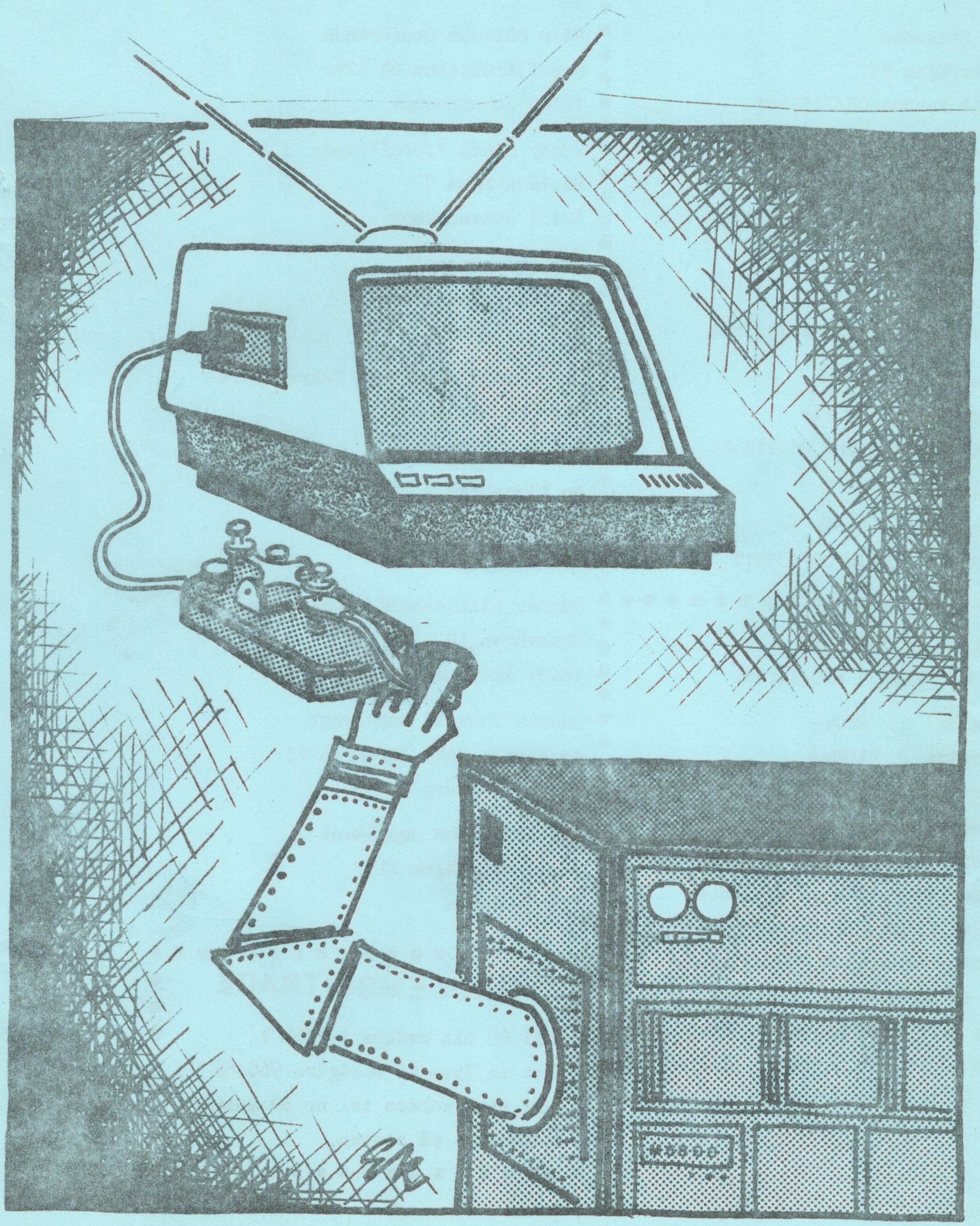
MPU-laren nr 3-1979

Tidskrift för Privatdataklubben PDS · Årgång 2



Redaktionen

MPU-larenredaktionen består av
följande medlemmar.

Alf Eriksson
Vallavägen 83
13641 Handen 08/7452667

Roger Holmstrand
Rapsodivägen 128
14200 Trångsund 08/7713852

Carl Axel Bruno
Timotejvägen 116
19177 Sollentuna 08/7543788

Lars Landqvist
Skagersvägen 20 1tr
12142 Johanneshov 08/912625

Bernt Svensson
Ploggatan 16 3tr
11634 Stockholm 08/403517

Annonser

ANNONSPRISER I MPU-laren.

Helsida 200:-
Halvsida 100:-
Kvartsida 50:-

Priserna gäller tryckfärdigt material.Radannonser för medlemmar är gratis om det inte gäller firma.Tidningens upplaga är för närvarande 200 ex.
För material till tidningen tala gärna med någon i redaktionen ovan eller sänd detta till,PD68 Box 98,12221 Enskede 1.

Styrelsen

PD68 nuvarande styrelse.

Nils Ohlsson Ordförande
Sandfjärdsgatan 68 12tr
12169 Johanneshov

Tommy Bladh V.Ordförande
Båstadsvägen 7
12151 Johanneshov

Åke Holm Kassör
Långsjövägen 15B
13554 Tyresö

Lars Olof Fagerström Sekreterare
Gärdebyplan 6 4tr
16374 Spånga

Bo Ljungblad V.Sekreterare
Brännkyrkagatan 117
11728 Stockholm

Gunnar Lilliesköld Suppleant
Mossvägen 1C
18499 Åkersberga

Anders Thunell Suppleant
co.Fittja R&TV Forvägen 15
14551 Norsborg

Gunnar Lovius Suppleant
Ragnebergsvägen 21
13667 Handen

MEDLEMSINTRÄDE

Vill DU bli medlem i PD68 ?
Sätt in 75:- på postgiro 960468-7
samt namn adress tel nr så kommer medlemskap på posten.

ORDFÖRANDE HAR ORDET

Fortfarande är mikrodatorn (och datorn), något som man i icke fackkretsar betraktar som skrämmande och oftast ser man endast nackdelar med den. Diskussionerna börjar dock bli mer nyanserade och i exempelvis Dagens Nyheter av tisdagen den 15 maj, kunde man under Ekonomi-sidan finna en stor artikel om mikrodatorn. Grosmolnen fanns visserligen med redan i rubriken, citat..
" En så löftesrik och skrämmande ....." , gav antydningar om att man åter tänkte servera en framtidsversion a´la Orwells 1984. Detta kommer dock delvis på skam och man går istället in för att på ett icke tekniskt sätt försöka förklara de möjligheter som finns och den revolution som kiselbiten innebär. Man nämner slutligen hjärtlungmaskinen som kontrast till kryssningsrobotar för att klargöra bredden på användningsområdet, och det är det längsta man kommer i den negativa kritiken. Exemplet är dock illa valt. Stål används i allt från skördemaskiner till raketspetsar. Ändå är man inte rädd för att utveckla och förbättra genom legeringar, som möjliggör konstruktion av nya raketer. Trots allt är säkert de flesta inte oroade över den möjlighet till ökade militära resurser som kan ske, dessa är ändå så överväldigande. Nej, man är istället rädd för att den betydligt närmare liggande vardagen skall vändas upp och ner. Att arbeten skall förändras till det sämre eller i sämsta fall försvinna. Man glömmer att de ökade resurser som datorn skapar bland annat används till att utöka redan befintliga rutiner, för att kunna utföra ett mera effektivt arbete genom exempelvis olika prognoser. De personella resurserna är oftast oförändrade, men med omdisponerade funktioner. Givetvis förekommer nya arbetsmoment som kan vara nog så enahanda. Men datorns natur gör den trots allt mest lämpad för de verkligt likartade uppgifterna, så totalt kommer säkert dessa att minska. Detta är inget nytt under stjärnorna, även om dagens samhälle förändras fortare. Vi har fortfarande bara foten över tröskeln, men vi snubblar framåt...

N.O.



6800 BASIC

Årets glada nyhet för 6800 användare, som nu får en BASIC helt i klass med Microsofts senaste 8K BASIC.

6800 BASIC är mycket snabb, så snabb att man på TSC misstänker att det kanske aldrig kommer en snabbare interpretator BASIC.
Fastän det är en interpreterande BASIC så är den nästan bubbelt så snabb, som existerande interpretatorer och komplilatorer för mikrodatorer, oavsett fabrikat, körda med samma klockfrekvens.

Alla standardinstruktioner ingår.
Snabb flyttalsaritmetik med 6 visade siffror och 7 siffror i beräkningarna.
Programraderna kan ha upp till 127 tecken med kolon och snedstreck som satsavskiljare.
IF...THEN...ELSE kontruktion för översiktiga program.
PEEK och POKE kan ske i hex-kod.
Obegränsat antal strängvariabler,subrutinanrop och nivåer på programslingor.
Antal tecken för en strängvariabel begränsas bara av tillgänligt minne, och behöver inte specificeras före användandet och kan dessutom ändras när som helst i programmet.
Variabelnamn kan bestå av 2 bokstäver.

Programlängd tape ca 9.5K disk ca 12K
Användarmanual medföljer, ingen listning.

MNEMONIC ASSEMBLER

Tillsammans med Text editor ger mnemonic assembler ett komplett programutvecklingssystem.
Assemblern är en av de mest mångsidiga som finns på mikrodatormarknaden, och kan enkelt anpassas till de flesta system.

Genom en ytterst effektiv LABEL hantering är Mnemonic ass. många gånger snabbare än andra residenta assemblers.

Alla standardmnemonics ingår.
LABELS upp till 6 signifikanta tecken.
Uttryck kan innehålla värden i decimalform,oktal, hex eller ASCII-kod, samt de fyra räknesätten.
Alla standarddirektiv och "pseudo-ops" ingår, samt direktiven SPC,PAG,TTL och OPT.
OPT direktivet erbjuder följande optioner:
utskrift av lista,uppdelning i sidor, objektkod (maskinkod) i minnet eller på yttre medium (band,remsa), alfabetiskt sorterad symboltabell.
Flexversionen tillåter decimala radnummer och formatterar automatiskt rader i olika fält(label, op-kod,operand(er)).

Manualen beskriver fullständigt inkoppling och användning av assemblern. Vidare ingår en rikligt kommenterad listning av programmet (källkod).

Programlängd ca 5K ord.

Swedish Electronics hb
Box 2065, 750 02 UPPSALA, Tel. 018-14 14 00

•••••••••• 6800 DEBUGGER ••••••••••••••••••••

Ett ytterst kraftfullt och helt komplett verktyg för felsökning i assemblerprogram. Debuggern kan simulera 6800:s alla funktioner inklusive avbrott och I/O. Till en mycket låg kostnad kan man med programmet ersätta en logikanalysator.

Obegränsat antal brytpunkter i RAM och ROM.
Varje brytpunkt ger upphov till en eller flera av 8 olika åtgärder. Åtgärderna kan vara villkorsberoende, och t.ex utföras vid visst minnes eller registerinnehåll. En " varvräknare", som ökar varje gång en brytpunkt passeras, kan begränsa eller fördröja de olika åtgärderna.
Uppgift om hur ofta brytpunkten passerats kan användas för statistiska ändamål eller för att allmänt studera programkörningens förlopp.
Eftersom simuleringen kontrolleras fullständigt kan "Trace Mode" kopplas in när som helst.
I "Trace Mode" visas registerinnehållet och op-kodens mnemonic efter utfärd instruktion eller, om man så vill, grupp av instruktioner.
Exekveringshastigheten kan kontrolleras och programkörningen kan stoppas när som helst.
Med" Trace Back" kan de 256 senast utförda instruktionerna visas.
Minnesskydd finns för skrivning, körning och simulering.
Fällor "Traps" kan sättas för att avbryta programmet när något allmänt villkor uppfylls, t.ex. för en avbrottinstruktion,vid ett visst antal subrutinnivåer eller efter ett visst antal instruktioner.
En rad i tagen kan assembleras och kod kan disassembleras.
Beräkningar kan utföras hexadecimalt.
Minne och register kan ändras och undersökas.

Programlängd 9K från hexadress 3000.
Användarmanual med kommenterad programlista medföljer.

Swedish Electronics hb

BOX 2065 75002 UPPSALA Tel: 018/100190

Repr.i Stockholm SA Teknik ,, 08/197080

Göteborg Datatrans ,, 031/180251

Linköping CN Electronic 013/96211

Medlemsregister

Här är namn och adress på alla som den 8/6 1979 har erlagt avgift för 1979. Ingen har så vitt vi vet bett att få avstå kontakt från likasinnade. Om du därför glatt överraskad upptäcker att en PD68 medlem bor i din närhet tycker vi att du kan skriva en rad till honom. Det tas säkert inte illa upp.

Tittar man på postnumren ser vi en viss högre medlemstäthet på vissa orter. Om någon känner för att ordna något på lokal nivå hjälper vi gärna till. Vi publicerar namn på kontaktmän om vi får reda på det. Hör av dig om du är intresserad.

Stämmer ditt namn och adress i registret? Det är samma information som trycks på etiketten som vi sätter på kuverten. Vill du ändra något eller tillägga data så har posten utmärkta kort där man kan fylla i både det ena och det andra. Adressändringskort, med andra ord. De behövs dessutom inte frankeras.

TB1

Denna sida innehöll medlemsregister.
Ej inskannat pga GDPR

Denna sida innehöll medlemsregister.

Ej inskannat pga GDPR

Denna sida innehöll medlemsregister.

Ej inskannat pga GDPR

MJUKVARUNYTT

WADUZITDO How to write a languae in 256 words or less.	Sep 78 Byte
A Memory pattern sensitivity test.	Okt 78 Byte
Testing Memory In BASIC	Okt 78 Byte
Tic - Tac --Toe game	Dec 78 Byte
Fast Fourier for the 6800.	Feb 79 Byte
Filling 6800 Op Code Holes	Mars79 Byte
GP Monitor for M6800 4K,8K,16K,32K Configuration	Okt 78 Interface
M6800 Data or program Adress Locator	Dec 78 Interface
JACPOT in Basic	Aug 78 Byte
6800 Disassembler	maj 79 Byte
Swtpc 6800 Display Routine	maj 79 Byte
6800 Register Display	maj 79 Byte
Tic-Tac-Toe: A Programming Exercise in Basic	maj 79 Byte

Vi söker pengar.....

Vår ordförande skrev i förra numret om "Mikrodator på Tekniska Museet", något som nämndes vid årsmötet och som därför inte var känt av andra än mötesdeltagarna. Därför var också intresset minst sagt svalt...

Nu gör vi ett nytt försök, kanske det sista av det här slaget. FRN (Forskningsrådsnämnden, ett organ för samordning av viss forskning under utbildningsdepartementet) utlyste i våras 900 000 kronor för att folk, organisationer, institutioner m fl skulle intressera sig för att informera breda grupper i samhället, t ex allmänheten, gymnasie- och grundskolestuderande m fl,om mikrodatorer och deras användning och konsekvenser.

Ansökningstiden gick ut 25 april utan att PD 68 skickade in något. Nu kommer det med största sannolikhet att bli en ny ansökningsomgång i höst, med ansökningstid i oktober. Vill klubben visa upp sig och samtidigt ge vanligt folk ett handtag när det gäller att få känna på vad en dator egentligen är, så finns chansen nu. Denna typ av projekt var det mycket magert med i förra omgången.

Vi skulle t ex kunna höra med Tekniska Museet om vi kunde få disponera en hörna och delta i deras stora mikrodatorutställning, som löper under oktober-juni. Vi kunde vara det praktiska inslaget och bygga upp ett klubbsystem som t ex - som det föreslogs vid senaste mötet - styr ett växthus. Andra idéer kan naturligtvis också vara tänkbara, det viktiga är att några personer intresserar sig så att vi kan gå in med en ansökan om pengar till detta.

Ar du intresserad av att pyssla med lite lärorika saker utan att det kostar dig något? Hör då av dig till Lars Olof Fagerström, tel: 08-787 8191 (dagtid) eller 761 61 90 (hem). Ring gärna med en gäng!

!!OBS!!

Här vi ännu en gång påminna alla medlemmar om att redaktionen välkomnar alla bidrag till tidningen. Mjukvara,hårdvara.teckningar,tips, frågor av allmänt intresse m.

Nästa Nummer

En liten bit..... del 2.

Nybörjarkurs i Basik.

Kassettinterface 300 & 1200 baud.

Spelhörnan.Spelprogram i Basik.

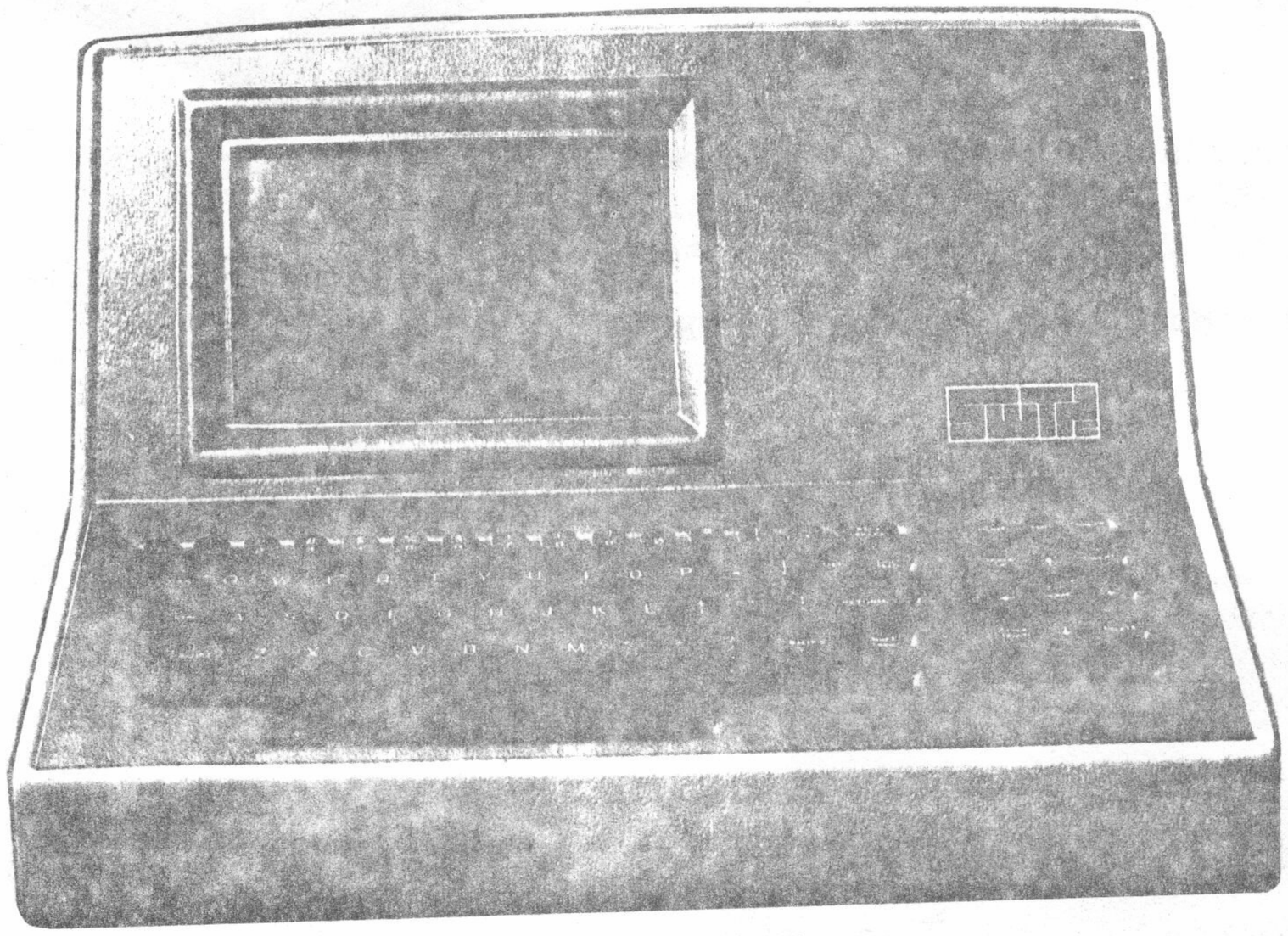
Radannonser

CRT-kort
säljes, SWTPC-buss, bra dokumentation. Pris 125:- kr. Ring
Lars Olof Fagerström,
Tel: 08-787 8191 (dagtid),
08-761 61 90 (kvällstid).

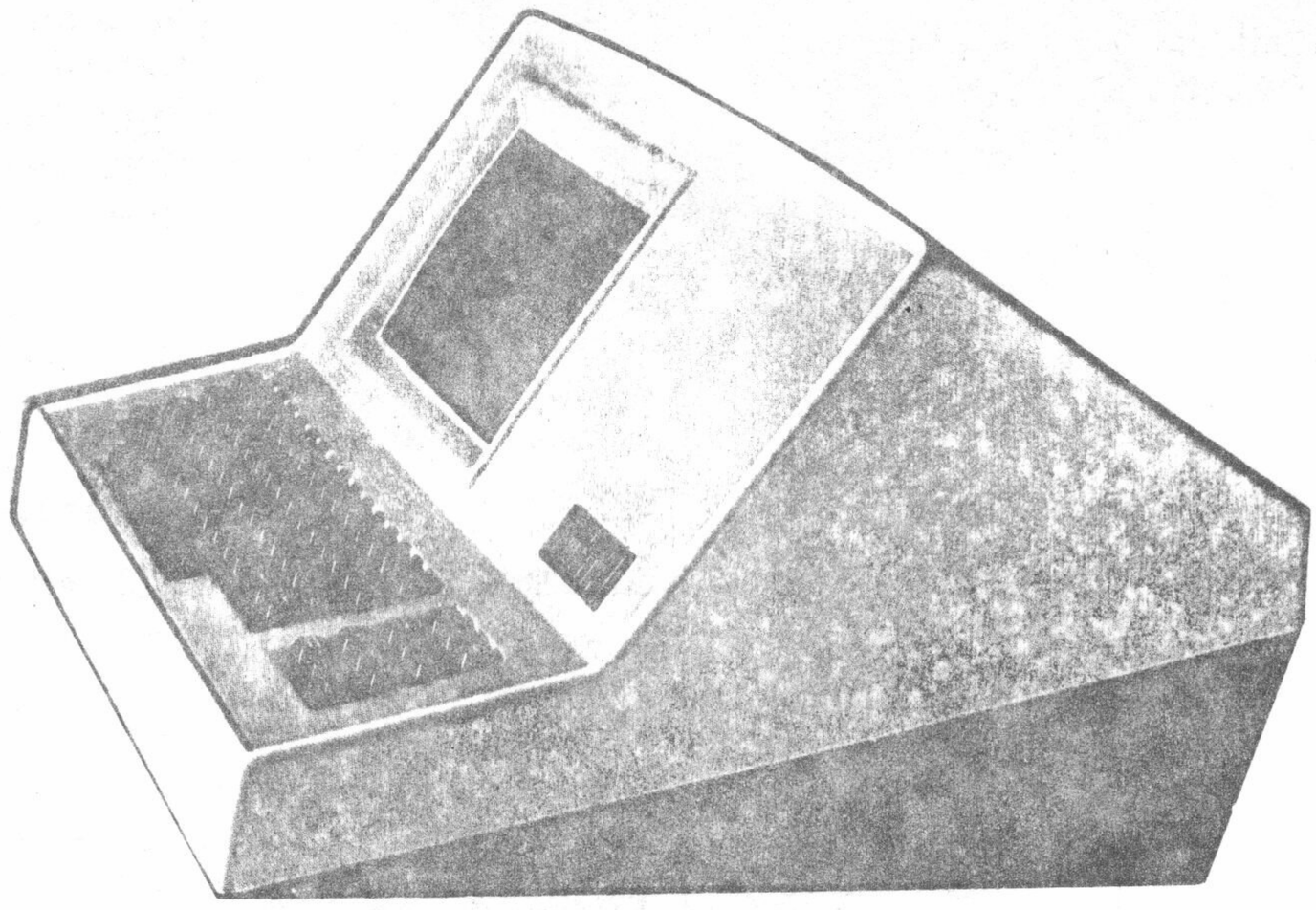
1 st. 40 kolumn. printer,
lika SWTPC PR-40 1000:-
Lars Landqvistt
Ringersvägen 20 1trr
12142 Johanneshov
Tel. Hem 08/912625
Tel. Arb 08/990657

CRT controller enl RT nr 10
Färdigbyggt 1000:- Epromkort
Färdigbyggt utan 2708 450:-
Skrivhjul Kassettdäck OLIVETTI
Mag 300:- A.Eriksson 08/7452667

CT-82 The Terminal of the Future



Pris:6200:~

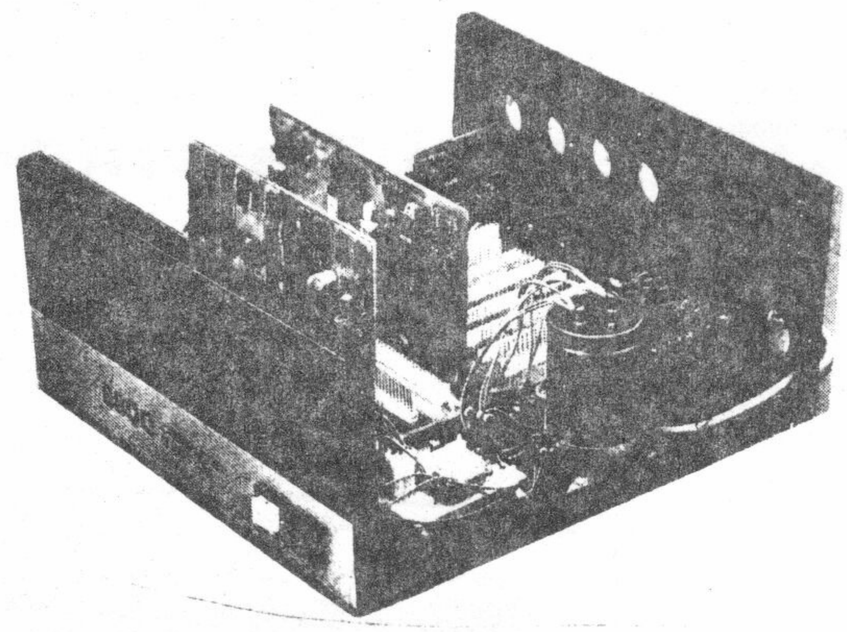


WEMOB AB

Box 72 · S-133 01 Saltsjöbaden · Sweden Telefon 08/7176288

MP-68 COMPUTER SYSTEM





MP-68: Grunddatorn består av låda med nätdel, moderkort och alla expansionskontakter. 4K statiskt RAM på 8K's kort, SWTBUG-ROM, SWTBUG RAM och serie interface.

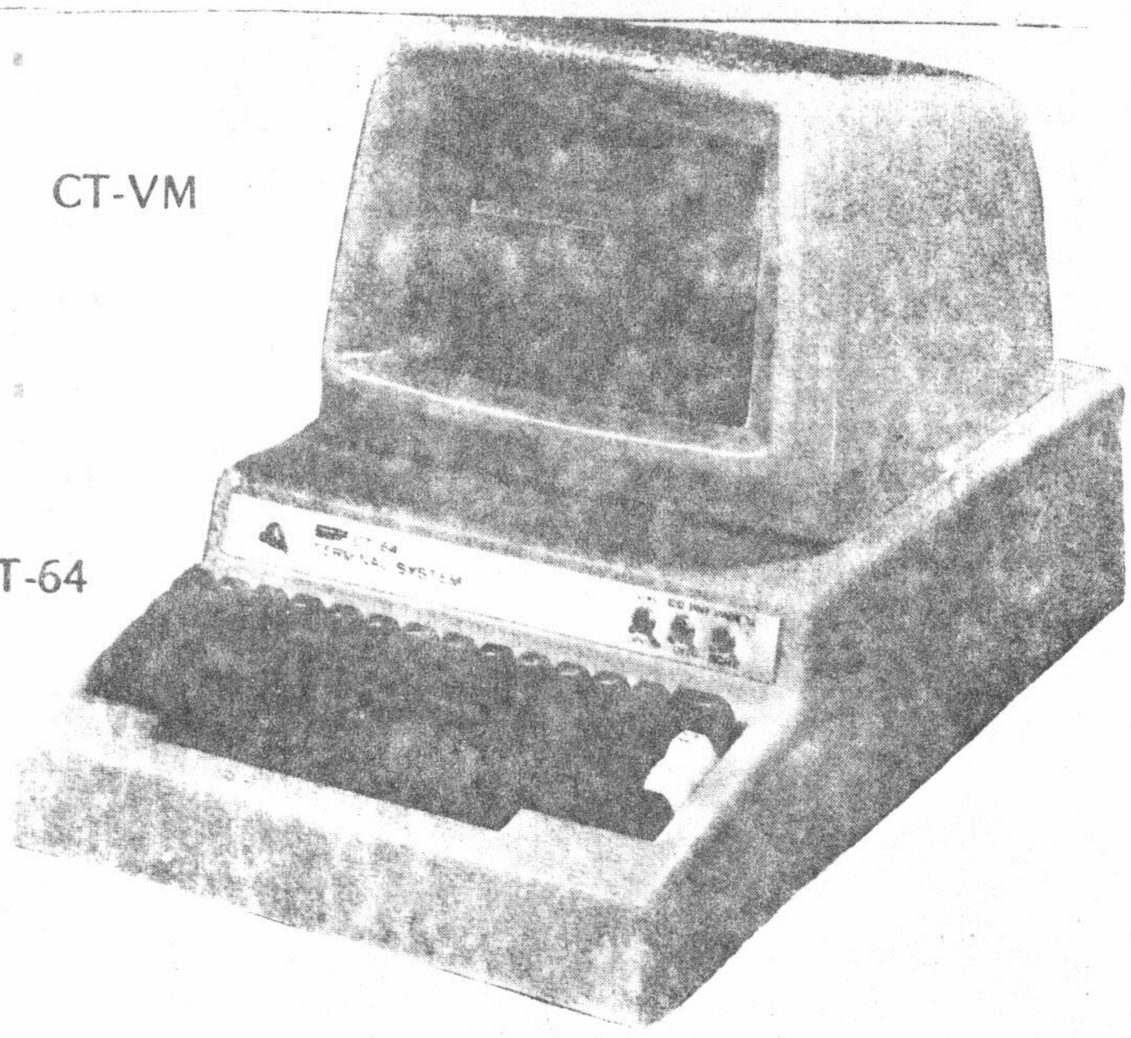
PRIS: 3.165:-

DESSUTOM FINNS MÅNGA TILLBEHÖRSKORT.

