X Inkräktarna
11917002S ---X Pärön