VISUAL DISPLAY TERMINAL

CT-VM

CT-64



* 16 Lines per page
* 64 or 32 Characters per line
* Upper and lower case letters
* Full 8 Bit Memory
* 128 Character Ascii Set
* 110/220 Volt 50-60 Hz Power Supply
* Scrolling or Page Mode Operation
* Control Characters Decoding – 32 Combination
* Prints Control Characters
* Usable with *any* 8 Bit Ascii Computer
* Reversed Background – HIGHLIGHTING

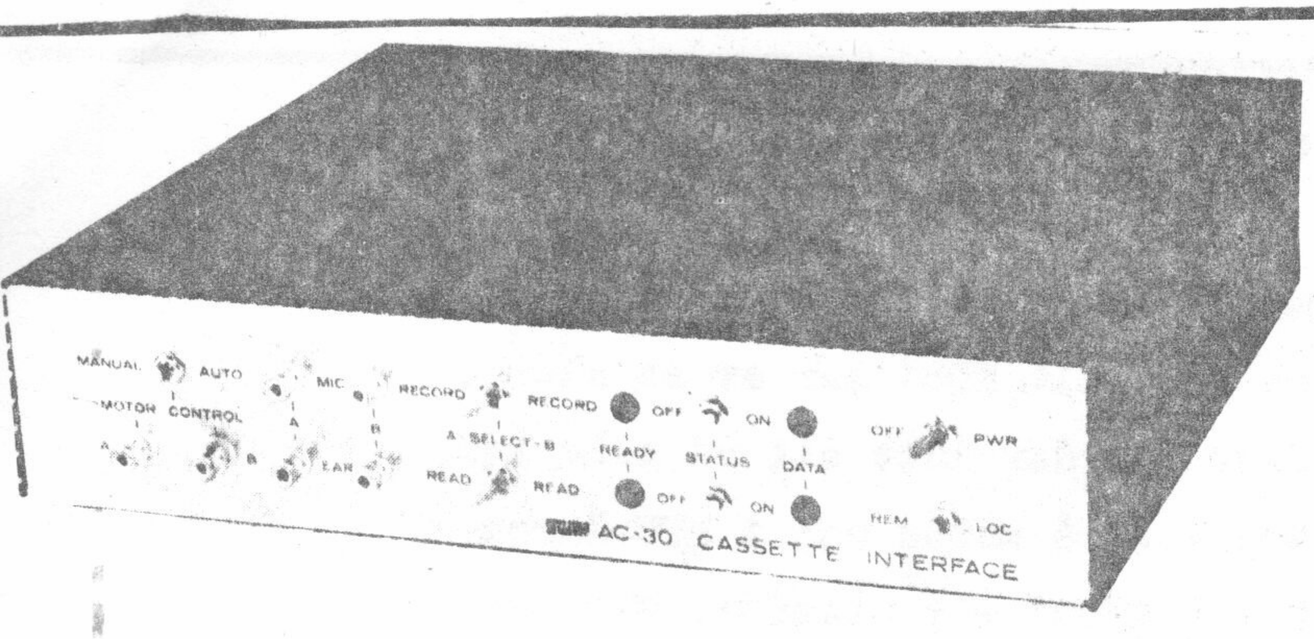
CT-64 Terminal: CT-64 2.340 **CT-VM Monitor:** CT-VM 1.260

MOD-2: Modulator för anslutning av CT-64 till en vanlig TV (UHF).

MOD-2 60:-

WEMAC AB

Box 72, S-133 01 Saltsjöbaden Sweden, Phone Stockholm (0)8 717 62 88



AC-30

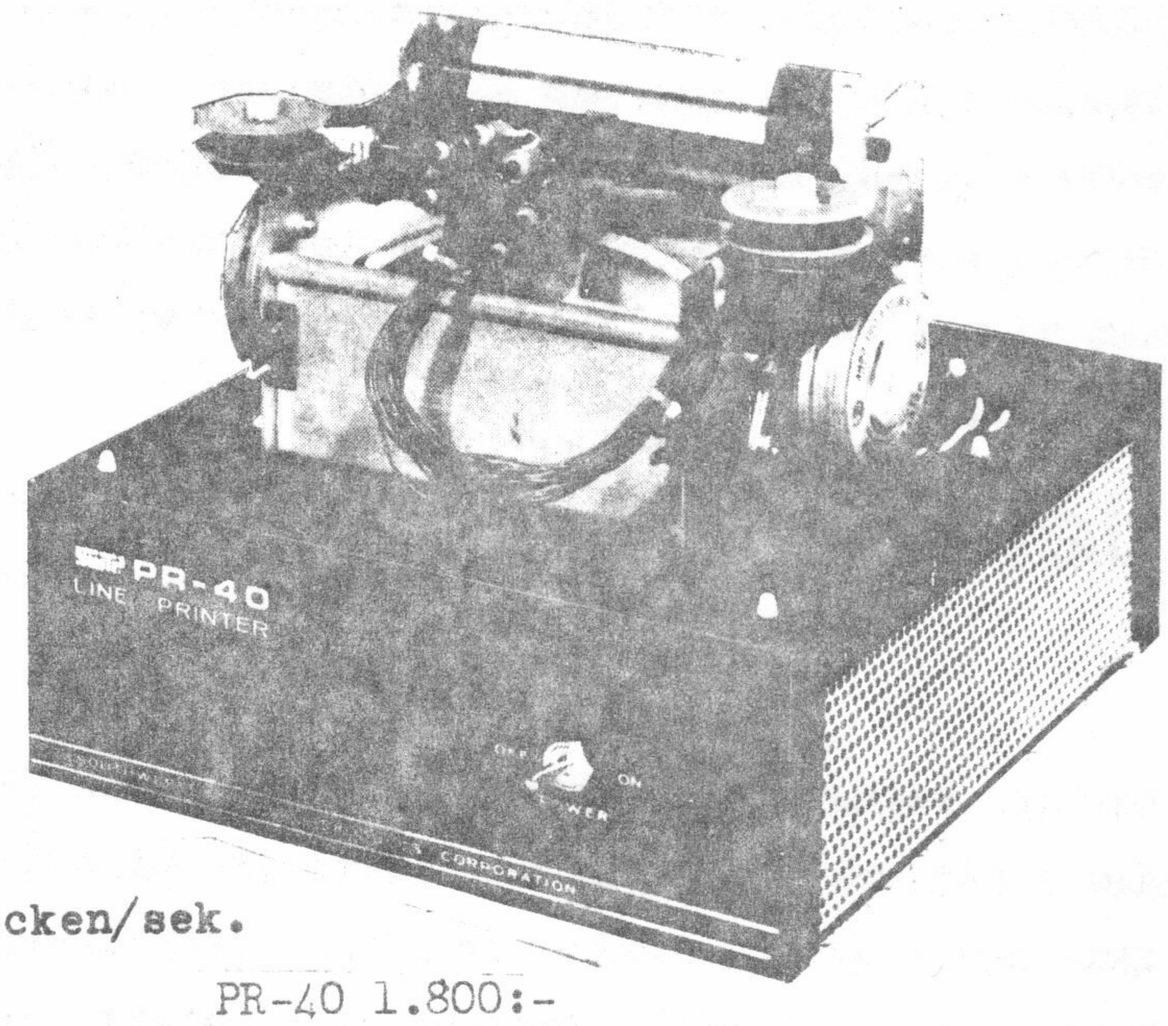
AUDIO CASSETTE INTERFACE

AC-30: 300 Bauds kassetteinterface med Byte/Cansas City - standard. Tillsammans med en CT-64 emulerar AC-30 en teleprinter med remsstans/remsläsare. Med AC-30 kan man fjärrstyra en eller två bandspelare.

AC-30 575:-

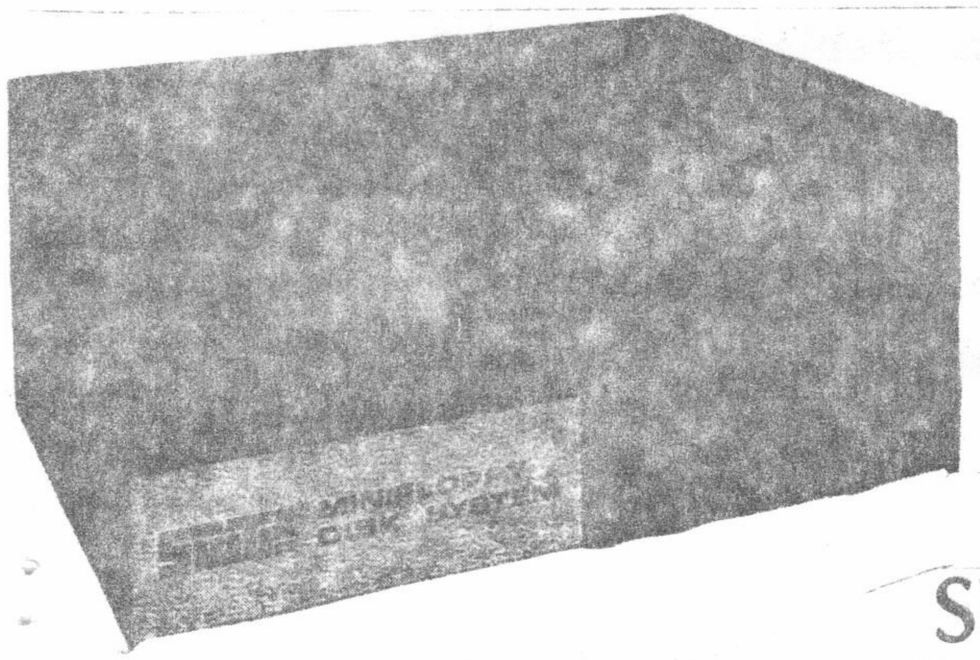
PR-40

ALPHANUMERIC PRINTER



PR-40: Printer 40 tecken/rad och 50 tecken/sek.

PR-40 1.800:-



MF-68

TWIN MINIFLOPPY DISK SYSTEM

MF-68 6.450:-

SWTBUG ROM OPERATING SYSTEM

SWTBUG: Ersättare för MICBUG med bl.a. Disc-Bootstrap.
Kommunikation genom äkta serie Interface.
Vektoriserad mjukvaruinterupt. - MICBUG kompatibel. -

SWTBUG 150:-

DESSUTOM FINNS ETT VÄLMATAT MJUKVARUBIBLIOTEK.

Box 72, 133 01 Saltsjöbaden 08/717 62 88, Torsvägen 61

EN LITEN BIT•••••••!

Ingress

Alla vi medlemmar i FD68 har en sak gemensam: Vi är intresserade av mikroda-- torer. En del av oss är 6800-fantaster, andra har valt FD68 mer av en slump.. Men att vi alla är intresserade av mikrodatorer betyder inte att vi alla kan lika mycket om dem. För nybörjaren kan facksnacket på möten och i MPU-laren ibland vara lite påfrästande och det är lätt att känna sig utanför. Men det avhjälps lättast genom att få del av de andras kunskaper - något som inte alltid är så lätt. Gapet kan vara lite för stort ibland. Dessutom känner man sig ofta lite dum när man frågar om saker som man tror att de andra behärskar perfekt. MPU-laren startar nu en serie artiklar för nybörjaren-och kanske också för dig som kan lite mer - om mikrodatorer och då naturligtvis 6800 i första hand. Artiklarna ska försöka lära ut litet om hur ett mikro - datorsystem är uppbyggt och om tekniken för maskinprogramering. Jag hoppas att vi ska komma en liten bit närmare varandra,Kunskapsmässigt, när serien är slutt

del 1

Det har sagts att en dator kan programeras att lösa allt som kan definieras. Med definiera menas här att lösningen på problemet kan brytas ned i en rad steg som i sin tur kan skrivas som datorinstruktioner. Att definiera ett pro - blem innan man börjar programera är mycket viktigt. Utan definition kan inga problem lösas. Men samtidigt kan definierandet av vissa problem ibland vara mycket svårt.

Det grundläggande för programmering är logist tänkande. Det behövs nämligen normalt lite logik, i bland gnska mycket, för att kunna definiera problemen, som man vill lösa, riktigt.

Datorns möjligheter och begränsningar.

En dator är en maskin och som alla maskiner måste den styras och övervakas för att utföra en uppgift. Tills dess att ett program är skrivet och lagrat i datorns minne "vet" den absolut ingenting, inte ens hur den ska ta in information från omvärlden. Den måste få reda på hur den ska bära sig åt,för att lösa alla de uppgifter den får.
Vissa datorer kan vara bättre lämpade för vissa typer av uppgifter och andra för andra typer,beroende på hur datorn är uppbyggd. En dator som är byggd att

förstå en multiplikationsinstruktion är bättre lämpad för vissa typer av beräkningar än en som måste använda en serie av additionsinstruktioner för att kunna multiplicera.

Det finns fem grundläggande fakta som är bra att hålla i huvudet när man arbetar med datorer.

1. Upprepande av operationer - en dator kan utföra likartade operationer tusentals gånger utan att tröttas eller för den skull göra fel.
2. Hastighet - datorn behandlar information med enorma hastigheter,direkt beroende av konstruktörens och programmerarens skicklighet.
3. Flexibilitet - Datorer avsedda för allmänna uppgifter kan programmeras att lösa problem av mycket skiftande slag.
4. Exakthet - Datorn kan programmeras att utföra beräkningar med den noggrannhet som programmeraren vill.
5. Intuition - en dator saknar helt intuition,den kan bara utföra saker precis som den är instruerad. Människan kan komma på svaret på ett problem utan att ha klarat av alla detaljer, men en dator måste gå den långa vägen.

Det finns gånger då det kan tyckas att datorn är intelligent, men det är bara resultatet av att den följer logiska steg uppställda av programmerare. Ett program kan bygga på ett annat program, som i sin tur kan bygga på ett tredje program, men i botten finns programmerarens / programmerarnas arbete, inte datorns.

Vad är en dator ?

Innan vi går vidare är det kanske lämpligt att mer detaljerat titta på hur en dator är upphyggd, vilket vi måste veta för att kunna programera.

Egentligen är datorn inte en maskin, utan en grupp maskiner. Centralenheten - CPU (vilket står för engelskans Central Processing Unit) - innehåller själva arbetsenheten i datorn, där behandlingen av informatin, data, sker. Inne i själva CPUn finns den aritmetiska och logiska enheten (ALU), de kretsar som avkodar, översätter, de instruktioner som datorn tar emot samt nödvändig logik för att nå datorns minne.

I en mikrodator finns alla dessa delar, utom minnet (vissa har t o m litet minne med), i en och samma kapsel, integrerade krets. Denna krets kallas mikroprocessor (MPU). Vanligen finns också minne, kretsar för att adressera

forts nästa sida

minnet och en del speciella kretsar för att koppla samman datorn med omvärlden, i den låda, eller något liknande, som vi kallar dator.

In och Ut.

De kretsar som kopplar samman datorn med omvärlden kallas interfacekretsar och bildar en eller flera s k In-Out-portar (I/O-portar). Det finns två olika typer av portar, pararella och seriella. Beteckningarna syftar på hur de minsta informationsenheterna, databitarna, förhåller sig till varandra när de går in eller ut från CPUn - om de är i serie eller om de kommer parallellt och samtidigt. Till detta kommer vi igen senare.

De andra delarna som normalt gör att CPUn fungerar som en dator är de egentligen in- och ut-enheterna, t ex tangentbord och bildskärm, tillsammans med någon ytterligare minnesenhet, t ex en bandspelare eller ett skivminne. Alla datorer måste ha åtminstone en in-och en ut-enhet för att kunna användas. Det är också mycket vanligt att man har något yttre minne, oftast en kasettbandspelare när det gäller mikrodatorer. Men kasettbandspelare är långsamma och inte särskilt väl lämpade att använda som s k massminne till datorer. Betydligt bättre, men också betydligt dyrare, är en s k floppy disk, som använder skivor belagda med ett magnetiserbart skikt som minne.

Skaaen mikrodator användas i någon form av affärsrörelse eller någon annanstans där t ex stora register måste läggas upp, är en floppy disk nödvändig. Vi återkommer till bandspelare och floppy disk senare i serien.

I nästa nummer kommer vi att gå in lite mer på djupet i datorn och lära oss litet mer om terminologin. Vi tittar också litet närmare på de instruktioner som styr datorn..

LARS OLOF FAGERSTRÖM

LUSEN

I nummer 3 av MPU-laren har väl alla upptäckt att sida 22 saknas samt lite konstigheter i slutet. Detta beror på en falör hos tryckeriet. Vi tar oss friheten att bifiga dessa 2 nya blad ,för att om ni önskar kunna byta dessa.

Hardware Half-Byte Shift Register M6800

Arne L Tillbom

$\overline{CS} = A2F_x$

$SHIFT = \overline{CS} \cdot A3 \cdot DBE$

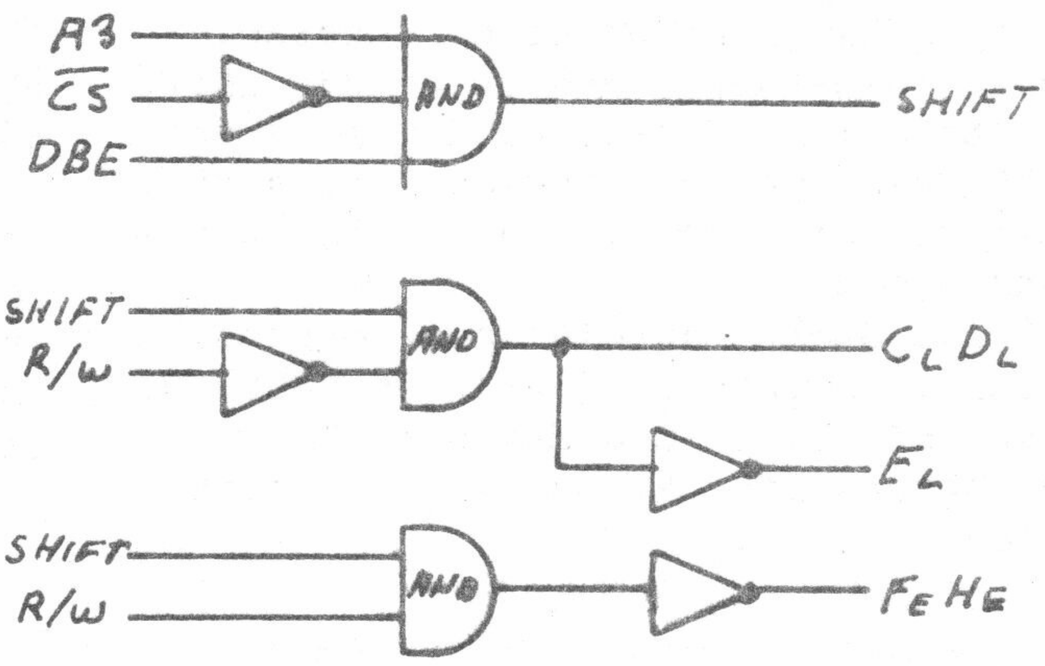
$C_L D_L = SHIFT \cdot \overline{R/W}$

$E_L = \overline{C_L D_L}$

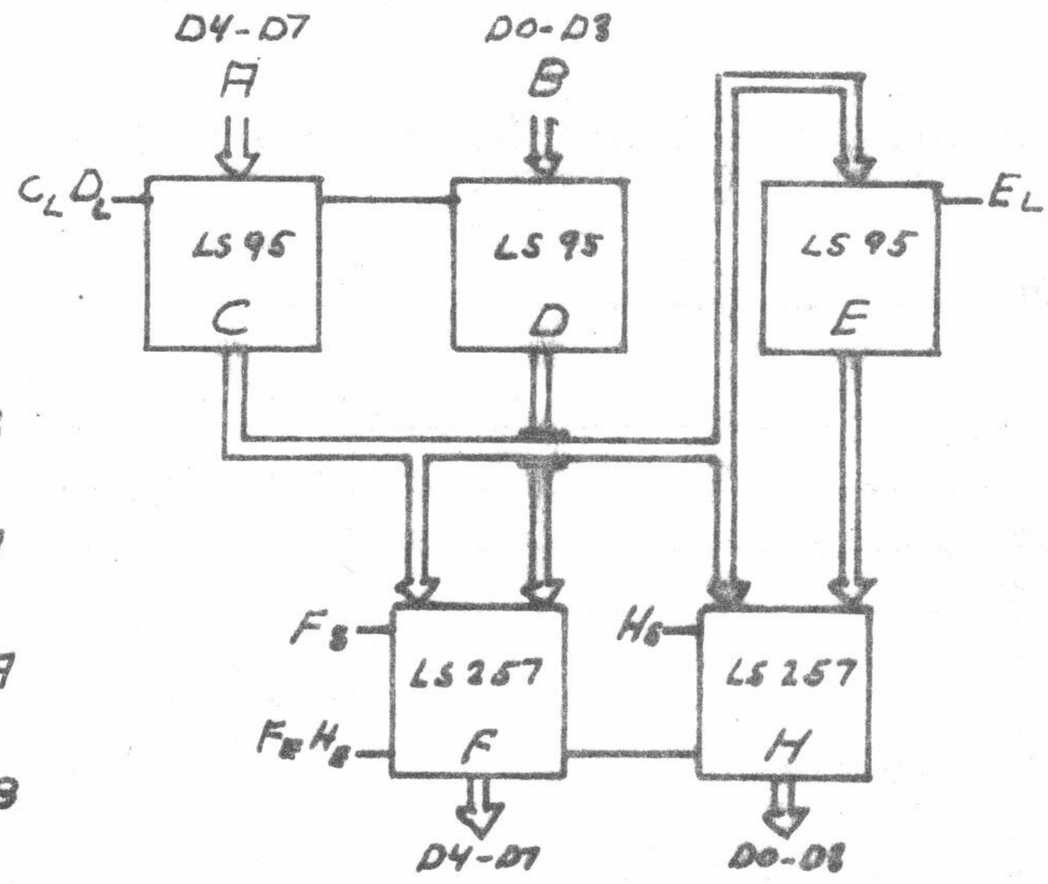
$F_S = A1$

$H_S = A0$

$\overline{F_E H_E} = SHIFT \cdot R/W$



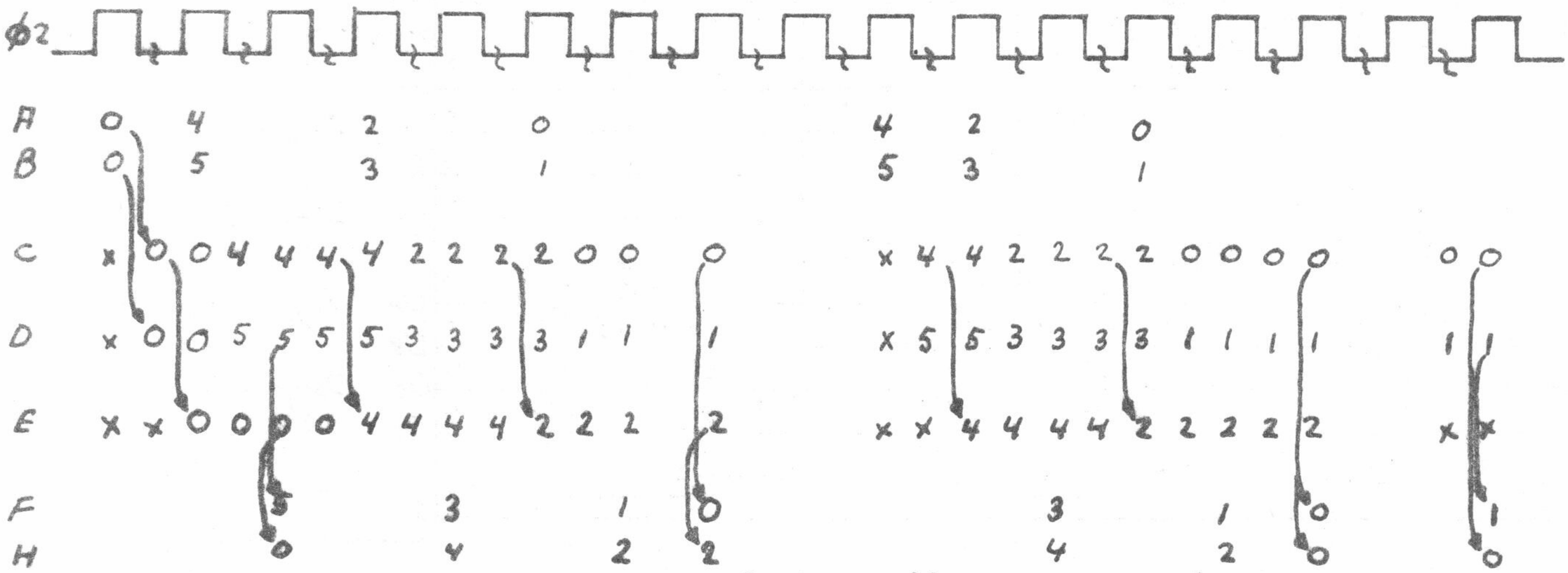
ØP	$C_L D_L$	E_L	F_S	H_S	$F_E H_E$
W					
R	D	E	READ	A2F8	
SWP	D	C	READ	A2F9	
L-SIGN	C	E	READ	A2FA	
R-SIGN	C	C	READ	A2FB	



SHIFT | 0 1 | 2 3 | 4 5 |

LEFT RIGHT SWAP

W W R W R W (R/LS) W W R W R RS SWP



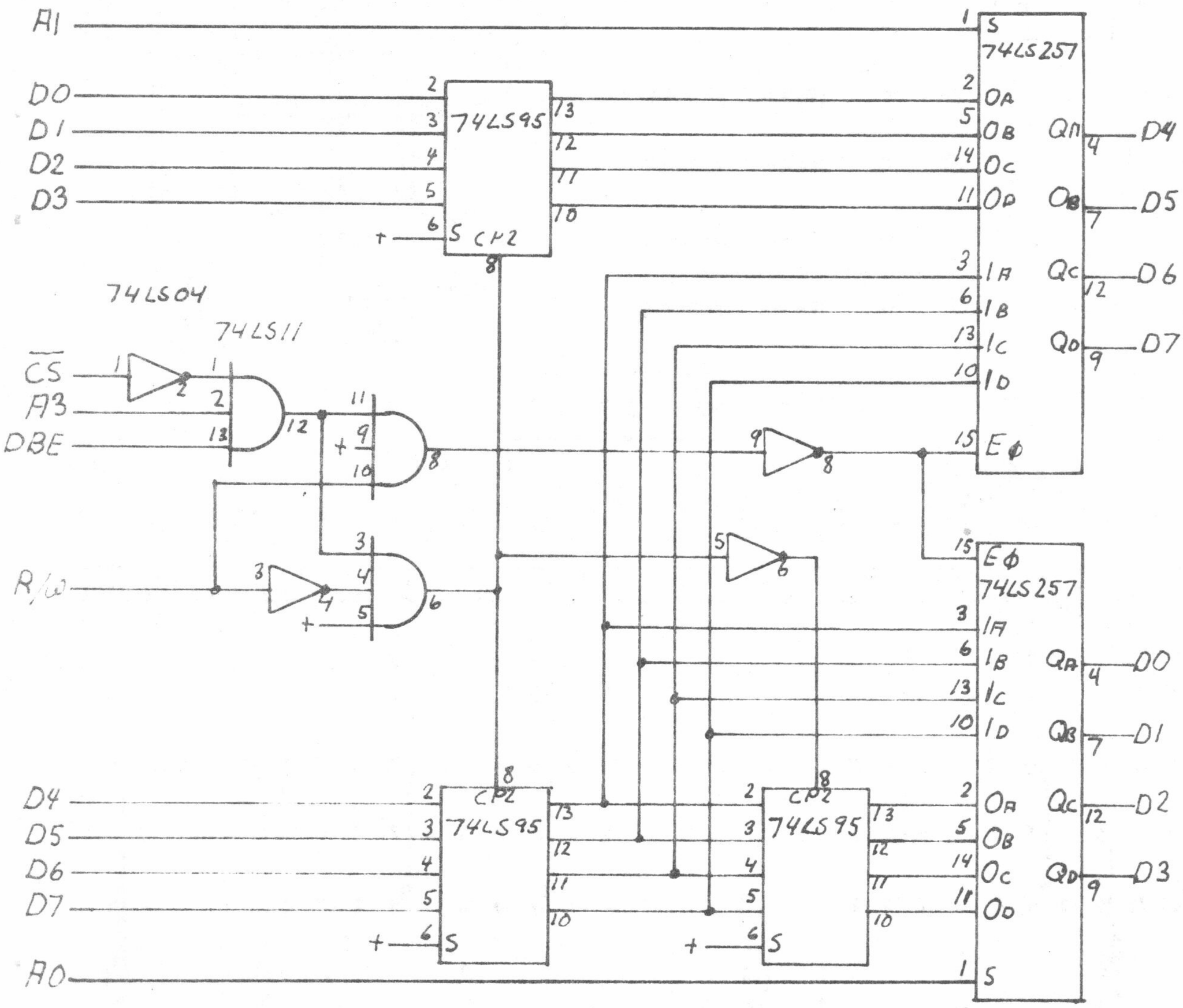
forts.n.sida

I samband med decimal aritmetik, tex. i BASIC, har man ofta behov att shifta ett fält fyra bitar höger eller vänster. Normalt löses detta med upprepade shift och rotate instruktioner, men på grund av att dessa måste genomlöpas fyra ggr. för varje shift-operation så blir tidsåtgången stor.
I SWTPc:s BASIC tar(614 khz) högershift 770 och vänstershift 1003 resp 1058 us. Motsvarande tider med Hardware-shift blir: högershift 347 och vänstershift 436 resp 449 us.

Med tanke på att en enda aritmetisk operation ´division´ kan kräva närmare fyrtio stycken shift operationer finns stora vinster att göra med hjälp av ett hårdvaru implementerat shiftregister.
Jag har byggt upp kretsen på samma kort som Hardware X-REG Stacken(MPULAREN 3/78) och utnyttjar avkodningen från denna.
Program patcharna till SWTPc:s BASIC går in på orginal platsen så inga adresser behöver ändras på annan plats i progr.

Lycka till
Arne L. Tillbom

HARDWARE HALF-BYTE SHIFT-REGISTER M6800



```
00020                          OPT  A O
00040                     *          RU4K      RU8K V2.0      RU8K V2.3
00050  045F               INCX6 EQU  A A045F      A0677          A06DF
00060                     *
00070  0567                     ORG  A A0567      A08BC          A0924
00080                     *
00090                     *                                ORGINAL KOD
00100  0567  36           SHLFTO PSHA                      36
00110  0568  37                  PSHB                      37
00120  0569  BD 045F            JSR  E INCX6              DF 57C6
00130  056C  C6 06              LDAB Ä 6                  04 DE
00140  056E  7F A2F8            CLR  E SHIFT              57 BDXX
00150                     *
00160  0571  09           SHLFT1 DEX                       XX
00170  0572  A6 00              LDAA X 0                  68 00
00180  0574  B7 A2F8            STAA E SHIFT              86 0509
00190  0577  B6 A2F8            LDAA E SHIFT              69 004A
00200  057A  A7 00              STAA X 0                  26 FA
00210  057C  5A                 DECB                      5A
00220  057D  26 F2              BNE  R SHLFT1             26 EE
00230  057F  33                 POPB                      33
00240  0580  32                 POPA                      32
00250  0581  39                 RTS                       39
00260                     *
00270  0582  36           SHRHT  PSHA                      36
00280  0583  37                  PSHB                      37
00290  0584  08                 INX                       DF
00300  0585  08                 INX                       57
00310  0586  08                 INX                       A6
00320  0587  08                 INX                       00
00330  0588  08                 INX                       84
00340  0589  A6 00              LDAA X 0                  F0 97
00350  058B  B7 A2F8            STAA E SHIFT              53 C604
00360  058E  C6 05              LDAB Ä 5                  DE 57
00370                     *
00380  0590  09           SHRHT1 DEX                       64
00390  0591  A6 00              LDAA X 0                  00 86
00400  0593  B7 A2F8            STAA E SHIFT              05 0866
00410  0596  B6 A2F8            LDAA E SHIFT              00 4A26
00420  0599  A7 01              STAA X 1                  FA 5A
00430  059B  5A                 DECB                      26
00440  059C  26 F2              BNE  R SHRHT1             F1 DE
00450  059E  B6 A2FB            LDAA E SHRDRS             57 A600
00460  05A1  01                 NOP                       9B
00470  05A2  01                 NOP                       53
00480  05A3  A7 00              STAA X 0                  A7 00
00490  05A5  33                 POPB                      33
00500  05A6  32                 POPA                      32
00510  05A7  39                 RTS                       39
00520                     *
00530  05A8                     END  A
TOTAL ERRORS   0
```

```
S00600004844521B
S11305673637BD045FC6067FA2F809A600B7A2F80E
S1130577B6A2F8A7005A26F233323936370808808E4
S11505870808A600B7A2F8C60509A600B7A2F8B6A2F83C
S1120599A7015A26F2B6A2FB0101A7003332399B
```

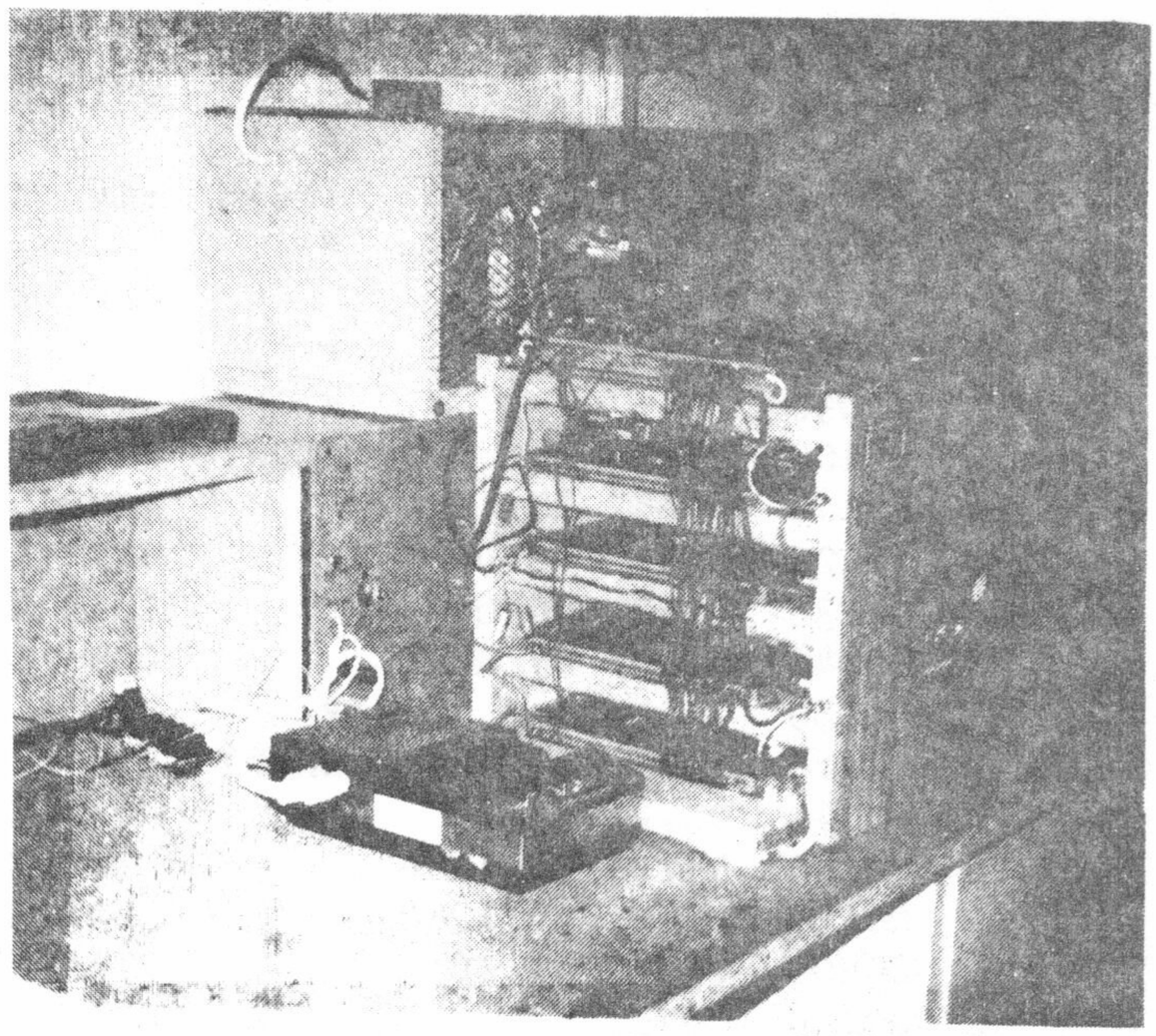
ACIBUG

BESKRIVNING AV ETT HEMBYGGE MED
UTVIDGAD MIKBUG

Årsskiftet 77-78 köpte jag ett D2-kit.Till detta hade jag 20k i 2102 oanvända i en låda.Strax efter träffade jag Alf Eriksson och han ordnade 4k-kort att sätta dem på.I och med det hade jag bestämt mig för "MOTOROLA-buss" och byggde en spånplatteram, där korten låg på våningar,se bild 1. Kontakterna jag använde var 30-poliga enkelsidiga som byggdes om till 2X43 poler.På senvåren fick jag möjlighet att köpa en terminal (TERMINET 300) genom jobbet.Då virade jag ett nytt CPU-kort med Alfs mönsterkort som grund. Det var försett med 4 st 1702 för att också kunna ta MINIBUG 3. Jag gjorde först i ordning MIKBUG,men behövde ju 3 st 1702 för att få med återstartsadressen. Sådan gratis PROM-plats var ju till för att utnyttjas. Jag började med att lägga in ett D2-kit kassettinterface för att kunna transportera program mellan CPU-korten.

ACIBUG

ACIBUG var alltså först en MIKBUG med en extra ACIA för att klara ett kassettinterface plankat från D2-kitet.I och med att jag lade in detta som kommandon var jag tvungen att gå in i källkoden för MIKBUG och därmed var lavinen i gång.Resultatet blev en BUG med följande kommandon:



A Alfanumerisk utskrift
Terminalen skriver ut innehållet i minnet och förutsätter att det är ASCII-kodat. Den startar i BEGA (A002 + 3) och slutar vid EOT (04) varvid den återvänder till ACIBUG.Rutinen ALFAN är skriven som en subrutin och använder subrutinen PDATA ur MIKBUG.

B Blockflyttare
Flyttar så många tecken som anges i BYTECT (A00B) från området ovanför BEGA till området ovanför ENDA. Ytterligare 256 tecken flyttas om rutinen åter anropas. Observera att om områdena överlappar varandra finns risk att man skriver över sånt som man vill flytta senare.Bäst är att i sådana fall göra flyttningen i två steg via en scratcharea. Rutinen BMOVE är skriven som en subrutin.

C Fortsätt

Vid SWI återvänder kontrollen till ACIBUG. Med C kan man fortsätta på andra sidan SWI.Detta möjliggör tracening av program. Man behöver dock stoppa in dessa extra SWI som senare ersätts av NOP,vilket innebär förlust av minne och exekveringstid. Man har också möjlighet att dela upp program i kortare delar som körs i tur och ordning.

D D2-load

Laddning från D2-kompatibla kassetter. Utförd som subrutin.

G Programstart

Motsvarar helt G i MIKBUG.

H Alfanumerisk inmatning ovanför stacken

Inmatning till den area ovanför stacken som sökrutinen använder.

I Alfanumerisk inmatning

Inmatning från tangentbordet till arean ovanför BEGA. Inmatningen avslutas med EOT (04, CTRL D) Om RAM-minne ej finns skrivs ? och ACIBUG får kontrollen. ALFIN är utförd som en subrutin.

L Laddning från kassett med S1-format.

Laddar kassetter som i MIKBUG, men via kassettinterfacet. Vid checksummefel erhålls SWI varvid sist inmatade cellen kan erhållas ur indexregistret. Med ändring på endast tre ställen kan inputrutinen styras över så att inmatning sker från terminalen.

M Minnesändring

Som i MIKBUG. Om man ej har tryckt på SPACE kan man ge en ny minnesadress utan att gå upp i ACIBUG genom att trycka på M och sedan slå in nya adressen.

P Stansning S1-format

Som i MIKBUG.Av utrymmesskäl kom ej möjligheten med S1-format till kassettinterfacet med.Det är av större vikt att kunna göra minnesdumpar till terminalen.

Q Direkt laddning från kassett

Tar kassettinnehållet och lägger i minnet oberoende av format, startkoder osv. Börjar i BEGA och slutar i ENDA (skyddar innehåll ovanför ENDA). Utförd som subrut.

R Register

Som i MIKBUG

S Sökrutin

Det minnesinnehåll man söker skall läggas ovanför stacken och avslutas med EOT (04). Sökningen börjar i BEGA. Slutet på en "match" kan avläsas ur indexreg. SEARCH är utförd som en subrutin och om den anropas från konsolen så ger "match" upphov till SWI efter RTS. Match erhålls normalt åtminstone i övre delen av stackminnet. Erhålls ej match återvänder rutinen till ACIBUG.

forts nästa sida

T Minnestest

Minnestest mellan BEGA och ENDA. MINTSO är utförd som subrutin, men minnesfel ger upphov till SWI där indexregistret pekar på felaktig cell.

SWI-interrupt

Som i MIKBUG

IRQ-interrupt

Som i MIKBUG

NMI-interrupt

Som SWI i MIKBUG. Kan användas som ABORT-funktion.

Kommentarer har bara tagits med på ny- eller omskriven kod. I övrigt hänvisas till källkoden för MIKBUG.

Carl Axel Bruno




```
PAGE  001              ACIBUG - 790420

00001                          TTL     ACIBUG - 790420
00002                          OPT     LLEN=110,PAGE=43,CREF
00003                  *-------*-----------------------------------------
00004                  * THIS IS AN EXTENSION OF MIKBUG (OWNED BY MOTOROLA)
00005                  * MADE BY C A BRUNO. FOR THE COMMENTS ON THOSE PARTS SEE MIKBUG LISTINGS
00006                  *
00007                  * ACIBUG CONTAINS THE FOLLOWING FUNCTIONS:
00008                  *
00009                  *  A ALFANUMERICAL OUTPUT  S
00010                  *  B BLOCKMOVE  S
00011                  *  C CONTINUE
00012                  *  D D2-COMPATIBLE LOAD  S
00013                  *  G GO
00014                  *  H ALFANUMERICAL INPUT FROM "TEXT" AND UPWARDS
00015                  *  I ALFANUMERICAL INPUT  S
00016                  *  L LOAD FROM CASSETTE (SII-FORMAT)
00017                  *  M MEMORY CHANGE
00018                  *  O D2-COMPATIBLE PUNCH  S
00019                  *  P PUNCH TO PRINTER (SII-FORMAT)
00020                  *  Q CASSETTE DIRECT LOAD  S
00021                  *  R DISPLAY REGISTERS
00022                  *  S SEARCH (SHALL BE PROCEEDED BY A "H"COMMAND FOLLOWED BY TEXT (OR MAN. INPUT)  S
00023                  *  T MEMORYTEST  S
00024                  *
00025                  *-----------------------------------------------------
00026                  *
00027                  * HARDWARE EQUALS
00028                  *
00029  8008  A ACIAS   EQU     $8008
00030  8009  A ACIAD   EQU     $8009
00031  8007  A PIASB   EQU     $8007
00032  8006  A PIADB   EQU     $8006
00033  8005  A PIAS    EQU     $8005
00034  8004  A PIAD    EQU     $8004
```

```
00036                          *
00037                          * START OF PROGRAM
00038                          *
00039A E000                            ORG     $E000
00040A E000 FE A000   A IRQINT LDX     IOV       * THE NMI-INTERRUPT IS USED FOR ABORT FUNCTION (CMP SWI)
00041A E003 6E 00     A        JMP     0,X
00042                          *--------------------------------
00043A E005 8E A042   A STAR   LDS     #STAKTO   * A PART OF STARTUP IS PATCHED TO HERE
00044A E008 86 03     A        LDAA    #$03      * RESET OF ACIA
00045A E00A B7 8008   A        STAA    ACIAS
00046A E00D 7E E0D3   A        JMP     STARK
00047                          *--------------------------------
00048A E010 CE A049   A SEARCH LDX     #TEXT-1   * THIS IS A NEW SEARCH ROUTINE. LOAD IS MOVED TO $E307
00049A E013 08                 SEAR2   INX               * INCREASE THE GOAL POINTER
00050A E014 FF A004   A        STX     ENDA      * STORE THE GOAL POINTER IN ENDA
00051A E017 E6 00     A        LDAB    0,X       * GET NEXT GOAL LETTER
00052A E019 FE A002   A        LDX     BEGA      * GET THE SEARCH POINTER
00053A E01C C1 04     A        CMPB    #$04      * HAVE YOU REACHED THE GOAL?
00054A E01E 26 01 E021         BNE     SEAR3     * IF NOT
00055A E020 39                 RTS               * RETURN IF YOU HAVE FOUND IT!
00056A E021 08           SEAR3 INX               *INCREASE THE SEARCH POINTER
00057A E022 FF A002   A        STX     BEGA      * STORE THE SEARCH POINTER IN BEGA
00058A E025 8C FFFF   A        CPX     #$FFFF    * RETURN TO CONTRL IF YOU HAVE REACHED TOP OF MEMORY
00059A E028 27 1A E044         BEQ     LOAD21
00060A E02A A6 00     A        LDAA    0,X       * GET A LETTER FOR TEST
00061A E02C 11                 CBA
00062A E02D 26 05 E034         BNE     SEAR4     * IF TEST NOT SATISFACTORY
00063A E02F FE A004   A        LDX     ENDA      * GET GOAL POINTER
00064A E032 20 DF E013         BRA     SEAR2
00065A E034 86 49     A SEAR4  LDAA    #$49      * GET LSB OF TEXT-1
00066A E036 08                 INX
00067A E037 F6 A005   A        LDAB    ENDA+1    * GET LSB OF GOAL POINTER
00068A E03A 09           SEAR5 DEX               * DECREASE THE SEARCH POINTER UNTIL ONE LETTER AFTER "START OF H
00069A E03B 5A                 DECB
00070A E03C 11                 CBA
00071A E03D 26 FB E03A         BNE     SEAR5     * IF START OF HOPE+1
00072A E03F FF A002   A        STX     BEGA      * START SEARCHING FROM HERE
00073A E042 20 CC E010         BRA     SEARCH
00074                          *--------------------------------
00075A E044                            ORG     $E044     * NORMAL MIKBUG FROM HERE AND ONWARDS(SOME EXTRA ADDRESSES ADDED
00076A E044 7E E0E3   A LOAD21 JMP     CONTRL
00077A E047 8D 0C E055   BADDR BSR     BYTE
00078A E049 B7 A00C   A        STAA    XHI
00079A E04C 8D 07 E055         BSR     BYTE
00080A E04E B7 A00D   A BAD    STAA    XLOW
00081A E051 FE A00C   A        LDX     XHI
00082A E054 39                 RTS
00083A E055 8D 53 E0AA   BYTE  BSR     INHEX
00084A E057 48                 ASLA
00085A E058 48                 ASLA
00086A E059 48                 ASLA
00087A E05A 48                 ASLA
00088A E05B 16                 TAB
00089A E05C 8D 4C E0AA         BSR     INHEX
00090A E05E 1B           BYT   ABA
00091A E05F 16                 TAB
00092A E060 FB A00A   A        ADDB    CKSM
00093A E063 F7 A00A   A        STAB    CKSM
00094A E066 39                 RTS
00095A E067 44           OUTHL LSRA
00096A E068 44                 LSRA
00097A E069 44                 LSRA
00098A E06A 44                 LSRA
00099A E06B 84 0F     A OUTHR  ANDA    #$0F
00100A E06D 8B 30     A        ADDA    #$30
00101A E06F 81 39     A        CMPA    #$39
00102A E071 23 02 E075         BLS     OUTCH
00103A E073 8B 07     A        ADDA    #$07
00104A E075 7E E1D1   A OUTCH  JMP     OUTEEE
00105A E078 7E E1AC   A INCH   JMP     INEEE
00106A E07B 8D F8 E075 PDATA2  BSR     OUTCH
00107A E07D 08                 INX
00108A E07E A6 00     A PDATA1 LDAA    0,X
00109A E080 81 04     A        CMPA    #$04
00110A E082 26 F7 E07B         BNE     PDATA2
00111A E084 39                 RTS
00112A E085 8D C0 E047 CHANGE  BSR     BADDR
00113A E087 CE E19D   A CHA51  LDX     #MCL
00114A E08A 8D F2 E07E         BSR     PDATA1
00115A E08C CE A00C   A        LDX     #XHI
00116A E08F 8D 37 E0C8         BSR     OUT4HS
00117A E091 FE A00C   A        LDX     XHI
00118A E094 8D 34 E0CA         BSR     OUT2HS
00119A E096 FF A00C   A        STX     XHI
00120A E099 8D DD E078         BSR     INCH
00121A E09B 81 4D     A        CMPA    #$4D
```



forts.n.sida

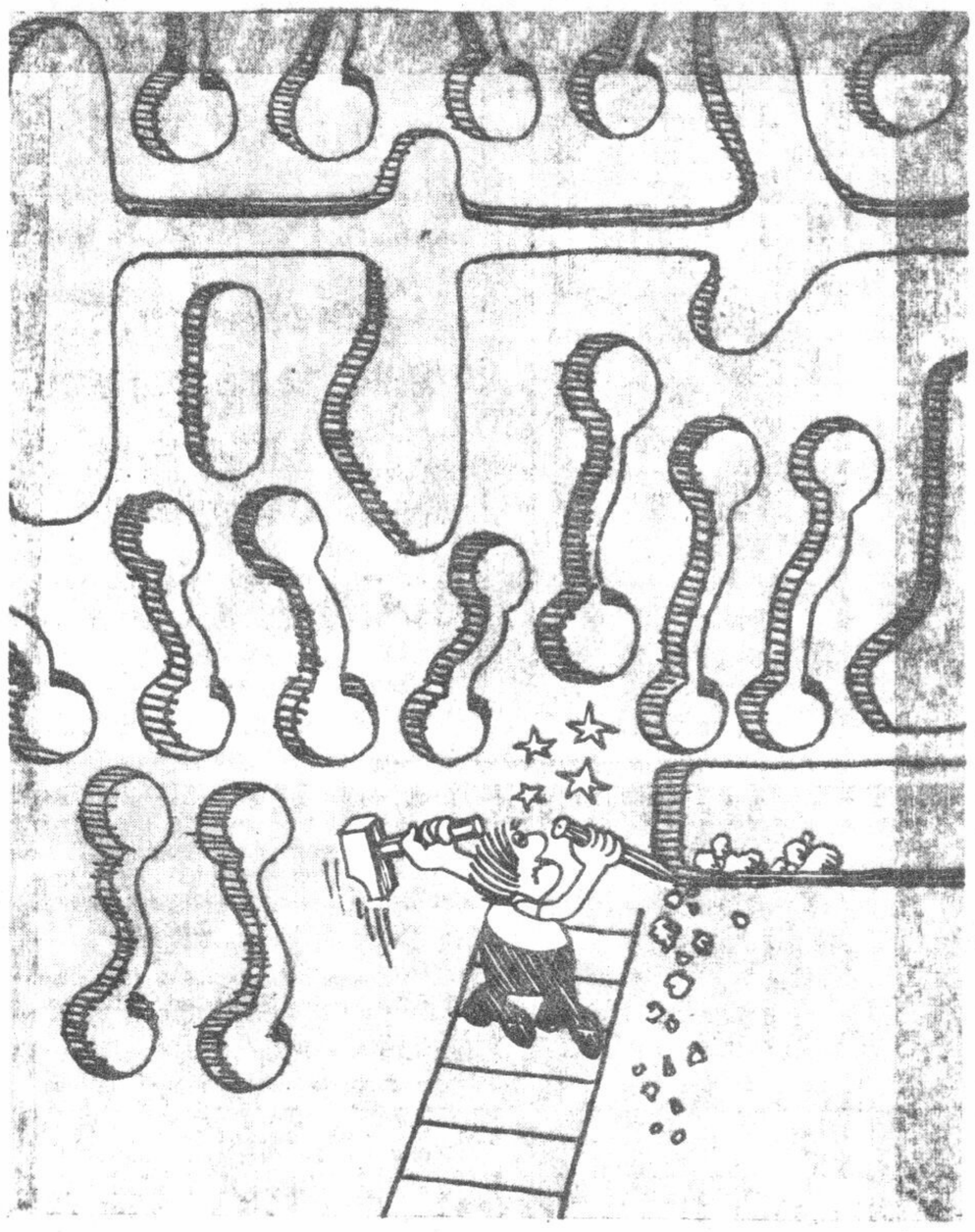
```
00122A E09D 27 E6 E085          . BEQ     CHANGE
00123A E09F 81 20    A            CMPA    #$20
00124A E0A1 26 E4 E087            BNE     CHA51
00125A E0A3 8D B0 E055            BSR     BYTE
00126A E0A5 09                    DEX
00127A E0A6 01                    NOP
00128A E0A7                       ORG     $E0A7
00129A E0A7 7E E3DF  A            JMP     CHA61     * CONTINUED AT $E3D5 (YOU CAN GIVE THE "M"-COMMAND IN MEMORYCHAN
00130                       * EXAMPLE:    *M0000
00131                       *              0000 FF M1000
00132                       *              1000 AA BB
00133                       *              1001 55
00134A E0AA 8D CC E078 INHEX  BSR     INCH
00135A E0AC 80 30    A INHE   SUBA    #$30
00136A E0AE 2B 94 E044            BMI     LOAD21
00137A E0B0 81 09    A            CMPA    #$09
00138A E0B2 2F 0A E0BE            BLE     INIHG
00139A E0B4 81 11    A            CMPA    #$11
00140A E0B6 2B 8C E044            BMI     LOAD21
00141A E0B8 81 16    A            CMPA    #$16
00142A E0BA 2E 88 E044            BGT     LOAD21
00143A E0BC 80 07    A            SUBA    #$07
00144A E0BE 39              INIHG RTS
00145A E0BF A6 00    A OUT2H  LDAA    0,X
00146A E0C1 8D A4 E067 OUT2HA BSR     OUTHL
00147A E0C3 A6 00    A            LDAA    0,X
00148A E0C5 08                    INX
00149A E0C6 20 A3 E06B            BRA     OUTHR
00150A E0C8 8D F5 E0BF OUT4HS BSR     OUT2H
00151A E0CA 8D F3 E0BF OUT2HS BSR     OUT2H
00152A E0CC 86 20    A OUTS   LDAA    #$20
00153A E0CE 20 A5 E075            BRA     OUTCH
00154A E0D0 7E E005  A START  JMP     STAR      * A PART OF STARTROUTINE IS PATCHED AWAY
00155A E0D3 BF A008  A STARK  STS     SP
00156A E0D6 CE 8004  A            LDX     #PIAD
00157A E0D9 6C 00    A            INC     0,X
00158A E0DB 86 07    A            LDAA    #$07
00159A E0DD A7 01    A            STAA    1,X
00160A E0DF 6C 00    A            INC     0,X
00161A E0E1 A7 02    A            STAA    2,X
00162A E0E3 86 34    A CONTRL LDAA    #$34
00163A E0E5 B7 8007  A            STAA    PIASB
00164A E0E8 B7 8006  A            STAA    PIADB
00165A E0EB 8E A042  A            LDS     #STAKTO
00166A E0EE CE E19C  A            LDX     #MCLOFF
00167A E0F1 8D 8B E07E            BSR     PDATAI
00168A E0F3 8D 83 E078            BSR     INCH
00169A E0F5 16                    TAB
00170A E0F6 8D D4 E0CC            BSR     OUTS
00171A E0F8 C1 4C    A            CMPB    #$4C      * NEW LOADFUNCTION VIA ACIA
00172A E0FA 26 03 E0FF            BNE     *+5
00173A E0FC 7E E307  A            JMP     LOAD      * OBSERVE THAT THE LOADFUNCTION IS MOVED
00174A E0FF C1 4D    A            CMPB    #$4D
00175A E101 27 82 E085            BEQ     CHANGE
00176A E103 C1 52    A            CMPB    #$52
00177A E105 27 18 E11F            BEQ     PRINT
00178A E107 C1 50    A            CMPB    #$50
00179A E109 27 32 E13D            BEQ     PUNCH
00180A E10B C1 47    A            CMPB    #$47
00181A E10D 26 04 E113            BNE     CONTIN
00182A E10F BE A008  A            LDS     SP
00183A E112 3B                    RTI
00184A E113 C1 53    A CONTIN CMPB    #$53      * A NEW SEARCHROUTINE IS INSERTED
00185A E115 26 04 E11B            BNE     CONTIS
00186A E117 8D E010  A            JSR     SEARCH
00187A E11A 3F                    SWI                * SWI IS USED TO TELL WHERE IT WAS FOUND (IX=LAST LETTER)
00188A E11B 7E E356  A CONTIS JMP     CONTI     * THE REST OF THE INPUT SCAN IS DONE IN ROM 4(E300-E3FF)
00189A E11E 01                    NOP
00190                       *-------------------------
00191A E11F                       ORG     $E11F
00192A E11F FE A008  A PRINT  LDX     SP
00193A E122 08                    INX
00194A E123 8D A5 E0CA            BSR     OUT2HS
00195A E125 8D A3 E0CA            BSR     OUT2HS
00196A E127 8D A1 E0CA            BSR     OUT2HS
00197A E129 8D 9D E0C8            BSR     OUT4HS
00198A E12B 8D 9B E0C8            BSR     OUT4HS
00199A E12D CE A008  A            LDX     #SP
00200A E130 8D 96 E0C8            BSR     OUT4HS
00201A E132 20 AF E0E3 C2         BRA     CONTRL
00202A E134                       ORG     $E134
00203                       *-----------------------------
00204A E134    0D    A MTAPE1 FCB     $0D
00205A E135    0A    A        FCB     $0A
00206A E136    00    A        FCB     $00
00207A E137    00    A        FCB     $00
```

```
00208A E138    00     A        FCB    $00
00209A E139    00     A        FCB    $00
00210A E13A    53     A        FCB    $53
00211A E13B    31     A        FCB    $31
00212A E13C    04     A        FCB    $04
00213A E13D 86 51     A PUNCH  LDAA   #$51
00214A E13F B7 8008   A        STAA   ACIAS
00215A E142 FE A002   A        LDX    BEGA
00216A E145 FF A00F   A        STX    TW
00217A E148 B6 A005   A PUN11  LDAA   ENDA+1
00218A E14B B0 A010   A        SUBA   TW+1
00219A E14E F6 A004   A        LDAB   ENDA
00220A E151 F2 A00F   A        SBCB   TW
00221A E154 26 04 E15A         BNE    PUN22
00222A E156 81 10     A        CMPA   #$10
00223A E158 25 02 E15C         BCS    PUN23
00224A E15A 86 0F     A PUN22  LDAA   #$0F
00225A E15C 8B 04     A PUN23  ADDA   #$04
00226A E15E B7 A011   A        STAA   MCONT
00227A E161 80 03     A        SUBA   #$03
00228A E163 B7 A00E   A        STAA   TEMP
00229A E166 CE E134   A        LDX    #MTAPE1
00230A E169 BD E07E   A        JSR    PDATA1
00231A E16C 5F                 CLRB
00232A E16D CE A011   A        LDX    #MCONT
00233A E170 8D 25 E197         BSR    PUNT2
00234A E172 CE A00F   A        LDX    #TW
00235A E175 8D 20 E197         BSR    PUNT2
00236A E177 8D 1E E197         BSR    PUNT2
00237A E179 FE A00F   A        LDX    TW
00238A E17C 8D 19 E197 PUN32   BSR    PUNT2
00239A E17E 7A A00E   A        DEC    TEMP
00240A E181 26 F9 E17C         BNE    PUN32
00241A E183 FF A00F   A        STX    TW
00242A E186 53                 COMB
00243A E187 37                 PSHB
00244A E188 30                 TSX
00245A E189 8D 0C E197         BSR    PUNT2
00246A E18B 33                 PULB
00247A E18C FE A00F   A        LDX    TW
00248A E18F 09                 DEX
00249A E190 BC A004   A        CPX    ENDA
00250A E193 26 B3 E148         BNE    PUN11
00251A E195 20 9B E132         BRA    C2
00252A E197 EB 00     A PUNT2  ADDB   0,X
00253A E199 7E E0BF   A        JMP    OUT2H
00254A E19C    13     A MCLOFF FCB    $13
00255A E19D    0D     A MCL    FCB    $0D
00256A E19E    0A     A        FCB    $0A
00257A E19F    14     A        FCB    $14
00258A E1A0    00     A        FCB    $00
00259A E1A1    00     A        FCB    $00
00260A E1A2    00     A        FCB    $00
00261A E1A3    2A     A        FCB    $2A
00262A E1A4    04     A        FCB    $04
00263A E1A5 FF A012   A SAV    STX    XTEMP
00264A E1A8 CE 8004   A        LDX    #PIAD
00265A E1AB 39                 RTS
00266A E1AC 37          INEEE  PSHB
00267A E1AD 8D F6 E1A5         BSR    SAV
00268A E1AF A6 00     A INI    LDAA   0,X
00269A E1B1 2B FC E1AF         BMI    INI
00270A E1B3 6F 02     A        CLR    2,X
00271A E1B5 8D 3C E1F3         BSR    DE
00272A E1B7 8D 36 E1EF         BSR    DEL
00273A E1B9 C6 04     A        LDAB   #$04
00274A E1BB E7 02     A        STAB   2,X
00275A E1BD 58                 ASLB
00276A E1BE 8D 2F E1EF IN3     BSR    DEL
00277A E1C0 0D                 SEC
00278A E1C1 69 00     A        ROL    0,X
00279A E1C3 46                 RORA
00280A E1C4 5A                 DECB
00281A E1C5 26 F7 E1BE         BNE    IN3
00282A E1C7 8D 26 E1EF         BSR    DEL
00283A E1C9 84 7F     A        ANDA   #$7F
00284A E1CB 81 7F     A        CMPA   #$7F
00285A E1CD 27 E0 E1AF         BEQ    INI
00286A E1CF 20 12 E1E3         BRA    IOUT2
00287A E1D1 37          OUTEEE PSHB
00288A E1D2 8D D1 E1A5         BSR    SAV
00289A E1D4 C6 0A     A IOUT   LDAB   #$0A
00290A E1D6 6A 00     A        DEC    0,X
00291A E1D8 8D 19 E1F3         BSR    DE
00292A E1DA 8D 13 E1EF OUT1    BSR    DEL
00293A E1DC A7 00     A        STAA   0,X
```

forts.n.sida

```
00294A E1DE 0D                  SEC
00295A E1DF 46                  RORA
00296A E1E0 5A                  DECB
00297A E1E1 26 F7 E1DA          BNE     OUTI
00298A E1E3 E6 02    A IOUT2    LDAB    2,X
00299A E1E5 58                  ASLB
00300A E1E6 2A 02 E1EA          BPL     IOS
00301A E1E8 8D 05 E1EF          BSR     DEL
00302A E1EA FE A012  A IOS      LDX     XTEMP
00303A E1ED 33                  PULB
00304A E1EE 39                  RTS
00305A E1EF 6D 02    A DEL      TST     2,X
00306A E1F1 2A FC E1EF          BPL     DEL
00307A E1F3 6C 02    A DE       INC     2,X
00308A E1F5 6A 02    A          DEC     2,X
00309A E1F7 39                  RTS
00310                *----------------------------------------
00311A E1F8 86 10    A D2LOAD   LDAA    #$10      * 00010000 DIVIDE BY ONE
00312A E1FA B7 8008  A          STAA    ACIAS     * THE NORMAL D2-KIT LOAD IS INSERTED HERE
00313A E1FD BD E35C  A BILD     JSR     INACIA
00314A E200 81 42    A          CMPA    #$42
00315A E202 27 05 E209          BEQ     RDBLCK
00316A E204 81 47    A          CMPA    #$47
00317A E206 26 F5 E1FD          BNE     BILD
00318A E208 39                  RTS
00319A E209 BD E35C  A RDBLCK   JSR     INACIA
00320A E20C 16                  TAB
00321A E20D 5C                  INCB
00322A E20E BD E35C  A          JSR     INACIA
00323A E211 B7 A002  A          STAA    BEGA
00324A E214 BD E35C  A          JSR     INACIA
00325A E217 B7 A003  A          STAA    BEGA+1
00326A E21A FE A002  A          LDX     BEGA
00327A E21D BD E35C  A STBLCK   JSR     INACIA
00328A E220 A7 00    A          STAA    0,X
00329A E222 08                  INX
00330A E223 5A                  DECB
00331A E224 26 F7 E21D          BNE     STBLCK
00332A E226 20 D5 E1FD          BRA     BILD
00333                *----------------------------------------
00334A E228 FE A002  A BMOVE    LDX     BEGA      * OBS! OM ENDA< BEGA SA MASTE ENDA+BYTECT<BEGA,
00335                *                           * ANNARS SKRIVS INFO ÖVER
00336                * OBSERVA IF END < BEGA CHECK THAT ENDA + BYTECT<BEGA OR YOUR INFO WILL BE DESTROYED
00337A E22B A6 00    A          LDAA    0,X
00338A E22D 08                  INX
00339A E22E FF A002  A          STX     BEGA      * YOU WILL MOVE BYTECT BYTES FROM BEGA+1-BEGA+BYTECT TO ENDA-END
00340A E231 FE A004  A          LDX     ENDA      *GET NEW ADDRESS OF BYTE
00341A E234 A7 00    A          STAA    0,X
00342A E236 08                  INX
00343A E237 FF A004  A          STX     ENDA      * INCREASE "NEW" POINTER
00344A E23A 7A A00B  A          DEC     BYTECT    * DECREASE BYTECOUNTER
00345A E23D 26 E9 E228          BNE     BMOVE     * ANOTHER BYTE TO BE MOVED?
00346A E23F 39                  RTS
00347                *------------------------------
00348A E240 86 00    A MINTS0   LDAA    #$00      * A NEW MEMORYTESTROUTINE
00349A E242 FE A002  A MINTS1   LDX     BEGA      * GET STARTADDRESS
00350A E245 B7 A04A  A          STAA    TEXT      * USE TEXT (IN STACKRAM) AS TEMP. STORAGE)
00351A E248 A7 00    A MINTS2   STAA    0,X       * STORE 01-FF IN RAM UPWARDS FROM BEGA(00 IN BEGA)
00352A E24A 81 FF    A          CMPA    #$FF      * DO NOT STORE $00
00353A E24C 26 02 E250          BNE     MINTS3    * IF NOT $00
00354A E24E 86 00    A          LDAA    #$00      * IF $00
00355A E250 08          MINTS3  INX               * INCREASE STORE POINTER
00356A E251 4C                  INCA              * AND THE TESTVALUE
00357A E252 BC A004  A          CPX     ENDA      * UNTIL THE TESTED AREA IS FULL WITH TESTVALUES
00358A E255 26 F1 E248          BNE     MINTS2    * IF NOT FULL YET
00359A E257 FE A002  A MINTS4   LDX     BEGA
00360A E25A F6 A04A  A          LDAB    TEXT      * THEN YOU RESTORE
00361                *   THE ADDRESSSTARTVALUE AND THE OUGHT-TO-BE-VALUE FROM THE STACK-RAM
00362A E25D A6 00    A MINTS5   LDAA    0,X       * AND ITS ASSOCIATED TESTVALUE
00363A E25F 11                  CBA
00364A E260 27 01 E263          BEQ     MINTS6    * IF THEY ARE ALIKE YOU DO NOT PULL THE EMERGENCY STOP
00365A E262 3F                  SWI
00366A E263 C1 FF    A MINTS6   CMPB    #$FF
00367A E265 26 02 E269          BNE     MINTS7
00368A E267 C6 00    A          LDAB    #$00      * YOU STILL DO NOT WANT $00 AS TESTVALUE
00369A E269 08          MINTS7  INX               * AND YOU INCREASE THE ADDRESS AND THE OUGHT-TO-BE-
00370                *  VALUE AS WAS MADE WHEN STORING TESTDATA
00371A E26A 5C                  INCB
00372A E26B BC A004  A          CPX     ENDA      * UNTIL YOU REACH THE END
00373A E26E 26 ED E25D          BNE     MINTS5    * IF NOT THE END YET
00374A E270 B6 A04A  A          LDAA    TEXT      * THEN YOU TAKE THE VALUE YOU USED IN BEGA AND INCREASE IT
00375A E273 4C                  INCA
00376A E274 81 FF    A          CMPA    #$FF
00377A E276 26 CA E242          BNE     MINTS1    * AND USE IT FOR ANOTHER ROUND
00378A E278 39                  RTS               * UNTIL YOU HAVE MADE 254 ROUNDS
00379                *------------------------------
```

```
00380A E279 FE A008   A FORTSA LDX    SP        * THIS "CONTINUE"-ROUTINE PROCEEDS AFTER A SWI AS IF NOTHING HAP
00381A E27C 86 FF     A        LDAA   #$FF
00382A E27E A1 07     A        CMPA   7,X       * CHECK IF THE SWI ADDRESS' LSB WAS $FF
00383A E280 26 02 E284         BNE    *+4       * IF SO
00384A E282 6C 06     A        INC    6,X
00385A E284 6C 07     A        INC    7,X       * SET PC TO THE ADDRESS AFTER SWI
00386A E286 BE A008   A        LDS    SP        * AND RESTORE THE STACKPOINTER
00387A E289 3B                 RTI              * AND LET'S FLY
00388                        *-------------------------------
00389A E28A 86 14     A DIRLOA LDAA   #$14      * THIS IS A LOADROUTINE THAT NEEDS NO SYNCSIGNS.
00390                        * YOU JUST LOAD HIGHER AND HIGHER IN MEMORY UNTIL YOU REACH THE CEILING (ENDA)
00391A E28C B7 8008   A        STAA   ACIAS     * INITIATE ACIA (8 BIT , NO PC , 1 STOPBIT
00392A E28F FE A002   A        LDX    BEGA
00393A E292 BD E35C   A DIRL01 JSR    INACIA    * GET THE INFO FROM ACIA
00394A E295 84 7F     A        ANDA   #$7F      * AND MASK AWAY PC (IF ANY)
00395A E297 A7 00     A        STAA   0,X       * STORE IT
00396A E299 08                 INX
00397A E29A BC A004   A        CPX    ENDA
00398A E29D 26 F3 E292         BNE    DIRL01    * AND CONTINUE UNTIL YOU REACH THE CEILING
00399A E29F 39                 RTS
00400                        *-------------------------------
00401A E2A0 86 51     A D2PUN  LDAA   #$51      * THIS IS THE NORMAL D2-PUNCHROUTINE
00402A E2A2 B7 8008   A        STAA   ACIAS
00403A E2A5 CE 01FF   A        LDX    #$01FF
00404A E2A8 8D 54 E2FE         BSR    PNLDR
00405A E2AA F6 A005   A PUND10 LDAB   ENDA+1
00406A E2AD F0 A003   A        SUBB   BEGA+1
00407A E2B0 B6 A004   A        LDAA   ENDA
00408A E2B3 B2 A002   A        SBCA   BEGA
00409A E2B6 27 02 E2BA         BEQ    PUND25
00410A E2B8 C6 FF     A        LDAB   #$FF
00411A E2BA 86 42     A PUND25 LDAA   #$42
00412A E2BC 8D 2D E2EB         BSR    OUTACI
00413A E2BE 37                 PSHB
00414A E2BF 30                 TSX
00415A E2C0 8D 36 E2F8         BSR    PUN
00416A E2C2 32                 PULA
00417A E2C3 4C                 INCA
00418A E2C4 B7 A00E   A        STAA   TEMP
00419A E2C7 CE A002   A        LDX    #BEGA
00420A E2CA 8D 2C E2F8         BSR    PUN
00421A E2CC 8D 2A E2F8         BSR    PUN
00422A E2CE FE A002   A        LDX    BEGA
00423A E2D1 8D 25 E2F8 PUND30  BSR    PUN
00424A E2D3 7A A00E   A        DEC    TEMP
00425A E2D6 26 F9 E2D1         BNE    PUND30
00426A E2D8 FF A002   A        STX    BEGA
00427A E2DB CE 0019   A        LDX    #$0019
00428A E2DE 8D 1E E2FE         BSR    PNLDR
00429A E2E0 FE A002   A        LDX    BEGA
00430A E2E3 09                 DEX
00431A E2E4 BC A004   A        CPX    ENDA
00432A E2E7 26 C1 E2AA         BNE    PUND10
00433A E2E9 86 47     A        LDAA   #$47
00434A E2EB 37           OUTACI PSHB
00435A E2EC F6 8008   A OUTC1  LDAB   ACIAS
00436A E2EF 57                 ASRB
00437A E2F0 57                 ASRB
00438A E2F1 24 F9 E2EC         BCC    OUTC1
00439A E2F3 B7 8009   A        STAA   ACIAD
00440A E2F6 33                 PULB
00441A E2F7 39                 RTS
00442A E2F8 A6 00     A PUN    LDAA   0,X
00443A E2FA 8D EF E2EB         BSR    OUTACI
00444A E2FC 08                 INX
00445A E2FD 39                 RTS
00446A E2FE 86 FF     A PNLDR  LDAA   #$FF
00447A E300 8D E9 E2EB         BSR    OUTACI
00448A E302 09                 DEX
00449A E303 26 F9 E2FE         BNE    PNLDR
00450A E305 39                 RTS
00451                        *-------------------------------
00452A E307                    ORG    $E307
00453                        ********** CHAR LENGTH =8,NO PARITY,1STOPBITS
00454A E307 86 14     A LOAD   LDAA   #$14      * 00010100 ; DIVIDE BY ONE, -..-
00455A E309 B7 8008   A        STAA   ACIAS     * THIS IS THE NORMAL MIKBUG-LOAD CONVERTED TO CASSETTE AND ACIA
00456                        *******LOAD LDA A #$3C *  YOU GET PRINTER INPUT IF YOU TAKE THESE TWO LINES INSTEAD
00457                        ******** STA A PIASB * AND CHANGE AT INACIO TO INCH INSTEAD OF INACIA
00458A E30C 8D 4B E359 LOAD3  BSR    INACIO
00459A E30E 84 7F     A        ANDA   #$7F
00460A E310 81 53     A        CMPA   #$53
00461A E312 26 F8 E30C         BNE    LOAD3
00462A E314 8D 43 E359         BSR    INACIO
00463A E316 84 7F     A        ANDA   #$7F
00464A E318 81 39     A        CMPA   #$39
00465A E31A 27 50 E36C         BEQ    LOAD22
```



SÄTTEN ÄR MÅNGA

forts.n.sida

```
00466A E31C 81 31     A        CMPA    #$31
00467A E31E 26 EC E30C         BNE     LOAD3
00468A E320 7F A00A   A        CLR     CKSM
00469A E323 8D 23 E348         BSR     BYTA
00470A E325 80 02     A        SUBA    #$02
00471A E327 B7 A00B   A        STAA    BYTECT
00472A E32A 8D 12 E33E         BSR     BODDR
00473A E32C 8D 1A E348 LOAD11  BSR     BYTA
00474A E32E 7A A00B   A        DEC     BYTECT
00475A E331 27 05 E338         BEQ     LOAD15
00476A E333 A7 00     A        STAA    0,X
00477A E335 08                 INX
00478A E336 20 F4 E32C         BRA     LOAD11
00479A E338 7C A00A   A LOAD15 INC     CKSM
00480A E33B 27 CF E30C         BEQ     LOAD3
00481A E33D 3F                 SWI                  * WILL PRINT OUT LAST ENTERED CELL
00482A E33E 8D 08 E348 BODDR   BSR     BYTA
00483A E340 B7 A00C   A        STAA    XHI
00484A E343 8D 03 E348         BSR     BYTA
00485A E345 7E E04E   A        JMP     BAD
00486A E348 8D 0A E354 BYTA    BSR     INHAX
00487A E34A 48                 ASLA
00488A E34B 48                 ASLA
00489A E34C 48                 ASLA
00490A E34D 48                 ASLA
00491A E34E 16                 TAB
00492A E34F 8D 03 E354         BSR     INHAX
00493A E351 7E E05E   A        JMP     BYT
00494A E354 8D 03 E359 INHAX   BSR     INACIO
00495A E356 7E E0AC   A        JMP     INHE
00496A E359 7E E35C   A INACIO JMP     INACIA   * HERE YOU CAN SWITCH BETWEEN INCH AND INACIA
00497A E35C B6 8008   A INACIA LDAA    ACIAS    * NEW INPUTROUTINE FROM ACIA
00498A E35F 47                 ASRA
00499A E360 24 FA E35C         BCC     INACIA   * UNTIL YOU GET REC FLAG
00500A E362 B6 8009   A        LDAA    ACIAD    * GET THE INFO FROM ACIA
00501A E365 39                 RTS
00503                  *-----------------------------------------
00504                  * HERE THE CONTROL-ORDER-SCANNING CONTINUES
00505                  *
00506A E366 C1 41     A CONTI  CMPB    #$41     * NEW ALFANUMERICAL PRINTOUT
00507A E368 26 05 E36F         BNE     CONT
00508A E36A 8D 60 E3CC         BSR     ALFAN
00509A E36C 7E E0E3   A LOAD22 JMP     CONTRL
00510A E36F C1 54     A CONT   CMPB    #$54     * NEW MEMORYTEST
00511A E371 27 66 E3D9         BEQ     MINTST
00512A E373 C1 49     A        CMPB    #$49
00513A E375 27 0A E381         BEQ     CON
00514A E377 C1 48     A        CMPB    #$48
00515A E379 26 0A E385         BNE     CO
00516A E37B CE A04A   A        LDX     #TEXT    * THE "H"-COMMAND PUTS SEARCH DATA WHERE THE SEARCH ROUTINE CAN
00517A E37E FF A002   A        STX     BEGA
00518A E381 8D 2D E3B0 CON     BSR     ALFIN    * NEW INPUTROUTINE FROM CONSOLE
00519A E383 20 E7 E36C         BRA     LOAD22
00520A E385 C1 4F     A CO     CMPB    #$4F
00521A E387 27 15 E39E         BEQ     OUT      * NEW D2-COMPATIBLE CASSETTETAPE OUTPUT
00522A E389 C1 42     A        CMPB    #$42
00523A E38B 27 16 E3A3         BEQ     BMV      * THE NEW BLOCKMOVE ROUTINE
00524A E38D C1 43     A        CMPB    #$43     * CONTINUE ROUTINE
00525A E38F 27 17 E3A8         BEQ     FORTS
00526A E391 C1 51     A        CMPB    #$51
00527A E393 27 16 E3AB         BEQ     DIRL     * THE DIRECT LOAD ROUTINE
00528A E395 C1 44     A        CMPB    #$44
00529A E397 26 D3 E36C         BNE     LOAD22
00530A E399 BD E1F8   A        JSR     D2LOAD   * THE NEW D2-LOADROUTINE
00531A E39C 20 CE E36C         BRA     LOAD22
00532                  *-----------------------------------------
00533A E39E BD E2A0   A OUT    JSR     D2PUN    * AS MOST ROUTINES ARE SUBROUTINES AND BECAUSE OF THE "LONG BRAN
00534A E3A1 20 C9 E36C         BRA     LOAD22   *  WE NEED THESE INTERMEDIATE STATIONS
00535A E3A3 BD E228   A BMV    JSR     BMOVE
00536A E3A6 20 C4 E36C         BRA     LOAD22
00537A E3A8 7E E279   A FORTS  JMP     FORTSA
00538A E3AB BD E28A   A DIRL   JSR     DIRLOA
00539A E3AE 20 BC E36C         BRA     LOAD22
00540                  *-----------------------------------
00541A E3B0 FE A002   A ALFIN  LDX     BEGA     * ALFANUMERICAL INPUT
00542A E3B3 BD E1AC   A ALFI1  JSR     INEEE    * GET CHARACTER
00543A E3B6 A7 00     A        STAA    0,X      * STORE IT
00544A E3B8 A1 00     A        CMPA    0,X      * CHECK THAT IT WAS CORRECTLY STORED
00545A E3BA 27 07 E3C3         BEQ     ALFI2    * IF SO
```

```
00546A E3BC 86 3F     A ALFI3. LDAA    #$3F      * OTHERWISE YOU PRINT A "?"
00547A E3BE BD E1D1   A        JSR     OUTEEE
00548                  ********** DEX***EARLIER YOU WOULD TELL WHERE THE FAULTY RAM WAS AND THEN PULL
00549                  *    THE EMERGENCY STOP
00550                  ********* LDA A #$04
00551                  ********* STA A 0,X
00552A E3C1 20 A9 E36C         BRA     LOAD22
00553                  ********* SWI
00554A E3C3 08         ALFI2   INX              * INCREASE THE INPUT POINTER
00555A E3C4 81 04     A        CMPA    #$04      * CHECK IF "EOT" = $04
00556A E3C6 26 EB E3B3         BNE     ALFI1
00557A E3C8 FF A004   A        STX     ENDA      * MAKES AN IMMEDIATE PRINTOUT POSSIBLE (OR CASSETTE OUTPUT)
00558A E3CB 39                 RTS
00559                  *---------------------------------------
00560A E3CC FE A002   A ALFAN  LDX     BEGA
00561A E3CF BD E07E   A        JSR     PDATA1    * ALFANUMERICAL PRINTOUT BY USING A MIKBUG SUBROUTINE
00562A E3D2 CE E19D   A        LDX     #MCL
00563A E3D5 BD E07E   A        JSR     PDATA1
00564A E3D8 39                 RTS
00565                  *-------------------------
00566A E3D9 BD E240   A MINTST JSR     MINTSO    * AN INTERMEDIATE STATION FOR MEMORYTESTSUBROUTINE
00567A E3DC 7E E0E3   A        JMP     CONTRL
00568                  *---------------------------------------
00569A E3DF A7 00     A CHA61  STAA    0,X       * A PATCH FROM MEMORY CHANGE ROUTINE OF MIKBUG
00570A E3E1 A1 00     A        CMPA    0,X
00571A E3E3 26 D7 E3BC         BNE     ALFI3
00572A E3E5 7E E087   A        JMP     CHA51
00573                  *-----------------------------------------
00574A E3E8 01                 NOP
00576A E3E9                    ORG     $E3E9
00577A E3E9 BF A008   A SFE    STS     SP        * SWIROUTINE MOVED TO HERE
00578A E3EC 30                 TSX
00579A E3ED 6D 00     A        TST     0,X
00580A E3EF 26 02 E3F3         BNE     *+4
00581A E3F1 6A 05     A        DEC     5,X
00582A E3F3 6A 00     A        DEC     6,X
00583A E3F5 7E E11F   A        JMP     PRINT
00584A E3F8                    ORG     $E3F8     * THE RESTART VECTORS
00585A E3F8     E000  A        FDB     IRQINT
00586A E3FA     E3E9  A        FDB     SFE
00587A E3FC     E3E9  A        FDB     SFE       * NMI IS USED FOR ABORTFUNCTION
00590                  * STACK MEMORY ASSIGNMENT
00591A A000                    ORG     $A000
00592A A000     0002  A IOV    RMB     2
00593A A002     0002  A BEGA   RMB     2
00594A A004     0002  A ENDA   RMB     2
00595A A006     0002  A NIO    RMB     2
00596A A008     0001  A SP     RMB     1
00597A A009     0001  A        RMB     1
00598A A00A     0001  A CKSM   RMB     1
00599A A00B     0001  A BYTECT RMB     1
00600A A00C     0001  A XHI    RMB     1
00601A A00D     0001  A XLOW   RMB     1
00602A A00E     0001  A TEMP   RMB     1
00603A A00F     0002  A TW     RMB     2
00604A A011     0001  A MCONT  RMB     1
00605A A012     0002  A XTEMP  RMB     2
00606A A014     002E  A STACK  RMB     46
00607A A042     0001  A STAKTO RMB     1
00608A A043     0001  A CCR    RMB     1
00609A A044     0001  A BREG   RMB     1
00610A A045     0001  A AREG   RMB     1
00611A A046     0001  A EXHI   RMB     1
00612A A047     0001  A EXLO   RMB     1
00613A A048     0001  A PCHI   RMB     1
00614A A049     0001  A PCLO   RMB     1
00615A A04A     0001  A TEXT   RMB     1         * THIS AREA EXTENDS FURTHER UP AGAINST A07F WHEN NEEDED
00616                  *-------------------------------
00617                          END
TOTAL ERRORS 00000

8009 ACIAD   00030*00439 00500
8008 ACIAS   00029*00045 00214 00312 00391 00402 00435 00455 00497
E3CC ALFAN   00508 00560*
E3B3 ALFI1   00542*00556
E3C3 ALFI2   00545 00554*
E3BC ALFI3   00546*00571
E3B0 ALFIN   00518 00541*
A045 AREG    00610*
E04E BAD     00080*00485
E047 BADDR   00077*00112
A002 BEGA    00052 00057 000[illegible] 00215 [illegible]323 00325 00326 00334 003[illegible] 00349 [illegible]359 00392 00406 00408 00419
```

FOR SEAR

forts.n.sida

```
                00422 00426 00429 00517 00541 00560 00593*
E1FD BILD       00313*00317 00332
E228 BMOVE      00334*00345 00535
E3A3 BMV        00523 00535*
E33E BODDR      00472 00482*
A044 BREG       00609*
E05E BYT        00090*00493
E348 BYTA       00469 00473 00482 00484 00486*
E055 BYTE       00077 00079 00083*00125
A00B BYTECT     00344 00471 00474 00599*
E132 C2         00201*00251
A043 CCR        00608*
E087 CHA51      00113*00124 00572
E3DF CHA61      00129 00569*
E085 CHANGE     00112*00122 00175
A00A CKSM       00092 00093 00468 00479 00598*
E385 CO         00515 00520*
E381 CON        00513 00518*
E36F CONT       00507 00510*
E366 CONTI      00188 00506*
E113 CONTIN     00181 00184*
E11B CONTIS     00185 00188*
E0E3 CONTRL     00076 00162*00201 00509 00567
E1F8 D2LOAD     00311*00530
E2A0 D2PUN      00401*00533
E1F3 DE         00271 00291 00307*
E1EF DEL        00272 00276 00282 00292 00301 00305*00306
E3AB DIRL       00527 00538*
E292 DIRLOI     00393*00398
E28A DIRLOA     00389*00538
A004 ENDA       00050 00063 00067 00217 00219 00249 00340 00343 00357 00372 00397 00405 00407 00431 00557
                00594*
A046 EXHI       00611*
A047 EXLO       00612*
E3A8 FORTS      00525 00537*
E279 FORTSA     00380*00537
E1AF INI        00268*00269 00285
E0BE INIHG      00138 00144*
E1BE IN3        00276*00281
E35C INACIA     00313 00319 00322 00324 00327 00393 00496 00497*00499
E359 INACIO     00458 00462 00494 00496*
E078 INCH       00105*00120 00134 00168
E1AC INEEE      00105 00266*00542
E354 INHAX      00486 00492 00494*
E0AC INHE       00135*00495
E0AA INHEX      00083 00089 00134*
E1EA IOS        00300 00302*
E1D4 IOUT       00289*
E1E3 IOUT2      00286 00298*
A000 IOV        00040 00592*
E000 IROINT     00040*00585
E307 LOAD       00173 00454*
E32C LOAD11     00473*00478
E338 LOAD15     00475 00479*
E044 LOAD21     00059 00076*00136 00140 00142
E30C LOAD22     00465 00509*00519 00529 00531 00534 00536 00539 00552
E30C LOAD3      00458*00461 00467 00480
E19D MCL        00113 00255*00562
E19C MCLOFF     00166 00254*
A011 MCONT      00226 00232 00604*
E240 MINTS0     00348*00566
E242 MINTS1     00349*00377
E248 MINTS2     00351*00358
E250 MINTS3     00353 00355*
E257 MINTS4     00359*
E25D MINTS5     00362*00373
E263 MINTS6     00364 00366*
E269 MINTS7     00367 00369*
E3D9 MINTST     00511 00566*
E134 MTAPE1     00204*00229
A006 NIO        00595*
E39E OUT        00521 00533*
E1DA OUT1       00292*00297
E0BF OUT2H      00145*00150 00151 00253
E0C1 OUT2HA     00146*
E0CA OUT2HS     00118 00151*00194 00195 00196
E0C8 OUT4HS     00116 00150*00197 00198 00200
E2EB OUTAC1     00412 00434*00443 00447
E2EC OUTC1      00435*00438
E075 OUTCH      00102 00104*00106 00153
E1D1 OUTEEE     00104 00287*00547
E067 OUTHL      00095*00146
E068 OUTHR      00099*00149
E0CC OUTS       00152*00170
A048 PCHI       00613*
A049 PCLO       00614*
```

```
E07E PDATA1 00108*00114 00167 00230 00561 00563
E07B PDATA2 00106*00110
8004 PIAD   00034*00156 00264
8006 PIADB  00032*00164
8005 PIAS   00033*
8007 PIASB  00031*00163
E2FE PNLDR  00404 00428 00446*00449
E11F PRINT  00177 00192*00583
E2F8 PUN    00415 00420 00421 00423 00442*
E148 PUN11  00217*00250
E15A PUN22  00221 00224*
E15C PUN23  00223 00225*
E17C PUN32  00238*00240
E13D PUNCH  00179 00213*
E2AA PUND10 00405*00432
E2BA PUND25 00409 00411*
E2D1 PUND30 00423*00425
E197 PUNT2  00233 00235 00236 00238 00245 00252*
E209 RDBLCK 00315 00319*
E1A5 SAV    00263*00267 00288
E013 SEAR2  00049*00064
E021 SEAR3  00054 00056*
E034 SEAR4  00062 00065*
E03A SEAR5  00068*00071
E010 SEARCH 00048*00073 00186
E3E9 SFE    00577*00586 00587
A008 SP     00155 00182 00192 00199 00380 00386 00577 00596*
A014 STACK  00606*
A042 STAKTO 00043 00165 00607*
E005 STAR   00043*00154
E0D3 STARK  00046 00155*
E0D0 START  00154*00588
E21D STBLCK 00327*00331
A00E TEMP   00228 00239 00418 00424 00602*
A04A TEXT   00048 00350 00360 00374 00516 00615*
A00F TW     00216 00218 00220 00234 00237 00241 00247 00603*
A00C XHI    00078 00081 00115 00117 00119 00483 00600*
A00D XLOW   00080 00601*
A012 XTEMP  00263 00302 00605*
```

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	48	5A	69	78	87	96	A5	B4	C3	D2	E1

MINNESTEST

Här har vi saxat ett litet program från tidningen Electronics. Alla förklaringar finns i texten varför inte mycket ytterligare behövs sägas. Det korta programet kan knappas in var som helst. Förslagsvis uppe på $A0XX någonstans. Eftersom rutinen kräver att X-registret innehåller den sista adress som ej skall testas, och accumulator A värdet $01 lägger vi ett par instruktioner före läget Start.

```
LDX  #$3000    CE 3000
LDAA #$01      86 01
```

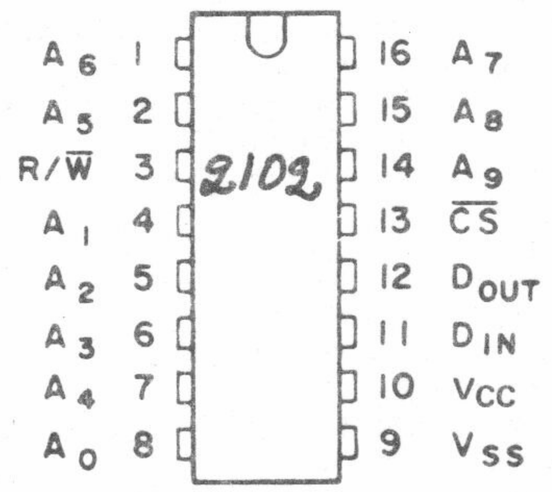
Detta exempel kommer att testa allt minne från $0000 t.o.m. $2FFF.

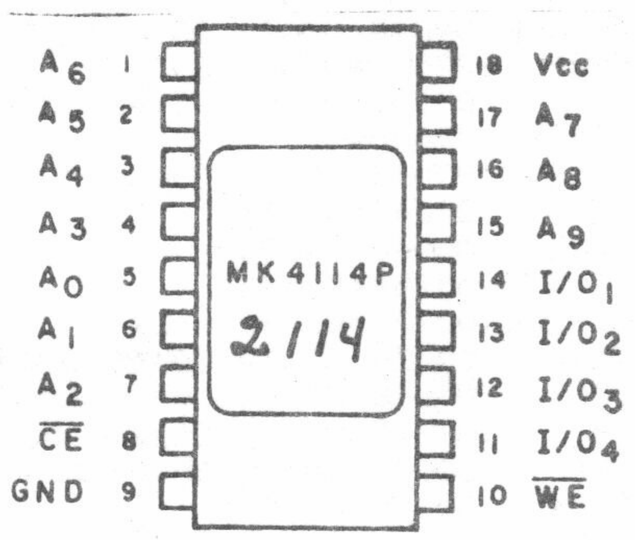
Observera att denna adress, i programmet kallad "Top" även ska knappas in vid :

```
CPX  =$Top     8C 3000
```

TB1

PIN CONNECTIONS





A quick march checks memory

by A. James Laurino
Wilmington, Del.

While forms of what is known as the march test are commonly used by chip manufacturers to check whether a random-access memory has its full complement of independent storage cells, users have no simple way of detecting RAM faults with a microprocessor. The 14-line version of the march test presented here provides an easy way to march memory to check for problems using the 6800 microprocessor. It can also be easily adapted for any other computer system.

This test will show whether each bit location in memory is capable of storing both 1s and 0s and whether storing any bit in a given location causes a change in the other locations. The test may not detect marginal failures, but there is no single, simple worst-case test for all systems.

The program makes two passes through the address range for each bit in the memory word (for example, 16 passes for an 8-bit machine). Test patterns are produced by shifting a single logic-1 bit through the accumulator to see if the bits within each word are independent, and that each bit location can store both 1s and 0s.

The first pass for each pattern stores that pattern in every location. On the second pass each location is read to see if there has been any change. If there has been no change, the program marks that location to show that it has been tested, by performing a logic complement operation. If one location responds to two addresses, then the complement pattern will found when the test reaches the second address affected.

The program is entered at START, which must be an address above the range to be tested. The address range extends from 0 to the value of Top − 1 (e.g. Top must be 1000 hex to test addresses 0 through 0FFF hex). Upon entry, accumulator A should contain 1, and the index register (X) should contain the value of Top.

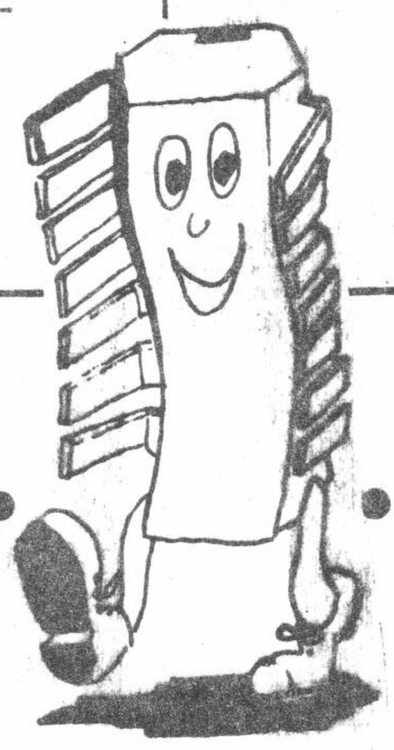
The locations in the address range to be tested are then filled with the first test pattern, generated by program lines 1 through 5. Each location is checked for a proper pattern and appropriately marked during lines 6 through 11. The next test pattern is generated and the test repeated until all patterns have been used (lines 12 and 13). At line 14, the test is concluded and the results reported.

If the memory is good, then register X will contain the value of Top and A will be zero. If a failure occurs, A will contain the failing pattern and X will have the failing address.

For example, consider the case where address line 5 (i.e., 2^5 bit) is stuck. The first failure would occur at address 20 hex, which would contain FE hex instead of the expected value of 1. □

6800 MEMORY CHECK PROGRAM

Label	Source statement				Comments
Start	DEX		09		Get the highest address to be tested
Fill	STA	A X	A7	00	Put the pattern in the current location
	DEX		09		Go to the next location
	BNE	Fill	26	FB	Continue through the address range
	STA	A X	A7	00	Put the pattern in the last location
Test	CMP	A X	A1	00	Check pattern in the current location
	BNE	End	26	0B	Stop if the pattern is not correct
	COM	X	63	00	Mark the location as tested
	INX		08		Go to the next location
	CPX	#Top	8C	-- --	Fill in the upper address limit here
	BNE	Test	26	F4	Continue through the address range
	ASL	A	48		Shift the one left to the next bit
	BCC	Start	24	E9	Repeat the test until all bits are used
End	SWI		3F		Return to monitor and display registers



NY KRETS

Motorola kommer med två nya intressanta kretsar för användande som klocka/kalender. De har beteckningarna MC146818 och MC 146819. Den senare är anpassad för anslutning till en mikroprocessorbuss. Den har därför fler pinnar (28) än den förra som har multiplexade pinnar för adr/data.

Kretsen innehåller 64 bytes RAM. Av dessa är 52 bytes lediga för användaren. Då kretsen har egen kristall för timingen och är gjord i CMOS teknik kan den drivas med ett batteri. Drar man då ur nätpluggen till den övriga datautrustningen fortsätter kretsen att fungera som vanligt. Genom att de 52 inre användar cellerna också överlever kan man däri lagra de händelsetidpunkter då något skall initieras. När processorn dras igång igen kan något program startas att göra jämförelser mot klockslag/datum mm.

Kretsen klarar veckodag (1-7), datum (30,31,28,29 etc.), månad (1-12) samt år (0-99). Vidare kan man få den att generera avbrotts signal för alarm tid:

- timme,min,sek jämförelse
- minut.sek jämförelse
- sek jämförelse
- timme jämförelse
- dag jämförelse

Man kan även få periodiska avbrott (fler än en per sekund):

- 15 olika hastigheter
- fyrkantvåg ut 4.19 MHz-1 sek
- 500 mS,250mS,125mS...30,5uS

Vidare kan man få:

- avbrottssignal till processor
- avbrott enablat av mjukvara
- avbrottsstatus tillgängligt för mjukvara

TB1

6800 FAMILJEN

Här kommer en lista på 6800 familjekretsar. Kretsar markerade med en ★ kommer under hösten 1979. Kretsar markerade med en → under våren 1980.

MC6800 CENTRALENHET	med 2 st 8-bitars ackumulatorer, 16 bits indexregister och stackpointer. Har 72 instruktioner med 195 operationskoder.
MC6801 ENCHIPSDATOR	med 2×8 bit alternativt 16 bit ackumulator, 16 bit indexregister och stackpointer. Har 82 instruktioner, 128 byte RAM, KLOCKGENERATOR, 2 K byte ROM, SERIELL PORT, 16 bit TIMER och max. 31 PARALLELLA PORTAR. Kan programeras för yttre adressering.
★ M6801E ENCHIPSDATOR	avsedd för flerprocessorsystem.
★ MC68701	Som 6801 där ROMdelen är ersatt av EPROM.
MC6802	Som 6800 men innehåller även klockgenerator och 128 byte RAM.
MC6803	Som 6801 men utan programminne.
MC6805 ENCHIPSDATOR	(28 pin) med 64 byte RAM, 1100 byte ROM, 20 parallella I/O, 8 plus 7 bit TIMER/DELARE och klockgenerator. MPU med en 8 bit ackumulator och indexregister samt 5 bit stackpointer. Har 58 insruktioner fördelade på ca. 200 operationskoder.
★ 146805 CENTRALENHET	i CMOS för direkt yttre adressering upp till 2 K byte. Har även 16 I/O, 112 bytes RAM och 1152 byte ROM.
MC6808	Som 6802 men utan RAM.
MC6809 CENTRALENHET	med 2×8 bit alternativt 16 bit ackumulator, dubbla 16 bit indexregister och stackpointer. Har 80 instuktioner med 348 operationskoder.
MCM 6810 RAM	128×8 bit statiskt.
★ 146818 KLOCKA	Realtidsklocka + RAM i CMOS. Multiplexad adress-databuss. 22 pin.
★ 146819 KLOCKA	Som 146818 men ej multiplexad. 28 pin.
MC6820 PIA	Programerbar Interface Adapter. För 2 8 parallella I/O med handskakningssignaler.
MC6821 PIA	Som 6820, men med bättre TTL anpassning på utgångarna.
MC6828 PIC	Priority Interupt Controller. Kan prioritera avbrottssignal inom 8 nivåer.

MCM6830 ROM	1024 8 bit.
MC6840 PTM	Programerbar Timer Module. Har 3 16 bits räknare för puls/tid räkning eller puls/frekvensgenerering.
MC6843 FDC	Floppy Disk Controller.
MC6844 DMAC	Direkt Memory Access Contoller. Kan snabbt överföra data mellan buss och I/O-krets. För 4 individuella kanaler.
MC6845 CRTC	CRT Controller. Programmerbar för alpha-numerisk eller grafisk presentation.
MC6846 COMBO CHIP	Innehåller 1/2 PIA, 1/3 PTM och 2 K byte ROM.
MC6847 VDG	Video Display generator. Tecken och synkgenerator med grafisk möjlighet. Ger svart vit eller färg signal för det Amerikanska NTSC systemet. (Never The Same Colour). Krånglig att anpassa till Europeisk TV standard.
→ MC6849 DDFC	Dubbel Density Floppy Controller.
MC6850 ACIA	Asynchronous Communications Interfase Adapter. För Terminal kommunikation upp till 500 Kbps.
↦ MC6851 UCIA	Kombinerad ACIA - SSDA.
MC6852 SSDA	Synchronous Serial Data Adapter. För Dataöver-föring upp till 600Kbps.
MC6854 ADLC	Advanced Data Link Controller. För seriell överföring av data med adress och datakontroll möjlighet. Programmerbar för SDLC, HDLC och ADCCP-snitt.
MC6860 MODEM	Modem för modulerad dataöverföring, max 600 bps.
MC6862	Digital Modulator för sändning av modulerad data upp till 2400 bit/sek.
M6871A HYBRIDKLOCKA	med 1 MHz eller 921,6 KHz frekvens. Ger klock-signaler för mikrodator samt styrsignaler för DMA, Dyn minnen, 2 processorsystem och bit rate generering.
M6871B HYBRIDKLOCKA	för D2-kit.
MC6875	Klockgenerator vars frekvens bestäms av yttre kristall eller RC-nät. Ger motsvarande signaler som 6871A samt power on reset.
MC6880=8T26	För överföring av 4 dubbelriktade signaler. Ger invertering.
MC6881	För överföring av 3 datasignaler i 3 riktningar.
MC6882	Octal Three State Buffer Latch.

MC6885=8T95 — För överföring av 6 enkelriktade signaler.

MC6886=8T96 — Som 6885. Ger invertering av signalerna.

MC6887=8T97 — För överföring av 6 enkelriktade signaler med 2 skilda Enable ingångar.

MC6888=8T98 — Som 6887. Ger invertering av signalerna.

MC6889=8T28 — Som 6880 men inverterar ej signalerna.

MC68488 GPIA — General Purpose Interface Adapter. För IEEE-standard (HP-bus). Kan programmeras för kontroll och överföring av data mellan mätinstrument.

B.Ljungblad

The INTELLEC 8 uses a seven-bit ASCII code, which is the normal 8 bit ASCII code with the parity (high order) bit always reset.

GRAPHIC OR CONTROL	ASCII (HEXADECIMAL)
NULL	00
SOM	01
EOA	02
EOM	03
EOT	04
WRU	05
RU	06
BELL	07
FE	08
H. Tab	09
Line Feed	0A
V. Tab	0B
Form	0C
Return	0D
SO	0E
SI	0F
DCO	10
X-On	11
Tape Aux. On	12
X-Off	13
Tape Aux. Off	14
Error	15
Sync	16
LEM	17
S0	18
S1	19
S2	1A
S3	1B
S4	1C
S5	1D
S6	1E
S7	1F

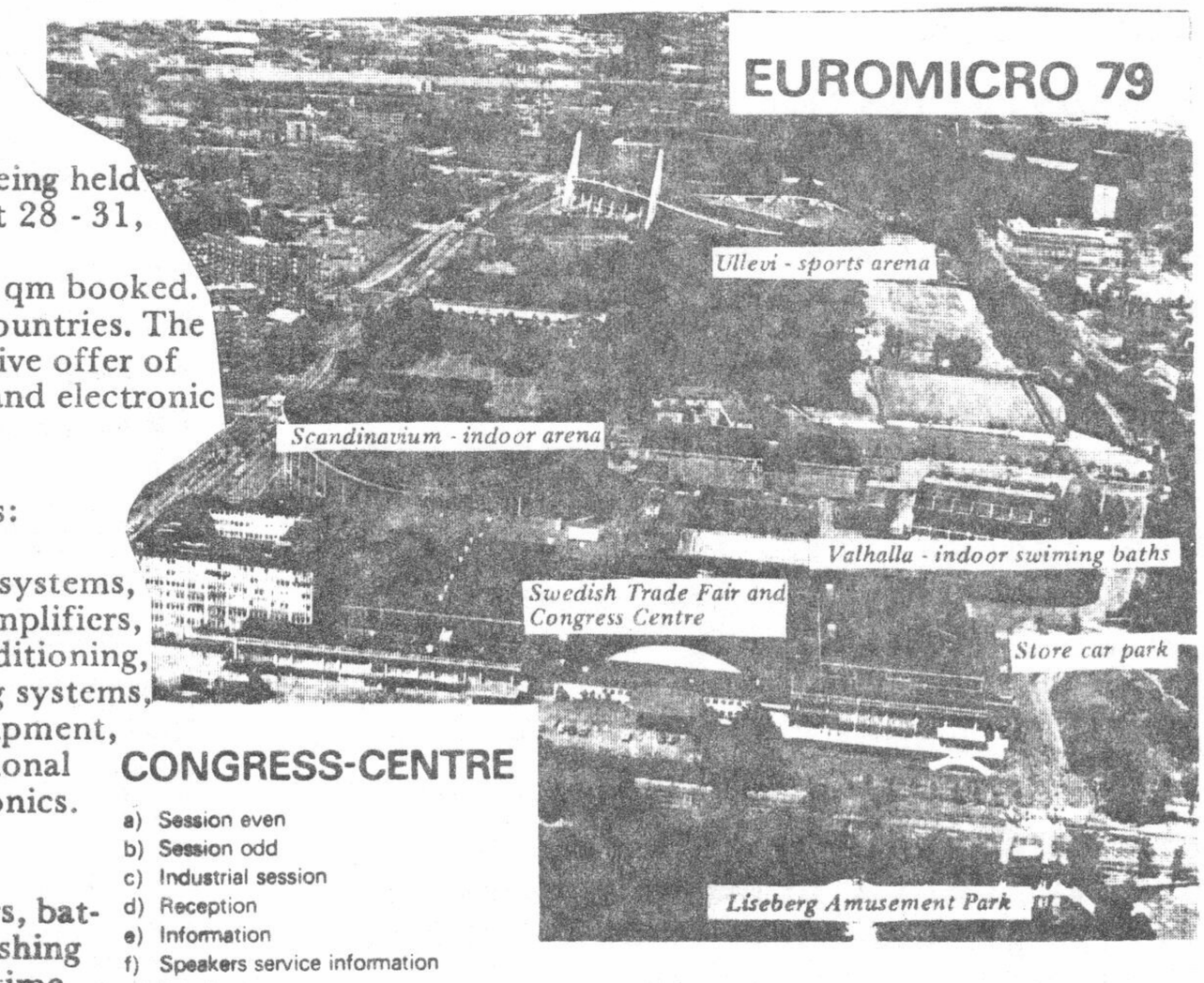
GRAPHIC OR CONTROL	ASCII (HEXADECIMAL)
ACK	7C
Alt. Mode	7D
Rubout	7F
!	21
"	22
#	23
$	24
%	25
&	26
'	27
(	28
)	29
*	2A
+	2B
,	2C
-	2D
.	2E
/	2F
:	3A
;	3B
<	3C
=	3D
>	3E
?	3F
[	5B
/	5C
]	5D
↑	5E
←	5F
@	40
blank	20
0	30

The new electronic center in Scandinavia

INDUSTRIAL ELECTRONICS 79
and COMPONENT 79
Two new international professional fairs are being held in Göteborg, Sweden during the period August 28 - 31, 1979.
Six months before the fairs opening are 6.000 qm booked. 900 companies will be represented from 21 countries. The fairs will give a complete picture of the extensive offer of companies which are in the field of electrical and electronic components and applied electronics.

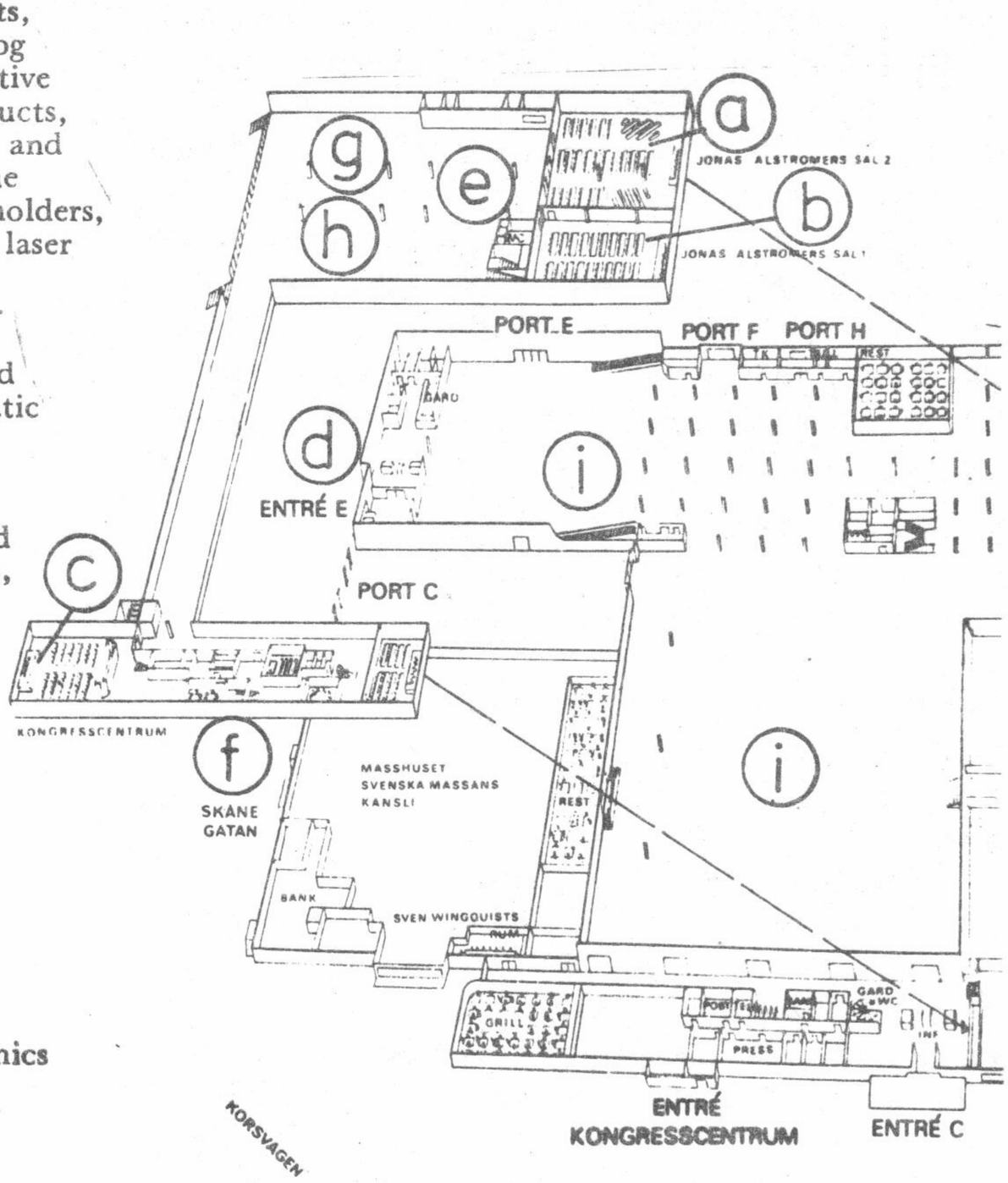
The fairs include the following product groups:
INDUSTRIAL ELECTRONICS
Medical and care electronics, electronic alarm systems, control systems, regulating control systems, amplifiers, audio/video, measuring equipment, signal conditioning, signal generation, data aquisition, data handling systems, radio equipment, optoelectronics, power equipment, electronic relays, service/maintenance, educational equipment, indicators, displays, special electronics.

COMPONENT
Connectors, cabinets, cases and racks, decoders, batteries, battery eliminators, battery chargers, lashing tape, computers, detectors, diodes, operating time meters, choke coils, attenuators, vacuum tubes, explosion-proof material, ferrites, fibre optics, filters, amplifiers, bushings, transducers, hydraulic components, holders, loudspeakers, indicators, instruments, analog and digital integrated circuits, insulating and protective materials, cables, leads and bus-bar systems, cable ducts, cable system material, cable terminals, clips, clamps and attachements, component handling, capacitors, time switches, power supplys units, printed circuit card holders, crystals, shrink hose, coolers, lamps, alarm systems, laser equipment, load cells,, rectifiers, readers, magnetic type, magnets, machines and tools, microwave components, micro-processors, memories, installation material, motors, resistors, marking material, printed circuit boards, converters, opto-electronics, pneumatic components, programmable systems, programming equipment, test equipment, knobs and handwheels, registering equipment, relays, counters, servo components, alarm panels, sirens, recorders, winding and coils, stabilizers, switches, buzzers, fuses, keyboards, thermostats, automatic timing equipment, timers, transformers, transistors, printing units, thyristors, scaling material, furnaces, tools, heat sources, overvoltage protector, overload protector.



CONGRESS-CENTRE

a) Session even
b) Session odd
c) Industrial session
d) Reception
e) Information
f) Speakers service information
g) Lunches
h) Coffee
i) Industrial exhibition



A comprehensive programme will be arranged in connection with the fairs. It will contain different aspects on the micro computer, how to use the electronics in the industry, further optoelectronics and a security conference with direction to electronics application.

SWEDISH TRADE FAIR FOUNDATION
Box 5222, S-402 24 Göteborg, Sweden.
Tel 031-20 00 00. Telex 20 600.

MC6809 MICROPROCESSOR

Med sitt 6809 CPU chip har MOTOROLA betydligt uppgraderat sin 6800 familj.

6809:s design medför viktiga egenskaper som hittills inte har funnits hos mikrodatorer. För dem som önskar använda den för generell databehandling i floppy disk miljö finns den fina egenskapen att kunna generera kod som är positions oberoende. Program som skapats på detta sätt kan laddas från disk och lagda var som helst i RAM utan hjälp av en " relocating loader ". M6809 förbättrar också användning av kompilatorer för högnivå språk. Exempelvis PASCAL som är ett modernt block strukturerat språk. Detta underlättas av de kraftigt utökade stack hanteringsmöjligheterna med upp till fyra stackar.

För instrument designern kommer M6809 att tillåta att program skrivs i PASCAL (eller andra högnivå, blockstrukturerade språk) och kunna kompileras till betydligt snabbare och effektivare maskinkod än tidigare. Detta är en konsekvens av den mängd egenskaper hos 6809 som direkt hanterar block strukturen hos moderna språk.

Instrument tillverkaren kommer också att uppleva en ny och unik industris uppkomst för att medverka i programutveckling för 6809 och som inte existerar för någon annan nu tillgänglig mikrodator. 6809 tillåter att kod skrivs för allmän distribution i form av ROM. Dessa kan sedan läggas på valfri adress av konstruktören utan att interferera med den egna mjukvaran. Tillverkare av periferiutrustningar kan erbjuda en avlusad, felfri driver-rutin för sin utrustning i en billig ROM kapsel för 6809.

På liknade sätt kan de som utvecklar generella mjukvaru paket (matte rutiner e. dyl.) sälja dessa som ROM för 6809 i vetskap att de inte kommer att ge orsak till "buggar" i annan mjukvara. Slutligen kan programerarens egen kod skrivas i modulär form med följd att vartefter mjukvaran växer, kommer interaktionen mella delarna att hållas i bra ordning. Detta blir konsekvensen av denna kodstruktur. De egenskaper hos 6809 som ger detta utbyte förklaras längre ned.

Filosofin som togs av konstruktörerna av 6809 var att uppnå en uppgradering av 6800 systemets kapacitet samtidigt som man bibehåller kompatibiliteten mot 6800 systemet. Konstruktörerna gjorde ett brilliant val i sättet att uppnå mjukvarukompatibilitet med 6800. I stället för att låsa sig till att bibehålla maskinkodsinstruktionerna valde man att bibehålla assembler språkets källkods definitioner. Därför kan nu varje assemblerkod som preparerats för 6800 köras igenom en 6809 assembler och producera kod som kan köras på 6809. Som konsekvens har 6809:ans instruktions set kraftfullt utökats, likafullt som kodningen av dessa instruktioner optimerats.

Registerstrukturen hos 6809 har utökats från 6800 till att inkludera följande:

- Ett andra, sexton bits index-register.
- Ett andra, sexton bits "användar" stackpekare register.
- Ett åtta bits direktminnes register. Kan användas för att direkt adressera valfri 256 bytes kaka i minnet.

Några av de mest framstående förstärkningarna i 6809:ans instruktions reportoar är:

- Ersättandet av 6800's index-adresserings möjlighet med en hel familj av adresserings-möjligheter. Dessa kan nyttja vilken som helst av 6809's fyra indexregister och stackpekare som en pekare till en operand

  eller som en pekare till en adress till en operand (s.k. indirekt adressering). Pekaren kan ha antingen ett fixt eller ett variabelt offset med tecken. Den kan (valfritt, men automatiskt) stegas upp eller ned. Till och med programpekaren kan användas för att nå operander eller adresser till operander genom användande av relativ adressering. Slutligen kan alla indexerade adress moder användas för att forma en adress som i sin tur kan laddas i valfri index eller stackpekar register.

- Sexton bits add, subtract, compare, load och store instruktioner för den sexton bits "D" accumulatorn (formad av A och B accarna). Plus förmågan att addera A,B eller D till valfri index/stackpekare. Vidare kan utbyte ske mellan valfritt CPU register av samma storlek. Detta utökar kraftigt förmågan att manipulera med pekare.

- En multipliceringsinstruktion som ger en sexton bits produkt av de två talen utan tecken i A och B accarna.

- Två bytes instruktioner som kan "pusha" resp "pulla" valfritt antal CPU register till eller från någon av stackarna.

- Tillägg av en "long branch" för hopp över hela programminnet med relativ adress.

Några av de förstärkningar som gjorts på 6809's hårdvara är:

- Inkluderandet av en klocka på chipet. Detta eliminerar den separata klock kretsen.

- Scmitt trigger på "Reset" ingången. Nu räcker en RC krets för att sparka igång systemet när spänningen slås på.

- Eliminerade av VMA (Valid Memory Address) och tillhörande grind nät för avkodning av det som hänger på bussen. 6809 ger i stället intrycket att läsa från adress $FFFF under sådan klockcykel då den inte använder adress-buss/ data-buss.

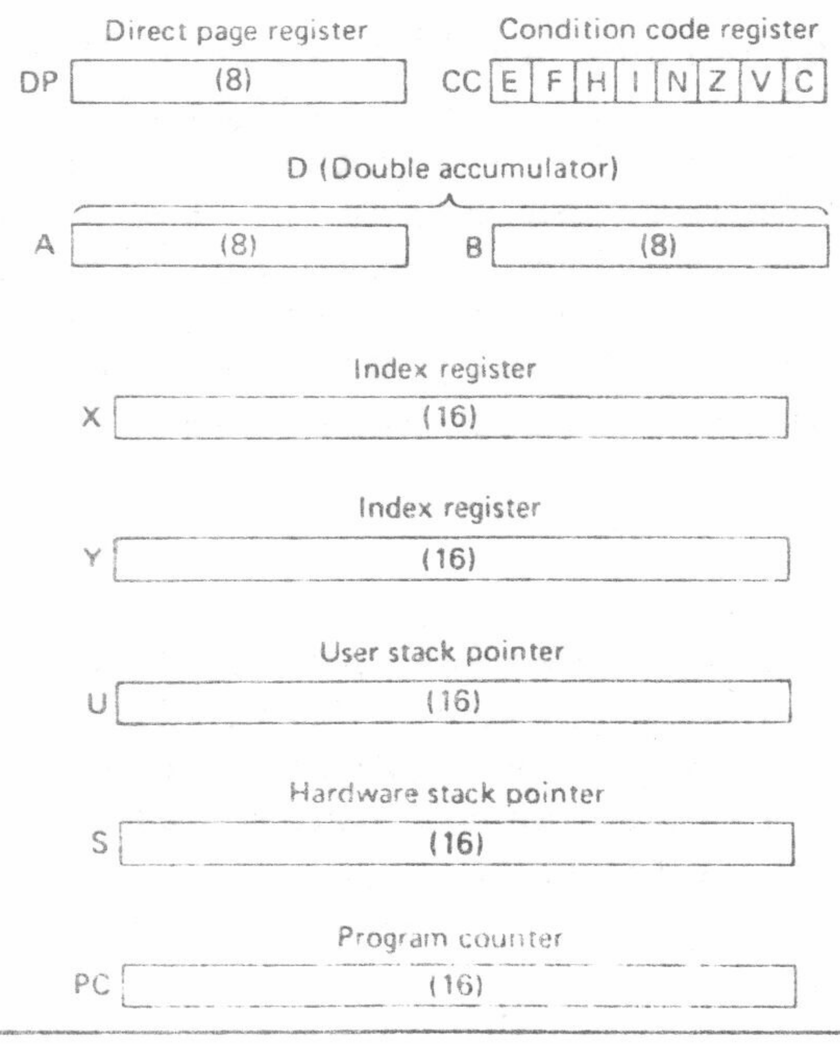
- Tre vektoriserade prioritets-interrupt nivåer. Varje disablar automatiskt lägre prioritets nivåer men lämnar högre nivåer enablade.

Registerstrukturen för 6809 visas i figur 1. De två högsta bitarna i statusregistret har tagits i bruk i 6809. E är en "entire" flagga som används automatiskt av RTI (Return from interrupt) instruktionen. Den gör att CPU kan skilja på en retur från det nya FIRQ (fast interrupt request) där endast statusregister och stackpekare har automatiskt sparats på stacken, och de vanliga NMI ocu IRQ där hela registeruppsättningen måste hämtas hem från stacken.

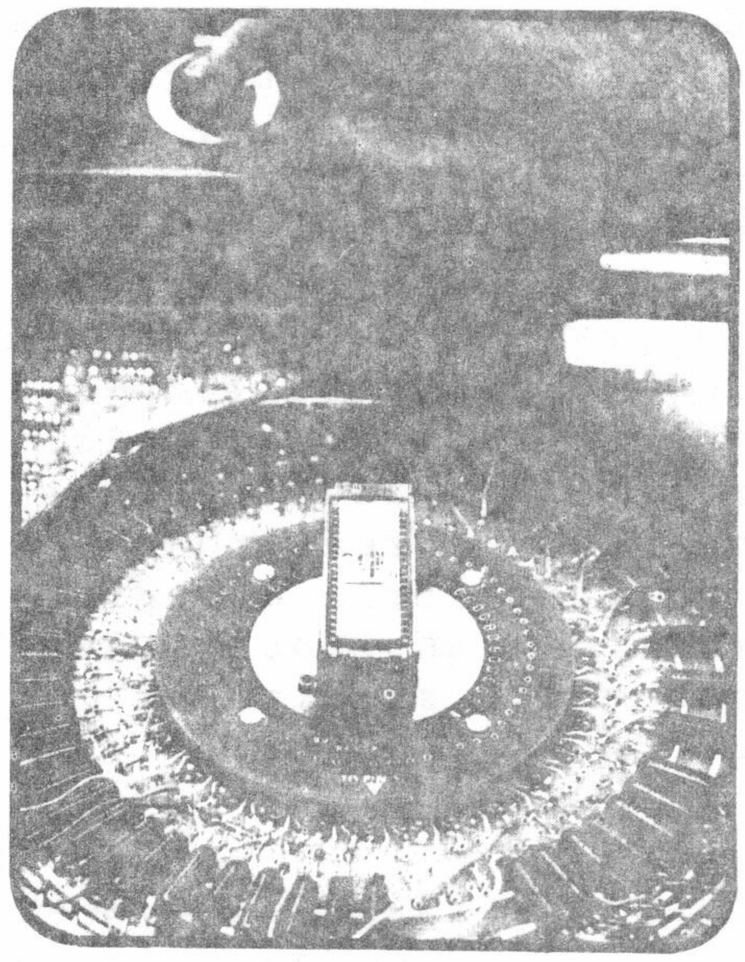
forts.n.sida

F biten är en mask för FIRQ avbrottet på samma sätt som I nu och i 6800 tjänstgör för IRQ. Är F nollställd och FIRQ ingången dras låg, initieras ett snabbt avbrott. E biten nollställs och programpekaren och statusordet stackas. Likaså sätts F och I maskerna vilket förhindrar nya FIRQ och IRQ avbrott. CPU'n vektoriserar till FIRQ avbrottets service rutin via adressen som är lagrad i $FFF6-FFF7.

Figure 1: Motorola 6809 CPU Registers



Om I flaggan är nollställd och IRQ ingången dras låg kommer ett IRQ avbrott att initieras. E biten sätts, alla CPU register (ej S) blåses ut på stacken samt I biten sätts. Därefter går CPU:n till service rutinen vars adress är lagrad i vektorn $FFF8-FFF9. Eftersom F flaggan ej automatiskt sätts av IRQ kan det högre prioriterade FIRQ nu avbryta IRQ rutinen.



NMI avbrottet har som tidigare den högsta prioriteten. En negativ flank på NMI ingången ger ovillkorligen ett NMI avbrott. Då sätts E , alla CPU register (ej S) stackas undan, I och F flaggorna sätts samt vektorisering sker via $FFFC-FFFD till NMI rutinen. Således kan 6809 utan extra hårdvara och utan extra instruktioner ge tre vektoriserade prioriterade avbrottsnivåer.

DP (direct page) registret i figur 1. definierar vilken sida (256 bytes) som kan adresseras med "direkt" mod. Om DP nollas, vilket också sker vid Reset, så blir 6809:s direkt-adressering lika med 6800:s direkt-adresseringsmod.

Några 6809 instruktioner jobbar med sextonbits dubbel accen D vilken är uppgjord av accumulatorerna A och B. A innehåller den övre (högst värda) byten medan B har den lägre.

De fyra sexton bits pekarna X,Y,U och S ger alla samma förmåga att accessera operander för alla instruktioner som tillåter den "indexerade" familjen av adressmoder. Tilläggas kan att de två stackpekarna U och S använder push och pull instruktioner för att stacka CPU register under programkontroll. Slutligen är det hårdvaru stackpekaren S som motsvarar 6800:s stackpekare och handhar adresser och CPU register under subrutinanrop och avbrott.

Instruktionsreportoaren för 6809 visas i fig 2. Medan CPU registret som är inblandad i en instruktion fått sitt namn utbrutet från mnemonicen och visat i REG kolumnen, förväntar sig däremot assemblerspråket att just CPU registrets namn kombineras med mnemonicen. Ett exempel klargör kanske:

```
CLRA                (ej CLR A)
STD  TEMP           (ej ST D TEMP)
ORCC #01            (ej OR CC #01)
ADDD ,X++           (ej ADD D ,X++)
```

New Figure 2: Motorola 6809 Instruction Set

Operation	MNE	REG	Description	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	Flags H I N Z V C
				← Bytes/Cycles →						
Move, 8-bit	TFR	R, R'	R' ← R } for R, R' = A, B, CC, DP	2/6						— — — — — —
	EXG	R, R'	R' ↔ R } for R, R' = A, B, CC, DP	2/7						— — — — — —
	CLR	A(B)	A(B) ← 0	1/2						— 0 1 0 0
	CLR		M ← 0			2/6	3/7	4/11	2+/6+	— 0 1 0 0
	LD	A(B)	A(B) ← M		2/2	2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
	ST	A(B)	M ← A(B)			2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
Move, 16-bit	TFR	R, R'	R' ← R } for R, R' = D, X, Y, U, S, PC	2/6						— — — — — —
	EXG	R, R'	R' ↔ R } for R, R' = D, X, Y, U, S, PC	2/7						— — — — — —
	LD	R	R ← M for R = D, X, U		3/3	2/5	3/6	4/10	2+/5+	— — ↕ ↕ 0
			for R = Y, S		4/4	3/6	4/7	5/11	3+/6+	— — ↕ ↕ 0
	ST	R	M ← R for R = D, X, U			2/5	3/6	4/10	2+/5+	— — ↕ ↕ 0
			for R = Y, S			3/6	4/7	5/11	3+/6+	— — ↕ ↕ 0
Load effective address	LEA	R	R ← Address of M for R = X, Y					4/9	2+/4+	— — ↕ — —
			for R = U, S					4/9	2+/4+	— — — — —
	PSHS	list	S stack ← List of desired CPU registers	2 (5 + b)						— — — — — —
	PSHU	list	U stack ← List of desired CPU registers	(b = number						— — — — — —
	PULS	list	List of desired CPU registers ← S stack	of bytes						(CC)
	PULU	list	List of desired CPU registers ← U stack	moved)						(CC)
Increment	INC	A(B)	A ← A + 1 (B ← B + 1)	1/2						— — ↕ ↕ ↕
	INC		M ← M + 1			2/6	3/7	4/11	2+/6+	— — ↕ ↕ ↕
Decrement	DEC	A(B)	A ← A − 1 (B ← B − 1)	1/2						— — ↕ ↕ ↕
	DEC		M ← M − 1			2/6	3/7	4/11	2+/6+	— — ↕ ↕ ↕
Complement	COM	A(B)	A ← $\bar{A}$ (B ← $\bar{B}$)	1/2						— 1 ↕ ↕ 0
	COM		M ← $\bar{M}$			2/6	3/7	4/11	2+/6+	— 1 ↕ ↕ 0
Set selected CC bits	OR	CC	CC ← CC OR M		2/3					↕ ↕ ↕ ↕ ↕
Clear selected CC bits	AND	CC	CC ← CC AND M		2/3					↕ ↕ ↕ ↕ ↕
Add, 8-bit	ADD	A(B)	A ← A + M (B ← B + M)		2/2	2/4	3/5	4/9	2+/4+	↕ ↕ ↕ ↕ ↕
Add with carry, 8-bit	ADC	A(B)	A ← A + M + C (B ← B + M + C)		2/2	2/4	3/5	4/9	2+/4+	↕ ↕ ↕ ↕ ↕
Decimal adjust acc. A	DAA		Correct BCD addition	1/2						— ↕ ↕ ↕ ↕
Add, 16-bit	ADD	D	D ← D + M		3/4	2/6	3/7	4/11	2+/6+	↕ ↕ ↕ ↕ ↕
	ABX		X ← X + B (unsigned addition of B to X)	1/3						— — — — —
Sign extend	SEX		A ← FF if bit 7 of B is set; 00 if bit 7 of B is clear	1/2						— — ↕ ↕ 0
Subtract, 8-bit	SUB	A(B)	A ← A − M (B ← B − M)		2/2	2/4	3/5	4/9	2+/4+	* ↕ ↕ ↕ ↕
Subtract with carry, 8-bit	SBC	A(B)	A ← A − M − C (B ← B − M − C)		2/2	2/4	3/5	4/9	2+/4+	* ↕ ↕ ↕ ↕

New Figure 2: Motorola 6809 Instruction Set (continued)

Operation	MNE	REG	Description	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	Flags H C Z N V
				← Bytes/Cycles →						
Negate, 8-bit	NEG	A(B)	A ⟵ 00 − A (B ⟵ 00 − B)	1/2						— ↕ ↕ ↕ ↕
	NEG		M ⟵ 00 − M			2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
Subtract, 16-bit	SUB	D	D ⟵ D − M		3/4	2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
Multiply	MUL		D ⟵ A*B (unsigned)	1/11						— b7 ↕ ↕ ↕
AND	AND	A(B)	A ⟵ A AND M (B ⟵ B AND M)		2/2	2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
Exclusive-OR	EOR	A(B)	A ⟵ A ⊖ M (B ⟵ B ⊖ M)		2/2	2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
OR (inclusive)	OR	A(B)	A ⟵ A OR M (B ⟵ B OR M)		2/2	2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
Compare, 8-bit	CMP	A(B)	A − M (B − M)		2/2	2/4	3/5	4/9	2+/4+	* ↕ ↕ ↕ ↕
	TST	A(B)	A − 00 (B − 00)	1/2						— — ↕ ↕ 0
	TST		M − 00			2/6	3/7	4/11	2+/6+	— — ↕ ↕ 0
	BIT	A(B)	A AND M (B AND M)		2/2	2/4	3/5	4/9	2+/4+	— — ↕ ↕ 0
Compare, 16-bit	CMP	R	R-M for R = X		3/4	2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
			for R = D, Y, U, S		4/5	3/7	4/8	5/12	3+/7+	— ↕ ↕ ↕ ↕
Rotate left	ROL	A(B)	C A, B, or M	1/2						— ↕ ↕ ↕ ↕
	ROL					2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
Rotate right	ROR	A(B)	C A, B, or M	1/2						— ↕ ↕ ↕ —
	ROR					2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ —
Arithmetic shift left	ASL	A(B)	C A, B, or M 0	1/2						— ↕ ↕ ↕ ↕
	ASL					2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
Arithmetic shift right	ASR	A(B)	C A, B, or M	1/2						— ↕ ↕ ↕ —
	ASR					2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ —
Logic shift left	LSL	A(B)	C A, B, or M 0	1/2						— ↕ ↕ ↕ ↕
	LSL					2/6	3/7	4/11	2+/6+	— ↕ ↕ ↕ ↕
Logic shift right	LSR	A(B)	C A, B, or M 0	1/2						— ↕ ↕ 0 —
	LSR					2/6	3/7	4/11	2+/6+	— ↕ ↕ 0 —
No operation	NOP		PC ⟵ PC + 1	1/2						— — — — —

Operation	MNE	Description	Inherent	Short relative	Long relative	Direct (Page zero)	Full	Full indirect	Indexed family	Flags H C Z N V
Jump unconditionally	JMP	PC ⟵ M				2/3	3/4	4/8	2+/3+	— — — — —
Branch unconditionally	BRA	PC ⟵ PC + M		2/3	3/5					— — — — —
if carry set	BCS	if C = 1		2/3	4/6(5)					— — — — —
if carry clear	BCC	if C = 0		2/3	4/6(5)					— — — — —
if equal zero	BEQ	if Z = 1		2/3	4/6(5)					— — — — —
if not equal zero	BNE	if Z = 0		2/3	4/6(5)					— — — — —
if minus	BMI	if N = 1		2/3	4/6(5)					— — — — —
if plus	BPL	if N = 0		2/3	4/6(5)					— — — — —
if overflow set	BVS	if V = 1		2/3	4/6(5)					— — — — —
if overflow clear	BVC	if V = 0		2/3	4/6(5)					— — — — —
if > zero (Signed numbers)	BGT	if Z + (N ⊕ V) = 0		2/3	4/6(5)					— — — — —
if ≥ zero (Signed numbers)	BGE	if N ⊕ V = 0		2/3	4/6(5)					— — — — —
if ≤ zero (Signed numbers)	BLE	if Z + (N ⊕ V) = 1		2/3	4/6(5)					— — — — —
if < zero (Signed numbers)	BLT	if N ⊕ V = 1		2/3	4/6(5)					— — — — —
if higher (Unsigned numbers)	BHI	if C or Z = 0		2/3	4/6(5)					— — — — —
if higher or same (Unsigned numbers)	BHS	if C = 0		2/3	4/6(5)					— — — — —
if lower or same (Unsigned numbers)	BLS	if C or Z = 1		2/3	4/6(5)					— — — — —
if lower (Unsigned numbers)	BLO	if C = 1		2/3	4/6(5)					— — — — —
NEVER	BRN	PC ⟵ PC + 2 (3)		2/3	4/5					— — — — —
Jump to subroutine	JSR	S stack ⟵ PC; PC ⟵ M				2/7	3/8	4/12	2+/7+	— — — — —
Branch to subroutine	BSR	S stack ⟵ PC; PC ⟵ PC + M (or use LBSR)		2/7	3/9					— — — — —
Return from subroutine	RTS	PC ⟵ S stack	1/5							— — — — —

Use LBRA, LBCS, LBCC, etc. as the mnemonic if the long relative branch is desired

Operation	MNE	Description	← Bytes Cycles → Inherent	Immediate	Flags H	C	Z	N	V
Software interrupt	SWI	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFFA, FFFB)	1/19		—	—	—	—	—
	SWI2	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFF4, FFF5)	2/20		—	—	—	—	—
	SWI3	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFF2, FFF3)	2/20		—	—	—	—	—
Clear and wait	CWAI	CC ⟵ CC AND M; S stack ⟵ PC, U, Y, X, DP, B, A, CC; Halt		2/20	↕	↕	↕	↕	↕
Synchronize I O	SYNC	Halt; Continue when interrupt occurs (see text)	1/2		—	—	—	—	—
Return from interrupt	RTI	CC ⟵ S stack; If E is set, then A, B, DP, X, Y, U, PC ⟵ S stack; If E is cleared, then PC ⟵ S stack	1/15(6)		↕	↕	↕	↕	↕

De fyra undantagen från denna regel är TFR, EXG, PSHS (PSHU) och PULS (PULU), exemplifierat nedan:

```
TFR  D,X    flytta D-acc till X-r
EXG  A,DP   A-acc till DP-reg och
            DP-reg till A-acc.
PULS A,X,Y  hämta A-acc, X-reg
            och Y-reg från stacken
```

6809 tillåter "full indirekt" adressering av en operand. Till exempel kan vi sätta upp så att när en programerbar timer ger ett avbrott, sker ett hopp till ett läge TIMR4 indirekt via ett adressregister. Ex:

```
LDX  #TIMR4   ladda x-reg med en
              adress med läge
              TIMR4.

STX  TIMER    lagra den i RAM
              adresserna RAM och
              RAM+1
```

Sedan när ett avbrott så småningom inträffar kan vi utföra följande i avbrottsrutinen:

```
JMP  [TIMER]  ladda programräk-
              naren med innehåll-
              et i RAM cellerna
              TIMER, TIMER+1.
```

De utökade nya adresseringsmöjligheterna som uppstått när man omdefinierat 6800:s indexerade adressmod till 6809:s familj av indexerade adressmoder framgår knappast av fig 2. En "bytes/cykler" av 2+/4+ anger minimum av bytes och klockcykler som åtgår för någon av dessa moder. Det exakta antal bytes och cykler som åtgår får man genom att addera värdena från fig 3 för den valda index adress variant som skall gälla till värdena i fig 2 för aktuell instruktion. Till exempel kan vi ta följande:

```
LDA  3,S
```

Instruktionen laddar accumulator A med fjärde byten på stacken. Detta är en två-bytes, fem-cykel instruktion. Innan vi går in på omfattningen av dessa indexerade adressmoder, skall vi klarlägga andra instruktioner listade i fig 2.

Ladda effektiv adress, LEA är en instruktion som tillåter alla 6809:s flexibla indexerings moder i bildandet av en pekare samt laddande av denna i någon av indexregistren eller stackpekarna. Exempel:

```
LEAX  3,S
```

Denna två-bytes instruktion laddar X registret med en pekare till den fjärde byten på stacken.

Instruktionerna för upp och ned stegning samt addition till pekare är följande:

```
LEAX  1,X   öka X (motsvarar gamla
            INX som ej 6809 har)

LEAX  -1,X  minska X (motsvarar i
            sin tur gamla DEX som
            ej finns)

LEAU  D,U   addera U-reg till D-
            reg och lägg resul-
            tatet i U-reg.
```

Likaså är LEAX B,X snarlik ABX. Skillnaden är att den förra behandlar B som ett två komplementtal med teckenbit (-128 till +127), medan ABX använder B som ett positivt tal mellan 0-255.

En stor del av 6809:s kraftfullhet ligger i de möjligheter som de indexerade adressmoderna ger. En assembler programerare kan dra mer eller mindre fördel av 6809 beroende på hur han förmår utnyttja dessa moder. För högnivåspråk användaren ser kompilatorn till att de blir så kraftfullt använda som möjligt.

Push och pull instruktionerna tillåter oss att spara undan och senare återta valfritt antal CPU register på någon av stackarna med användande av två-bytes instruktioner. Medan vi har full frihet i uppräknandet av de register vi vill spara undan, är ordningen i utläggningen på stacken en funktion av hårdvaran. I fig 4 ser vi ordningen i det fall samtliga register skickas ut på stacken. Det innebär att

```
PSHS  A,CC,X
```

och **forts.n.sida**

```
PSHS  A,X,CC
```

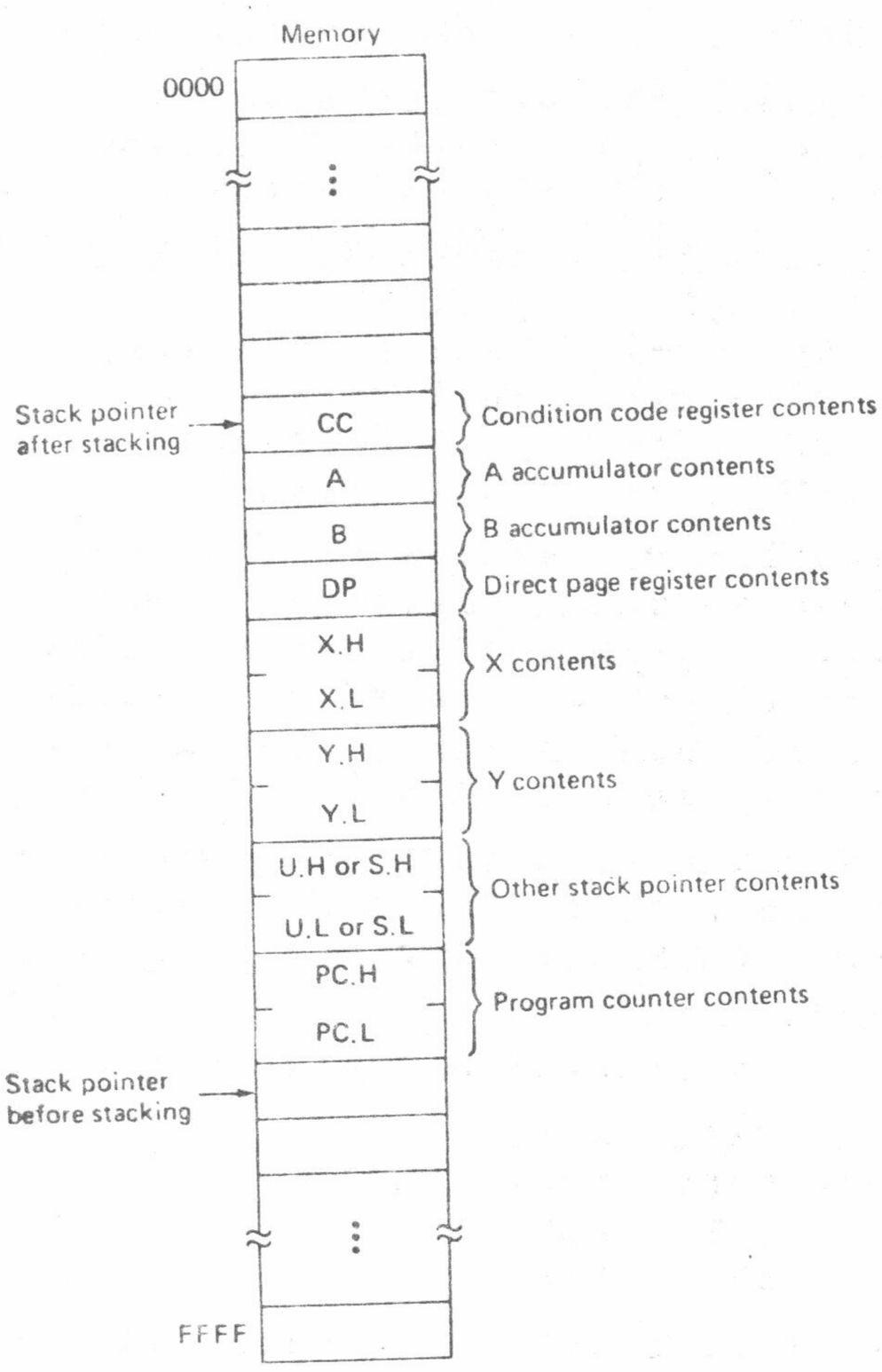
New Figure 3: Indexed Addressing Options

	Indexed addressing form	Source code form	Description (where R = X, Y, U, or S)	Additional bytes	Additional cycles
Non-indirect addressing	No offset, no change in R	,R	Get operand pointed to by R	0	0
	No offset, change R	,R+ ,R++ ,−R ,−−R	Get operand and then increment R once Get operand and then increment R twice Decrement R once and then get operand Decrement R twice and then get operand	0 0 0 0	+2 +3 +2 +3
	Fixed offset	offset, R	Temporarily add signed offset (expressed as sign plus magnitude) to R and then get operand, for −16 ≤ offset ≤ +15 −128 ≤ offset ≤ +127 −65536 ≤ offset ≤ +65536	 0 +1 +2	 +1 +1 +4
	Variable offset	A, R B, R D, R	Temporarily add signed offset (held in accumulator in 2s-complement form) to R and then get operand	0 0 0	+1 +1 +4
	Relative addressing	label, PCR	Use relative addressing to access a constant having a label − and located within ±128 bytes of the next instruction − and located anywhere	 +1 +2	 +1 +5
		offset, PC	Temporarily add signed offset (expressed as sign plus magnitude) to the address of the next instruction and then get operand, for −128 ≤ offset ≤ +127 −65536 ≤ offset ≤ +65536	 +1 +2	 +1 +5
Indirect addressing Use operand found as above as a pointer to the actual operand	No offset, no change in R	[,R]		0	+3
	No offset, change R	[,R++] [,−−R]		0 0	+6 +6
	Fixed offset	[offset, R]	−128 ≤ offset ≤ +127 −65536 ≤ offset ≤ +65536	+1 +2	+4 +7
	Variable offset	[A, R] [B, R] [D, R]		0 0 0	+4 +4 +7
	Relative addressing	[label, PCR]	For label located within ±128 bytes of the next instruction For label located anywhere	+1 +2	+4 +8
		[offset, PC]	−128 ≤ offset ≤ +127 −65536 ≤ offset ≤ +65536	+1 +2	+4 +8



kommer att först lägga X-reg, sedan A-accumulatorn samt slutligen CC (statusregistret) på hårdvarustacken (utpekad av S). Eftersom fyra bytes blir utlagda tar denna instruktion 4+4 cykler att utföra. Det motsvarar åtta mikrosekunder om en 1 MHz klockad 6809 används.

Figure 4: 6809 Stacking Order



Observera att operanderna som pushas ut på stacken hamnar på succesivt lägre adresser. Precis som i 6800. Emellertid kommer 6809:s stackpekare att peka på sista inlagda värde på stacken. I 6800 lämnades stackpekaren på första lediga adress. Även om detta är en skillnad i hårdvarans uppträdande, har de relaterade instruktionerna konstruerats att ge samma resultat. Tag till exempel 6800:s TSX instruktion. Den är identisk till 6809 instruktionen TFR S,X i så måtto att det resulterande innehållet i X efter operationen i båda fallen kommer att peka på toppen av stacken. Minns att 6800:s TSX laddar X med en adress som är ett högre än innehållet i stackpekaren.

För att sätta och nollställa valda bitar i statusordet CC, använder man i 6809 AND och OR immediate i stället för att ha separata instruktioner. Därför har 6800:s sex en-bytes instruktioner (SEC,CLC,SEV,CLV,SEI och CLI) ersatts av de två instruktionerna:

ORCC och ANDCC

Till exempel:

```
ORCC  #01
```

Den kommer att ettställa den i statusordet minst signifikanta biten. I fig 1 ser vi att det är carry flagga. Det motsvarar då 6800:s SEC instruktion. Konstruktörerna av 6809 tog dessa lågfrekvent använda instruktioner och omdefinierade deras kod för att lösgöra lite operationskoder för viktigare bruk.

De ovillkorliga instruktionerna JMP och JSR kan användas med vilken som helst av den indexerade familjens adressmoder. Det är nu till exempel möjligt att utföra en indirekt JMP eller JSR via en vektor tagen ur en tabell. Således, om X-registret innehåller bas adressen till en tabell med subrutinadresser, kommer:

JSR [B,X]

att hoppa till den subrutin vars adress är lagrad i tabellen på adresserna X+B och X+B+1. B står för accumulator B.

Varje av de villkorliga och ovillkorliga "branch" instruktionerna har nu två former av adressering. Den korta är 6800:s två-bytes instr. för hopp till (c:a) +- 128 steg från aktuellt programläge. En lång ovillkorlig (villkorlig) branch hoppar vart som helst, men kräver tre (fyra) bytes. Den långa branchen tar sex cykler att utföra när hopp utföres men bara fem annars.

forts.n.sida

cess. CPU:n kommer att förbli i stoppat läge tills ett avbrott inkommer. Om detta avbrott är enablat kommer CPU:n att agera som om den inte var i stoppat läge. Efter retur från avbrotts-rutinen har stopp statusen noll-ställts och instruktionerna fort-sätter att avverkas. Om däremot avbrottet är disablat kommer nu stoppstatusen att nollställas och programmet fortsätta utan att hoppa till avbrottsrutinen. Till exempel kan man tänka sig följande loop där man läser en data byte från en yttre krets varje gång denna ger ett "av - brott" - med avbrottskällan dis-ablad:

```
FAST  SYNC        vänta på avb.
      LDA  ,X     hämta data fr.
                  porten.
      STA  ,Y+    lagra data och
                  öka adressen.
      DECB        alla tecken?
      BNE  FAST   om nej, om
                  igen !
```

Förstärkt har också 6800:s SWI (programstyrda avbrott) blivit.

Två nya software interrupt har tillkommit. De kallas SWI2 och SWI3. Liksom SWI slänger de ut alla processorregister på stack SV De disablar IRQ och FIRQ samt vektoriserar via en adress uppe i FF sidan till en service rutin. Med dessa och med FIRQ har nu ad-resskartan fått tre nya ingångar uppe bland avbrottsvektorerna. Följande assemblersnutt tilldel-ar värden till dessa vektorer:

```
ORG  $FFF2

FDB  SWI3
FDB  SWI2
FDB  FIRQ
FDB  IRQ
FDB  SWI
FDB  NMI
FDB  RESTAR
```

Härav följer att exempelvis in-gångsinstruktionen i IRQ rutinen har programläget IRQ.

vändarstack och möjligheten att adressera operander inom de båda stackarna hjälper en kompilator för ett högnivåspråk att gå bra på 6809. I själva verket kan de båda indexregistren X och Y även användas som stackpekare. Ex:

```
STA  ,-X minska X, lagra A.
```

använder X-registret som stack-pekare och pushar accumulator A ut på denna stack. Likaså kan:

```
STB  ,-X    push  B
STD  ,--X   push  D (2 bytes)
STY  ,--X   push  Y-reg
STU  ,--X   push  U-stackp.
STS  ,--X   push  S-stackp.
```

användas för att pusha andra re-gister på "X" stacken. Motsvar-ande "pull" från "X" stacken är:

```
LDA  ,X+    hämta A, öka X-r.
LDB  ,X+    pull B
LDD  ,X++   pull D (2 bytes)
LDY  ,X++   pull Y-reg
LDU  ,X++   pull U-stackp.
LDS  ,X++   pull S-stackp
```

Naturligtvis skulle vi i verk-ligheten pulla registren i mot-satt ordning som vi pushade dem. Förmodligen skulle vi inte först stacka undan A och B separat och sedan stacka D (A+B).

Det kan vara värt att notera att endast de följande instruk-tionerna kräver två-bytes opera-tionskod:

villkorliga långa hopp

```
     SWI2  SWI3
CMPD CMPY  CMPS  CMPU
     LDY   LDS
     STY   STS
```

Den första byten, 10 eller 11 (hex) talar om för CPU:n att den skall titta på efterföljande, andra byte, för att avkoda vilken av dessa instruktioner den skall jobba med. Av den utökade 6809 instruktionsreportoaren har man nogsamt valt dessa till att bli de ineffektivt kodade instruk-tionerna. Av den anledningen att

de är minst frekventa i sådana applikationer som har små ROM baserade program. Alla andra instruktioner och alla de olika adressmoderna har fått effektiv kodning för denna miljö. Samtidigt är dessa ineffektivt avkodade instruktioner mycket användbara i stora floppy-disk baserade system. Där spelar lite extra minnesåtgång föga roll pga dessa två bytes operationskoder. Detta kontrasterar skarpt mot de Zilog Z80 instruktioner som använder av 8080 outnyttjade koder. Således kan man förvänta att en Z80 instruktion som använder indexerad adressering är två bytes (en för operationskod och en för indexet). I själva verket är alla Z80 instruktioner som använder indexerad adressmod antingen tre eller fyra bytes långa.

De indexerade adressoptions som syns i fig 3 är de INTRESSANTA nya finesserna med 6809. Speciellt då de kan tillämpas på hela trettioen av instruktinerna i tabell 2. Så fort en pekare är uppsatt, kan den användas med fast eller variabelt offset eller automatiskt upp/ned stegas. Vidare kan innehållet på en adress som är så utpekad, användas som operand eller som adress till en operand. Notera också att för de som inte har något behov av att sätta upp en separat stack med U-stackpekaren, finns tre generella pekare tillgängliga (X, Y samt U) för samtidig accessering av operander.

En tilltalande förmåga har man uppnått hos 6809 helt enkelt med att den fått samma indexerade möjligheter för maskinstackens pekare S som de övriga pekarna. Till exempel kan två bytes instruktionen:

```
LEAS  -8,S
```

användas som första instruktion i en subrutin. Den subtraherar åtta från stackpekaren. Detta sätter upp ett temporärt arbetsutrymme på stacken av åtta bytes. Dessa kan adresseras inom subrutinen med instruktioner som har operander 0,S till 7,S. Det fina med detta är att vi kan skapa utrymme för temporära variabler inom en subrutin och veta att detta utrymme kan användas utan att påverka andra variabler. Vidare behöver vi inte vara rädda att ett interrupt stör oss. Stackpekaren står så att det fixar sig automatiskt. När vi lämnar vår subrutin får vi inte glöma att frigöra de åtta byte vi tagit. Det gör vi med:

```
LEAS  8,S
RTS
```

Den adderar till åtta till stackpekaren igen.

Tidigare har vi sagt att en av de ovanliga och ovanligt kraftfulla egenskaperna hos 6809 är dess stöd för skapande av positionsoberoende kod. Den gör det på fem sätt:

1. Positions oberoende kontrollöverföring - med korta och långa hopp.
2. Positions oberoende temporärt lagringsutrymme - använder arbetsarea på stacken.
3. Positions oberoende access till konstanter inom den positions oberoende koden. Med användande av den indexerade adressmoden exemplifierat av följande: LDA CONST,PCR.
4. Positions oberoende access till tabeller inom den positions oberoende koden. Med användande av 'load effective address' instruktionen LEAX TABLE,PCR som laddar basadressen till en tabell in i X-reg följt av indexerade adresseringar i tabellen.
5. Positions oberoende access till konstanter och variabler utanför den positionsoberoende koden. Detta är viktigt för riktade applikationer som instrument konstruktion. Där kan maskinnära mjukvara utvecklas i ROM för massanvändning av olika projekt.

forts.n.sida

Ett exempel kan vara en printer tillverkare som även tillverkar en ROM preparerad för 6809 med en subrutin för drift av sin printer. Det kan då vara nödvändigt attssubrutinen får sig tillsänt ett antal parametrar. Om dessa parametrar är konstanter (I/O port adress, en skalfaktor för printer timing via CPU systemklockan) eller globala variabler (ex en "BUSY" flaggvariabel), räcker det att initiera en pekare (X,Y eller U) före anrop till "mass-use" ROM:en. Genom att organisera dessa parametrar i en tabell och ladda pekaren med basadressen till tabellen, kan parametrarna sedan packas upp i subrutinen med de indexerade adressmoderna. En "mass-use" ROM kräver exempelvis att vi skapar följande tabell:

```
RTBL FDB $1400  full adress till
                port 1.
     FDB $1402  full adress till
                port 2.
     FCB 25     8-bits tal för
                skalning av CPU
                klockan.
     FDB BUSY   full adress till
                variabeln BUSY.
```

Därefter kan ett anrop till ROM rutinen se ut som:

```
LDX #RTBL   X pekar på tabellen

JSR PRINT   till ROM rutinen.
```

PRINT subrutinen kan sedan ladda accumulator A med klock skalfaktorn med:

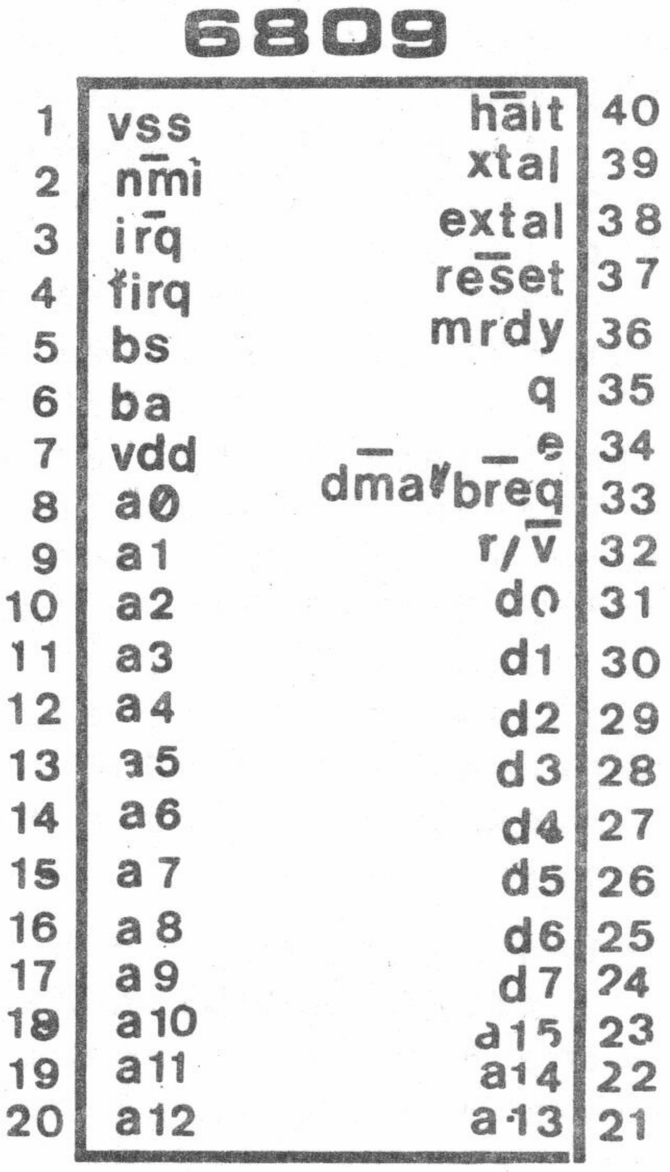
```
LDA  4,X
```

Den kan skriva till port 2 med:

```
STA  [2,X]
```

Den kan läsa den globala variabeln BUSY med:

```
LDA  [5,X]
```



Slutligen kan nämnas ett par grejor som rör RESET ingången på CPU:n. Den har fått en högre tröskelnivå än motsvarande ingångar på övriga 68XX kretsar. Konsekvensen blir att vid uppstartningen vet man att periferikretsarna är ordentligt nollställda och att den släppt när 6809 CPU snurrar igång. Inne i CPU:n nollställer RESET flaggorna I och F vilket disablar de maskbara avbrotten. Vidare nollställs direkt-side registret och sedan går CPU:n till den första instruktionen via vektorn på $FFFE och FFFF.

/anm/ överstående var (fritt) översatt och bearbetat från ett särtryck av John B. Peatman / McGraw-Hill Book Company av:

TB1

```
M6800-M6809 CROSS-ASSEMBLER  2.2


00099                          ********** VECTOR ADDITION / 16-BIT **********
00100                          *
00101                          *     PERFORM AN ELEMENT-BY-ELEMENT ADDITION ON
00102                          *     TWO VECTORS OF N 16-BIT ELEMENTS EACH.
00103                          *     PLACE THE RESULT IN A DIFFERENT VECTOR.
00104                          *     LET N BE 20.
00105                          *
00106                          *     SETUP:          3 LN, 10 BY,  10 CY
00107                          *     OPERATION:      5 LN, 11 BY, 32*20=640 CY
00108                          *     TOTAL:          8 LN, 21 BY, 650 CY
00109                          *
00110                          ***************************************************


00112 1077 8E   108E      3 ANBNCN LDX   #TABLEA
00113 107A 108E 10B6      4        LDY   #TABLEB
00114 107E CE   10DE      3        LDU   #TABLEC


00116 1081 EC   81        8 AN1    LDD   ,X++
00117 1083 E3   A1        9        ADDD  ,Y++
00118 1085 ED   C1        8        STD   ,U++
00119 1087 8C   10B6      4        CMPX  #2*20+TABLEA
00120 108A 26   F5        3        BNE   AN1


00122 108C 20   FE        3        BRA   *

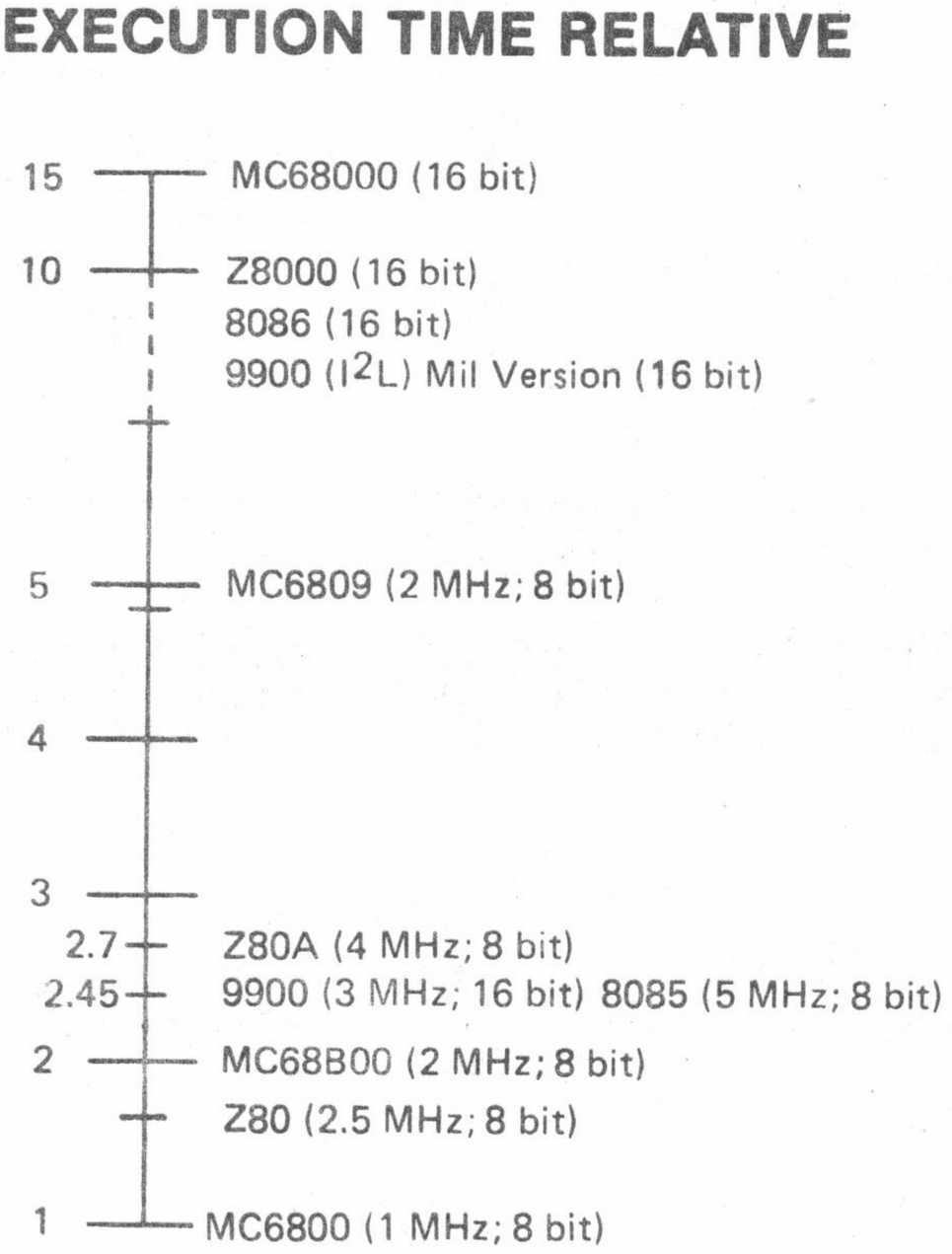


00124 108E      0000        TABLEA FDB   $00,$01,$02,$03,$04
00125 1098      0005               FDB   $05,$06,$07,$08,$09
00126 10A2      0010               FDB   $10,$11,$12,$13,$14
00127 10AC      0015               FDB   $15,$16,$17,$18,$19
00128 10B6      0099        TABLEB FDB   $99,$98,$97,$96,$95
00129 10C0      0094               FDB   $94,$93,$92,$91,$90
00130 10CA      0089               FDB   $89,$88,$87,$86,$85
00131 10D4      0084               FDB   $84,$83,$82,$81,$80
00132 10DE      0000        TABLEC FDB   0,,,,,,,,,,,,,,,,,,,,0
```

Fig 5. Program i 6809 kod. Ett av de tester i jämförelsen.

forts.n.sida

with all these improvements, how does the MC6809 benchmark against competition?



SUMMARIZED BENCHMARKS

Performance Criteria	MC6809	Z80A	MC6800	8085
Number of Instructions	*1.0	1.56	1.72	2.30
Number of Bytes	1.0	1.31	1.58	1.80
Number of Microseconds	1.0 (2 MHz)	1.80 (4 MHz)	2.40 (2 MHz)	2.20 (5 MHz)

*Normalized to 1.00 for the MC6809 – poorer performance has higher numbers.

Fig 6. Prestanda jämfört mellan olika mikroprocessorer.

relative execution times for eight benchmarks

		I/O Handler	Character Search	Computed Go To	Double Shift Right 5 Bit	Vector Addition 16-Bit Elements	Vector Addition 8-Bit Elements	16 × 16-Bit Multiplication	Move Block (64 Bytes)	Average Execution Time
6809	2.0 MHz	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	1.5 MHz	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	1.0 MHz	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
Z80	4.0 MHz	1.4	0.8	2.1	2.7	1.6	1.8	3.3	1.0	1.8
	2.5 MHz	2.2	1.2	3.4	4.4	2.6	2.9	5.2	1.6	2.9
9900	3.0 MHz	2.6	2.3	2.8	1.5	1.7	3.0	0.5	1.6	2.0
6800	2.0 MHz	0.9	1.4	1.9	1.3	3.1	2.8	5.0	3.3	2.4
	1.5 MHz	1.2	1.9	2.5	1.7	4.1	3.7	6.7	4.3	3.3
	1.0 MHz	1.8	2.8	3.7	2.5	6.1	5.5	10	6.5	4.9
8080+	3.0 MHz	1.9	1.8	2.8	6.1	2.3	2.7	9.6	2.4	3.7
8085	2.0 MHz	2.8	2.6	4.2	9.1	3.4	4.1	14.3	3.7	5.5

Fig 7. 6809 exekveringstid relativt andra mikroprocessorer. 6809 satt till 1.0 Högre värde - sämre prestanda.

actual execution times for eight benchmarks (μs)

		I/O Handler	Character Search	Computed Go To	Double Shift Right -- 5 Bits	Vector Addition 16-Bit Elements	Vector Addition 8-Bit Elements	16 / 16-Bit Multiplication	Move Block (64 Bytes)
6809	2.0 MHz	28	287.5	34.5	15	325	180	82	344.5
	1.5 MHz	37.3	383	46	20	433	240	109.3	459.3
	1.0 MHz	56	575	69	30	650	360	164	689
Z80	4.0 MHz	38.3	220.5	73.3	41	518	323	267	342
	2.5 MHz	61.3	352.8	117.2	65.6	828.8	516.8	427.2	547.6
9900	3.0 MHz	72	661	98	22	537	537	42	537
6800	2.0 MHz	24.5	404	64.5	19	993.5	498.5	409.5	1123.5
	1.5 MHz	32.7	539	86	25.3	1325	665	546	1498
	1.0 MHz	49	808	129	38	1987	997	819	2247
8080+	3.0 MHz	52.7	506.7	96.7	91.3	732	492	784	841
8085	2.0 MHz	79	760	145	137	1098	738	1176	1262

Fig 8. Exekveringstid uttryckt i µS för utförda jämförande tester.

SPELHÅLAN

FÖDELSEDAG

Här kommer ännu ett bidrag till våra spelprogram i BASIC. Det är överfört från en stor UNIVAC kärra och bearbetat för att passa våra 8k SWTPc Basic:s. Vem som är författare vet vi ej. Att han är amerikan och har initialerna RJC är klart.

Vad gör nu detta program. Jo det jobbar med dag/datum algorithmer. Först ber den om dagens fullständiga datum. Därefter talar det om vilken veckodag det är. Sedan räknar det ut ens födelse veckodag, hur många dagar gammal man är, hur många dagar man kommer att vara år 2000 etc. Däremellan har programmet förståss frågat efter hur gammal man är och datum/månad för födsel.

TB1

```
0001 REM THIS PROGRAM CORRECTED AS OF 1-23-68  RJC
0002 REM FIXAT FÖR M6800 BASIC AV TBL
0003 REM 1979-04-03
0004 PRINT
0005 PRINT
0006 PRINT "* * *   F Ö D E L S E D A G  * * *"
0007 PRINT
0010 DIM Z$(12),L(12),Y$(7)
0020 FOR I=1 TO 12
0030 READ Z$(I)
0035 NEXT I
0040 FOR I=1 TO 12
0042 READ L(I)
0044 NEXT I
0045 FOR I=1 TO 7
0046 READ Y$(I)
0048 NEXT I
0050 DATA JANUARI,FEBRUARI,MARS,APRIL,MAJ,JUNI,JULI,AUGUSTI
0052 DATA SEPTEMBER,OKTOBER,NOVEMBER,DECEMBER
0054 DATA 31,28,31,30,31,30,31,31,30,31,30,31
0056 DATA TORS,FRE,LÖR,SÖN,MÅN,TIS,ONS
0100 PRINT "ANGE DAGENS DATUM NUMERISKT;"
0102 PRINT "ÅR,MÅNAD,DATUM  EX: 1979,03,23"
0110 INPUT Z1,X1,Y1
0120  X9=X1
0122  Y9=Y1
0124  Z9=Z1
```

```
0126 GOSUB 900
0128  U=X8
0140 PRINT "IDAG ÄR DET ";Y$(U-7*INT(U/7)+1);"DAG"
0142 PRINT "( HOPPAS JAG.... )"
0205 PRINT
0220 PRINT "HEJ JAG ÄR M6800 DATORN."
0225 PRINT "VAD HETER DU";
0230 INPUT N$
0400 PRINT "HUR GAMMAL ÄR DU, ";N$;
0405 INPUT A
0406 REM ACCEPTERA 5-95 ÅR
0407  A=INT(A)
0410 IF INT((A-5)/95)=0  GO TO 425
0415 PRINT "SKÄRPNING, ";N$;","
0417 PRINT "DU DRIVER MED MIG■"
0420 GOTO 400
0425 PRINT "JAHA, DU ÄR ";A;"ÅR GAMMAL."
0430 PRINT "VET DU PÅ RAK ARM HUR"
0432 PRINT "MÅNGA DAGAR GAMMAL DU ÄR ( JA/NEJ )"
0435 GOSUB 800
0440 ON X7  GO TO 500,445,435
0445 PRINT "OK, SMARTING ,HUR MÅNGA ?"
0465 PRINT "TASKIGT ";N$;", DU FÅR"
0467 PRINT "ALLT VARA SNABBARE ÄN SÅ■■"
0470 GOTO 9999
0500 PRINT "OK, JAG SKALL BERÄTTA"
0505 PRINT "I VILKEN MÅNAD ÄR DU FÖDD"
0506 PRINT "( JANUARI,FEBRUARI ETC. )";
0510 INPUT M$
0515 FOR X=1 TO 12
0520 IF M$=Z$(X) GO TO 540
0525 NEXT X
0530 PRINT N$;", DU KANSKE ÄR TREVLIG,"
0532 PRINT "MEN DU STAVAR TASKIGT■■"
0535 GOTO 505
0540 PRINT "PÅ VILKEN DAG (1-31)";
0545 INPUT Y
0546 PRINT
0548  Z=Z1-A
0550 IF INT((Y-1)/L(X))=0 GO TO 600
0555 IF (X-1)*(Y-28)*(Z-4*INT(Z/4)+1)=1 GO TO 600
0560 PRINT "SKÄRPNING, ";N$;","
0562 PRINT "ANNARS ÅKER DU UT■"
0565 GOTO 540
0600 REM INHOPPS RAD
0601  X9=X1
0602  Y9=Y1
0603  Z9=Z
0605 GOSUB 900
0606  W=X8
0607  X9=X
0608  Y9=Y
0609  Z9=Z
0610 GOSUB 900
0611  V=X8
0612 IF W >= V GOTO 615
0613  Z=Z-1
```

forts.n.sida

```
0615 IF ABS(W-V)>30 GO TO 700
0620 ON SGN(W-V)+2 GO TO 630,650,640
0630 PRINT "DU FYLLER ÅR OM"
0632 PRINT "ENDAST ";V-W;" DAGAR."
0635 GOTO 700
0640 PRINT "DIN FÖDELSEDAG VAR FÖR BARA "
0642 PRINT W-V;" DAGAR SEDAN. GRATTIS■■■"
0645 GOTO 700
0650 PRINT "GRATTIS, KÄRA ";N$
0652 PRINT "GRATTIS PÅ FÖDELSEDAGEN■■"
0700  X9=X
0702  Y9=Y
0704  Z9=Z
0706 GOSUB 900
0708  V=X8
0710 PRINT "DU FÖDDES PÅ ";Y$(V-7*INT(V/7)+1);"DAGEN,"
0712 PRINT "DEN ";Y;" ";Z$(X);" ÅR ";Z
0715 PRINT "VILKET GÖR DIG ";U-V;"DAGAR GAMMAL"
0720 PRINT "DU FÖDDES ";V;" DAGAR EFTER KRISTUS,"
0722 PRINT "OCH DEN 1/1 ÅR 2000"
0725 PRINT "KOMMER DU ATT VARA"
0727 PRINT 730511-V;" DAGAR GAMMAL."
0730 PRINT "VAD SÄGS OM DET VA■■■"
0735 PRINT
0740 PRINT "TJA..., ":N$
0755 PRINT " DET HAR VARIT KUL ATT SNACKA MED DIG"
0756 PRINT "ÅTERKOM GÄRNA NÅGON DAG"
0760 PRINT "AJÖKEN"
0770 GOTO 9999

0800 INPUT A$
0805  X7=1
0810 IF A$="NEJ" GO TO 850
0820  X7 =2
0825 IF A$="JA" GO TO 850
0830  X7 =3
0835 PRINT "ETT ENKELT JA ELLER NEJ DUGER BRA■"
0850 RETURN
0900 REM
0902  X8=Y9-1
0905 FOR I1=1 TO X9-1
0910  X8=X8+L(I1)
0915 NEXT I1
0918  I1=INT(Z9/100)
0920 IF Z9<>4*INT(Z9/4) GO TO 935
0922 IF Z9/100=I1 GO TO 935
0925 IF X8>59 GO TO 935
0928 IF X9=3 GO TO 935
0930  X8=X8-1
0935  X8=X8+36524*I1+INT(365.25*(Z9-100*I1))
0950 RETURN

9999 END
```

OPkod på svenska

ADD A ACK.A Adderas binärt till minnet resultat i ACK A

ADD B ACK B Adderas binärt till minnet resultat i ACK B

ABA ACK A Adderas binärt till ACK B resultat i ACK A

ADC A ACK A Adderas binärt till minnet med föregående CARRY resultat i ACK A

ADC B ACK B Adderas binärt till minnet med föregående CARRY resultat i ACK B

AND A ACK A Logiskt och minnet resultat i ACK A

AND B ACK B Logiskt och minnet resultat i ACK B

BIT A Status Kod Registeret(CCR) som AND A men ACK A oförändrad

BIT B CCR som AND B men ACK B oförändrad

CLR Nollställ minne

CLR A Nollställ ACK A

CLR B Nollställ ACK B

CMP A JämförrACK A och minne(som SUB A men ACK A oförändrad)

CMP B Jämför ACK B och minne(som SUB B men ACK B oförändrad)

CBA Jämför ACK A och ACK B (som SBA men ACK A oförändrad)

COM Invertera minne

COM A Invertera ACK A

COM B Invertera ACK B

NEG Gör minne neg (2-kompl)

NEG A Gör ACK A neg (2-kompl)

NEG B gör ACK B neg (2-kompl)

DAA Decimaljustering av ACK A

DEC Minska minne med 1

DEC A Minska ACK A med 1

DEC B Minska ACK B med 1

EOR A Exclusive Or mellan minne och ACK A

EOR B Exclusive Or mellan minne och ACK B

INC Öka minne med 1

INC A Öka ACK A med 1

INC B Öka ACK B med 1

LDA A Minne överförs till ACK A

LDA B Minne överförs till ACK B

ORA A Inclusive Or mellan minne och ACK A

ORA B Inclusive Or mellan minne och ACK B

PSH A Lägg ACK A på stacken

PSH B Lägg ACK B på stacken

PUL A Hämta ACK A från stacken

PUL B Hämta ACK B från stacken

ROL Skifta minnet åt vänster,B7 - Carry,Carry -B0

ROL A Skifta ACK A åt vänster,B7 - Carry,Carry -B0

ROL B Skifta ACK B åt vänster,B7 - Carry,Carry - B0

ROR Skifta minnet åt höger,B0 - Carry,Carry - B7

ROR A Skifta ACK A åt höger,B0 - Carry,Carry - B7

ROR B Skifta ACK B åt höger.B0 - Carry,Carry -B7

ASL Skifta minne åt vänster,B7 - Carry,00 - B0

ASL A Skifta ACK A åt vänster,B7 - Carry,00 - B0

ASL B Skifta ACK B åt vänster,B7 - Carry,00 -Bo

ASR Skifta minne åt höger,B7 - B7,B0 -Carry

ASR A Skifta ACK A åt höger,B7 -B7,B0 - Carry

ASR B Skifta ACK B åt höger,B7 - B7,B0 - Carry

LSR Skifta minne åt höger,00 - B7,B0 -Carry

LSR A Skifta ACK A åt höger,00 -B7,B0 - Carry

LSR B Skifta ACK B åt höger,00 - B7,B0 - Carry

STA A Lagra ACK A i minnet

STA B Lagra ACK B i minnet

SUB A Minska ACK A med innehållet i minne

SUB B Minska ACK B med innehållet i månne

SBA Minska innehållet i ACK A med innehållet i ACK B

SBC A Minska innehållet i ACK A med innehållet i minne och med föreg Carry

SBC B Minska innehållet i ACK B med innehållet i minne och med föreg Carry

TAB Överför ACK A till ACK B

TBA Överför ACK B till ACK A

TST Sätt CCR för minne minus 00

TST A Sätt CCR för ACK A minus 00

TST B Sätt CCR för ACK B minus 00

CPX Jämför Indexregister (IX) med minne

DEX Minska IX med 1

DES Minska stackpekaren med 1

INX Öka IX med 1

INS Öka stackpekaren med 1

LDX Hämta IX från minne

LDS Hämta stackpekaren från minne

STX Lagra IX i minne

STS Lagra stackpekaren i minne

TXS IX överförs till stackpekaren och minskas med 1

TSX Stackpekaren överförs till IX och ökas med 1

BRA Hoppa vist antal steg framåt eller bakåt

BCC Förgrening om Carryn ej satt

BCS FÖrgrening om Carryn satt

BEQ Förgrening om nollbiten satt i CCR

BGE Förgrening om varken negbiten eller överflödsbiten satt i CCR

BGT Förgrening om varken nollbiten,negbiten eller överflödsbiten satt

BHI Förgrening om varken Carry eller nollbiten satta

BLE Förgrening on nollbiten eller(negbiten eller överflöd)satt

BLS Förgrening om nollbiten eller Carry satt

BLT Förgrening om negbiten eller överflöd satt(dock inte om båda satt)

BMI Förgrening om negbiten satt

BNE Förgrening om nollbiten ej satt

BVC Förgrening om överflöd ej satt

BVS Förgrening om överflöd satt

BPL Förgrening om negbiten ej satt

BSR Förgrening till subrutin(återhoppsadress på stacken)

JMP Ovillkorligt hopp

JSR Hopp till subrutin

NOP Gå till nästa OPkod

RTS Återvänd från subrutin(programräknaren från stacken)

RTI Återvänd från avbrottsrutin(A-,B-,IX-,CCR och prog från stacken)

SWI Mjukvaruavbrott(A-,B-,IX-,CCR-,och program till stacken)Programräknaren (PC) sätts till början av mjukvaruavbrottsrutinen

WAI Lagra undan A-,B-,IX-,CCR och PC och vänta på avbrott

CLC Nollställ Carry

CLI Nollställ avbrottsmasken

CLV Nollställ överflöd

SEC Sätt Carry

SEI Sätt avbrottsmasken

SEV Sätt överflöd

TAP ACK A överförs till CCR

TPA CCR överförs till ACK A

C-A BRUNO

Bubbelminnen

Bubbelminnena kommer närmast att konkurrera med sekundärminnen t ex floppy. Förutsättningarna att få ned priset vid en storskaletillverkning förefaller goda, men än så länge är det där man ej kan ge förebyggande underhåll eller där man ej kan acceptera fel som bubbelminnen används.

Pris- accesstidsmässigt ligger bubbelminnena f.n. mellan halvledarminnen och skivminnen.Halvledarminnenas ställning som primärminne är ej hotad då bubbelminnena har väsentligt för lång accesstid.

Eftersom bubblorna är icke-flyktiga kommer vi säkert snart att kunna köpa bubbelminneskort med 8 minneskapslar innehållande bl a operativsystem och som skall ersätta floppydisksystem. Redan idag finns kommersiella kapslar som rymmer 250 kbit/kapsel och på labstadiet har man 1 Mbit/kapsel.

Man kan inte byta kort lika lätt och billigt som man byter en diskett, så vi skall väl vänta med att dödförklara floppyn.

Vad är då ett bubbelminne och hur ansluter jag det till mitt system? Det fungerar p g a att vissa magnetiska ämnen (exempelvis oxider av järn blandat med t ex yttrium,samarium eller lutetium) inom sig har områden (domain) med riktade magnetfält, trots att ämnet som helhet är omagnetiskt. När man utsätter det för ett yttre magnetfält ökar andelen av den typ på område vars magnetfält är lika riktat som det yttre magnetfältet på bekostnad av de motriktade områdena, som minskar till cylindriska öar, s.k. bubblor, innan de försvinner helt.Dessa bubblor är som små lättrörliga stavmagneter, observera dock att ingen materialtransport sker(jmf. vågor på sjön).

Själva bubbelminnet består av en yta av sådana magnetiska ämnen,som man avsatt på ett omagnetiskt substrat med liknande kristallegenskaper. Ovanpå denna yta har man dessutom lagt ett magnetiskt mjukt material, som är lätt påverkat av yttre magnetiska fält.Detta har etsats så att enbart omväxlande T-n och I-n kvarstår, se figur 1. Även andra mönster förekommer. Vinkelrätt mot ytan ligger ett permanent magnetfält (bias),som är avpassat så att bubblorna är stabila och lagom stora.

I ytans plan ligger två magnetspolar som matas med 90^{o} fasförskjutem växelström med ca 50 till 300 kHz frekvens.Det alstrade roterande fältet gör att bubblan klockas genom mönstret. Om bubblans överdel är en sydpol dras den mot närmaste nordpol, som flyttar sig ett steg i taget i takt med det roterande fältet och därvid drar bubblan med sig. Slutar man klocka förblir bubblorna

stabila till dess att man klockar igen. Enda kravet är att biasfältet inte upphör, men det är säkerställt av permanentmagneter, så minnet har ej krav på konstant energitillförsel för att behålla minnet.

Ettor och nollor består av bubbla resp. icke bubbla. Nya bubblor genereras av en miniatyrspole i ena änden på det långa skiftregister som man kan likna bubbelminnet vid. I slutet på skiftregistret sitter t ex magnetoresistiva (ändrar motstånd med ändringar i magnetfältet) motstånd i en bryggkoppling. Dessutom finns en "kopierare" (replicator) som automatiskt återför innehållet till skiftregistrets början.

Denna typ på bubbelminne med en enda lång slinga får dock en lång accesstid (250 kbit X 10 us = 2500 ms max,eller 1,25 sek i medel) Man kan minska accesstiden genom att dela upp skiftregistret i flera parallella korta återkopplade skiftregister och med ytterligare skiftregister för serie/parallell- och parallell/serieomvandlingen, allt utfört som "bubbelregister". Dessa bubbelminnen ställer större krav på kontrollkretsarna, samtidigt som de medger att en del skiftregister får vara trasiga. Vilka som är trasiga finns angivet i ett extra skiftregister. Dessutom finns ofta ett register med en kod som ger möjlighet till initialsynkronisering (möjliggör adressering).

I dessa bubbelminnen är accesstiden ca 2 ms, vilket är ca 50 gånger snabbare än för en floppy. Denna typ på bubbelminnen anpassas till datorsystemet med hjälp av specialkretsar som innehåller matning till spolarna, timingsignaler till in- och utmatning,läsförstärkare, in- och utlatchar,initialsynk, kretsar för uteslutning av felaktiga skiftregister med mera.

Priset för kort accesstid är alltså en krångligare kontrollogik, som oftast består av specialkretsar, som medger såväl ackumulator- som DMA-överföring av data till och från minnet.

Denna artikel bygger på ett par engelska tidningsartiklar, som Du kan få låna om Du skriver till Carl Axel Bruno,Timotejv 116, 191 77 SOLLENTUNA. Där kan Du till och med få tips om hur du tillverkar och testar Dina egna minneskapslar i Ditt eget halvledarlab.

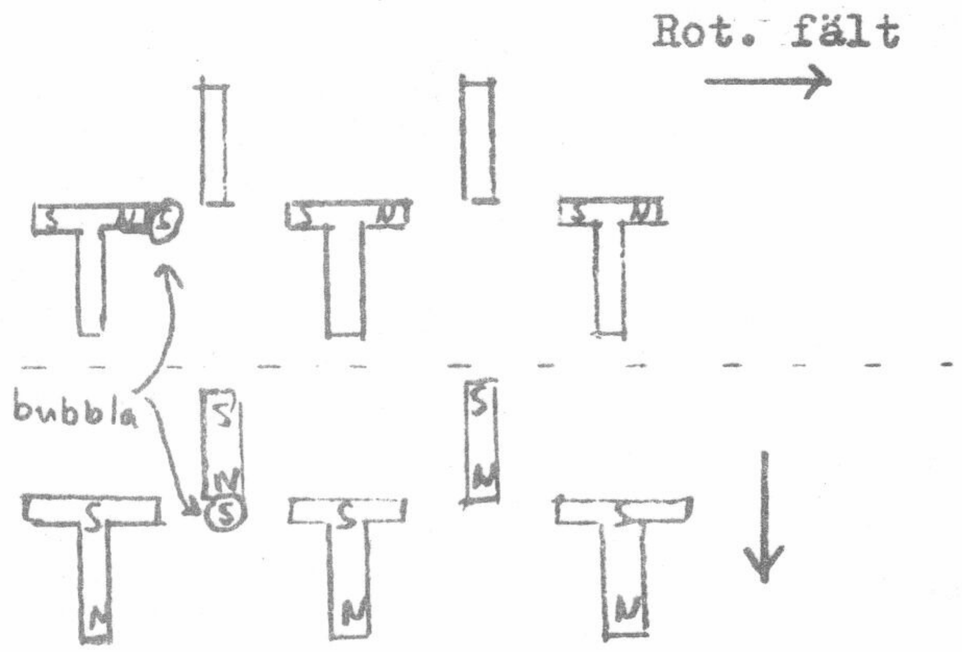


fig. 1

User Group Library

I detta nummer av MPU-laren har vi förmånen att kunna visa två fina program ur Motorola's User Group Library. Vi kan också visa en förteckning över de program som finns i biblioteket. Vad är då User Library för ett konstigt djur ? I en av pärmarna kan vi läsa följande (översatt):

- Motorola's User Group Library ställer till förfogande en samling generella program och rutiner utvecklade för M6800. Dessa program förser mikroprocessor systemkonstruktören med verktyg för att reducera systemets utvecklingstid.

Medlemskap

Användare kan få medlemskap i UGL på ett av följande sätt:

1. Medlemsavgift på $150.
2. Bidraga med ett acceptabelt 6800 program/rutin.

Medlemskap ger användaren pärmar med källkodslistor av de flesta program. Det ger också rätt till alla uppdateringar till UGL under ett år.

Förnyelse av medlemskap

Användaren kan förlänga sitt medlemskap i UGL genom:

1. Betala förnyad avgift på $80.
2. Bidraga med ett acceptabelt 6800 program någon gång under den aktiva medlemsperioden.

Det förnyade medlemskapet ger användaren alla uppdateringar till UGL under nästa år.

När medlemskapet står i begrepp att upphöra blir medlemmen informerad därom i ett brev.

User Group mailing address

All korrespondens som berör UGL skall adresseras till:

M6800 User Group Library
Motorola, Inc.
Microsystems
16, Chemin de la Voie-Creuse
1211 - Geneva -
SCHWEIZ

Som medlem kan man köpa källkod på program i form av diskett, kassett eller papperstape. Att köpa kassett/diskett är mest prisvärt. Papperstape är tydligen dyrt i hanteringen. En diskett med c:a 20 program kostar $50. Den köps från den lokale distributören. I vårt fall är det Interelko.

I programförteckningen kanske du hittar något vettigt program av allmänt intresse. Låt oss då veta detta. Sedan får vi se vad som kan göras. Vi måste begära tillstånd för publiceringen.

TB1

MOTOROLA M6800 USER GROUP LIBRARY
SUBJECT INDEX

1. SYSTEMS ROUTINES

Monitors, Operating Systems - 29, 90, 100, 114, 121, 123
Simulators - 100, 108
Debug Routines - 39, 43, 49, 53, 114
Interpreters, Compilers - 65
Cross-Computer - 19, 67

2. MATH ROUTINES

Integer - 7, 8, 9, 23, 27, 32, 46, 72, 95, 96, 125
Floating Point - 15, 32, 50, 75, 76, 89, 101
BCD Routines - 22, 31, 32, 96, 98
Functions - 23, 27, 55, 72, 75, 76, 89, 92, 95, 113
Number Base, Code Conversion - 1, 6, 22, 24, 25, 31, 35, 37
46, 68, 70, 77, 86, 88, 97, 125

3. I/O DRIVERS & ROUTINES

Printer Drivers - 4, 5, 10, 11, 18, 60, 93, 103, 110
Punch Drivers - 10, 11, 17, 60, 104, 105, 109
Card & Tape Reader - 10, 11, 16, 51, 59, 60, 104, 106
Plotter & Graphing - 54, 64
Disk - 41, 58, 79, 81, 99, 106
Miscellaneous I/O - 66, 80, 102

4. UTILITY ROUTINES

Sort - 61, 107
Clocks - 62, 117, 118
Cross-Computer - 87, 99
D2 Kit - 73, 116, 117, 122, 123, 125
Test Programs - 12, 13, 14, 38, 115, 124
PROM/EROM Programmer - 21, 28, 69, 82, 84
General - 33, 34, 36, 42, 45, 47, 53, 56, 57, 63, 71, 78,
83, 87, 94, 111, 112, 119, 120, 124

5. GAMES & DEMO

Games - 20, 30, 40, 52, 85, 122
Demo - 44, 48, 73, 86

forts.n.sida

PROGRAMS IN THE M6800 USER GROUP LIBRARY

1. EXBUG ROUTINE CBCDHX

   Convert a Hexadecimal character to a binary number.

2. EXBUG ROUTINE INCHNP

   Input character with no parity.

3. EXBUG ROUTINE INCH

   Input one character

4. EXBUG ROUTINE OUTCH

   Output one character.

5. EXBUG ROUTINE PDATA

   Print CR/LF/DATA string

6. BINARY TO DECIMAL ASCII CONVERSION

   Converts a 16 bit binary number into five ASCII decimal characters.

7. HIGH SPEED DOUBLE PRECISION MULTIPLY

   Multiplies two 16 bit unsigned binary numbers resulting in a 32 bit product.

8. REENTRANT 16 BIT DIVIDE

   Divides two 16 bit unsigned numbers giving a 16 bit result.

9. REENTRANT DOUBLE PRECISION MULTIPLY

   A reentrant routine that multiplies two 16 bit unsigned binary numbers to produce a 32 bit product.

10. M6800 RESIDENT I/O PACKAGE - EXBUG VERSION (MEXRIO)

    I/O routines for use with the Resident Editor and Resident Assembler utilizing EXBUG 1.1 or 1.2

11. M6800 RESIDENT I/O PACKAGE - MIKBUG VERSION (MKRIO)

    I/O routines for use with the Resident Editor and Resident Assembler utilizing MIKBUG Revision 9

12. MPU INSTRUCTION TEST

    To insure the operation of the instruction set.

13. MEMORY TEST

    To test memory. To verify that a particular data pattern can be written to a specific location and read back error free.

14. PIA TEST

    Test program to functionally verify the EXORciser I/O Module (or PIA's).

15. 32 BIT REENTRANT FLOATING POINT MULTIPLY

    Takes two 32 bit floating point numbers and multiplies them. No normalization is attempted. Exponent overflow is not detected.

16. HIGH DENSITY TAPE LOAD

    Loads a properly formatted binary object paper tape into memory from a teletype.

17. HIGH DENSITY TAPE PUNCH

    Punches a properly formatted binary paper tape from memory for loading with the biload program.

18. LPOUT

    ACIA line printer driver.

* 19. INTERDATA CROSS ASSEMBLER

    A cross assembler program that is written for an Interdata computer that will assemble source programs written for the M6800 Microprocessor.

20. TIC-TAC-TOE

    Allows an operator to play the game of Tic-Tac-Toe at the EXORciser console device.

---

* The source listing of this particular program is not included in the User Group Library. The program's source is available only on paper tape, cassette or diskette. (See program description).

21. "PUNCH" FOR DATA I/O PROM PROGRAMMER

Punches a paper tape from any starting address i, memory to any finish address.

22. BCD TO BINARY CONVERSION

Convert four BCD digits to their binary equivalent.

23. SQUARE ROOT OF A 16 BIT NUMBER

Program finds square root of a 16 bit number.

24. UPC (UNIVERSAL PRODUCT CODE) LABEL CONVERSION

Decodes the Universal Product Code symbol found on grocery store labels when scanned with an optical wand.

25. ASCII DATA TO MORSE CODE CONVERSION

Take ASCII data from the keyboard of the EXORciser and convert it to morse code.

26. 32 BIT FLOATING POINT PACKAGE

Subroutines for adding, subtracting, multiplying and dividing floating point numbers of a specified format.

27. RANDOM NUMBER GENERATOR

Generates a (Psuedo) random sequence of 16 bit binary numbers R(n) using a linear congruential sequence.

28. "PUNCH" FOR PROLOG SERIES 90 PROM PROGRAMMER

To generate a formatted tape for the PROLOG series 90 PROM programmer for programming type 2708 Erasable PROM's.

29. M6800 REAL TIME OPERATING SYSTEM

A realtime operating system for the Motorola M6800 Microprocessor

30. BLACKJACK

EXORciser Deals Blackjack

31. BINARY TO BCD CONVERSION

Converts a 16-bit binary number (unsigned) to packed BCD 5-digit (3 byte).

32. DECIMAL ARITHMETIC PACKAGE

This is a group of 8 multiple-precision BCD arithmetic subroutine. These subroutines are integer and floating point addition, subtraction, multiplication and division. Also included is a subroutine to move the numbers to and from the scratchpad (operations: A+C, A-C, A*B and C/A).

33. DELAY SUBROUTINE

To generate a delay from 1→65, 536 milliseconds. To test a PIA bit (for either a one or a zero) until either the test is met or a timeout occurs.

34. OFFSET LOAD MEMORY FROM TAPE

Load memory from paper tape reader (ASR 33) at a location in memory at a constant offset location from that specified on the tape.

35. BNASC

This subroutine converts an unsigned 16-bit number from binary to ASCII of any base between 2 and 16.

36. CHRISTMAS CARD LIST

Prints Address labels. Provision for index, loading, verification, and deletion of individual label.

37. REENTRANT ASCII TO BAUDOT CONVERTER SUBROUTINE (ASTOBAUD)

Converts ASCII characters to their BAUDOT equivalents including shift codes.

38. TEXT STRING (TXTSTG)

A text string is input via keyboard followed by EOT (CTRL/D) delimiter. The text string is then printed repeatedly. Exit is by CPU RESET. Useful as I/O checkout, general system functioning, and effective for demonstrations.

forts.n.sida

39. DUMP/HEX/DISPLAY

Dumps a block of memory in hex and display mode. Can be called as a subroutine or driven from the control console.

40. HANGMAN

This is a game in which the computer selects a word from a programmed list of words. The selection is randomized. The player then enters guesses of letters in the word. The game is completed when the word is correctly identified, or when eight incorrect guesses result in display of the entire HANGMAN!

41. MEMORY DUMP ON DISKETTE (FDISK)

Dump the contents of memory onto an EDOS2 structured diskette in the Motorola defined format (S0, S1, S9).

42. VERIFY PAPER TAPE COPIES (TAPVRF)

Compares paper tape copies punched in any format with the original paper tape.

43. DEBUG

Debug package for M6800. The package provides commands for: examine and/or change of memory locations, examine and/or change of processor registers, step-by-step program execution, input of binary formatted paper tape on Teletype Reader.

44. TOWERS OF HANOI SOLUTION

Solves the "Towers of Hanoi" puzzle for up to 9 discs. Was written to demonstrate how to write recursive programs on the M6800.

45. WL*Ø3 (WIRE LISTER)

Creates, edits, and prints a file of signal names and pin coordinates from a PC board. The print-out could be used in an automatic wire-wrapping system.

46. NON-LINEAR CONVERSION TABLES (CVTBLS)

This program converts any X value between 0 and 10,240 to a Y value between 0 and 10,240. The conversion utilizes look-up tables of 11 Y values, with a linear interpolation between them.

47. ASSEMBLER/EDITOR INTERNAL TRANSFER ROUTINE (EXBUG)--AETX.1

This program, when loaded after the co-resident Assembler/Editor software, allows internal two-way transfer between the Assembler and Editor without the external tape transfers otherwise needed. Edit buffer and assembler symbol table are separately maintained.

48. BIORHYTHM

Produces a plot of a person's biorhythm - physical, emotional and intellectual cycles.

49. TRACE

Trace is a subroutine which the user jumps to at various "test points" is a new un-debugged program for the purpose of printing out the address followed by the contents of the A and B registers. This allows tracing of the flow of the new program up to the point at which it "blows".

50. 24 BIT REENTRANT FLOATING POINT PACKAGE

(1) addition; (2) subtraction; (3) multiplication; (4) division; (5) float; (6) fix to 16 bit integer; (7) negate; (8) clear; (9) copy; (10) compare*; (11) string to flt pnt*; (12) flt pnt to string*
* - not reentrant

51. RDR1.B

Ram memory loader for Motorola format hex paper tape using 300 char/sec reader. Loads memory with a single program and halts.

52. MISSIONARIES AND CANNIBALS (MISCAN)

The object of the game is to move three missionaries and three cannibals across a river, in a boat that can only accommodate two people.

53. DUMP

Memory dump using MIKBUG for I/O

54. CHARACTER GENERATION ON X-Y RECORDER

The system accepts alphanumeric characters from keyboard and translates them into a series of x-y displacement voltages, forming the shape of entered character on x-y recorder.

55. CYCLIC REDUNDANCY CHECK CHARACTER GENERATOR

Calculation of the CRCC of a data buffer stored in locations Bebuffer-Ebuffer.

56. DECODE

Provides the convenience of a "backwards" assembler; that is, given a machine-resident block of object code the program will analyze it and provide a pseude-source listing of the code.

57. PRTMSG

A reentrant message printing routine.

58. DISK FILE TRANSFER

To give EXORciser users with single drive unit EXORdisks the capability to copy a file from one diskette to another without using paper tape or cassette as intermediate media.

59. QUICK LOAD (QLOAD)

Loads an EXORciser format object tape into memory without stopping the tape after each record. Will detect and indicate checksum and no change errors.

60. M6800 RESIDENT I/O PACKAGE - MINIBUG II VERSION (M2RIO)

I/O routines for use with the Resident Editor and Resident Assembler utilizing Minibug II

61. MPL CORE SORT PROGRAM

Sorts data arrays in ascending order. Data array can either be BINARY (1) or BINARY (2) and a data entry can be of arbitrary length. The sort is done in place.

62. 24-HOUR CLOCK

Maintains and displays 24 hour clock.

63. TTYIO

This is an input-output sub-routine. It will output a defined character string or store a character string in a defined buffer (location).

64. TELETYPE GRAPH GENERATOR

Collects variables during real time data taking and prints out a graph at the end of a run.

65. MICRO-BASIC

BASIC Interpreter for the M6800.

66. INSTRUMENT INTERFACER

Allows peripherals which do not comply with the IEEE standard for programmable instruments to be connected to a controller which does comply.

• 67. M6800 CROSS ASSEMBLER FOR WANG 2200B

An M6800 Cross Assembler for Wang 2200B with 4K memory.

68. GRAY CODE TO BINARY CONVERSION

Subroutine for conversion of 16 bit Gray code into 16 bit binary. (Reentrant)

69. PROM PROGRAMMER

For EDOS-2 system, this program allows programming of a 2708 type EPROM with any origin from an object file on diskette.

70. EIA TO ASCII

Converts EIA coded paper tapes to ASCII coded paper tape.

forts.n.sida

71. TELEPHONE DIALER

Provides properly timed pulses to a relay from a stored telephone number.

72. ISQRT

Calculates the integer square root of any positive number up to 255 decimal. (Subroutine)

73. MUSIC

Allows the M6800 to play song sequences which are loaded into memory. Written for the MEK6800-D2 kit.

74. FALSE LOADER

Allows an object tape with EXORciser format to be loaded and verified into a specific memory area.

75. FUNCTION exp(x), ln(x)

Calculates the exponent and natural log in hexadecimal floating point format. (Subroutine)

76. FLOATING POINT MATH ROUTINES

FIX - Changes a floating point value to integer.
FLOAT - Changes an integer value to floating point.
SQRT - Calculates square root of a floating point value.

77. BINA

Reentrant subroutine to convert an 8 bit binary value into ASCII characters in any base. Also includes a reentrant 8 bit division subroutine.

78. LPNT - MEMORY DUMP TO LINE PRINTER

Allows dumping of memory to line printer. Works like EXBUG "PRNT" function.

• 79. MECOPY

An MDOS program to copy ASCII MDOS files to an EDOS-2 diskette.

• 80. HP2644A INTERFACE I/O

Interfaces HP2644A CRT terminal to an EXORciser so that Data Cartridges can be used for EXBUG functions.

81. CRT/EDOS

Four programs which modify EDOS 2.2 to work with CRT.

82. PROM

Converts PROM paper tapes from Weatherford format to a binary format for DATA I/O PROM programmers.

83. MEMOVE

Reassembles object code to new memory location without reassembly of source file, or moves a block of memory without alteration from one location to another.

84. PNGEN

Allows preparation of PROM programmer input tapes in BNPF format.

85. NIM GAME

Program plays the game of NIM against a human opponent.

86. MORSE CODE SEND AND RECEIVE

Converts Morse code to ASCII and prints on a terminal, or allows operator to type on terminal keyboard and program sends Morse code equivalent.

87. EXORCYB

Allows EXORciser to communicate with time-sharing computer via an additional ACIA and Modem.

88. CODEC - ASCII/BAUDOT

Performs the conversion in codes between ASCII and Baudot and vice-versa.

89. MATH

24 bit floating point routines for addition, subtraction, multiplication, division, square root, sine, cosine and tangent.

90. MONTR4

Provides extended monitor functions using Mikbug I/O routines.

91. KEYBRD

Subroutine to interface hexadecimal keyboard to PIA and perform strobe, debounce and decode functions.

92. TRIG

Subroutine to calculate the SIN and COS of an angle.

93. QUMEPR

Routine for QUME Sprint Micro-3 (R) printer. Outputs ASCII data to printer with provisions for both 10 and 12 pitch printwheels, various line spacing, and top of form control.

94. PPX

Two re-entrant subroutines to perform the PUSH X and PULL X functions. No registers are altered and works with interrupts.

95. INTEG

Subroutine integrates a series of N evenly-spaced points (N must be between 7 and 255 inclusive).

96. SBBCD

Subroutine for subtraction of BCD numbers. Routine is reentrant and works with double-precision.

97. CONV

Four re-entrant subroutines are provided to perform the following conversions:

- BOAC - Converts a 2 byte binary number to octal ASCII.
- OABC - Converts octal ASCII characters to 2 byte binary.
- OABC - Converts a 2 byte binary number to decimal ASCII.
- BOAC - Converts decimal ASCII characters to 2 byte binary.

98. BCDRTN

This is a collection of 8-digit BCD subroutines including moves from a working register (4-bytes), swap registers, binary and 10's complements, clear register, shift nibbles, character search, addition, and zero word test.

99. FILETX

This program allows transfer of XEROX 560 disk files to/from MDOS disk files.

100. PDP8EM

This program emulates a DEC PDP-8 (Reg. T.M.) on the EXORciser. Every program written for a PDP-8 will run on the EXORciser using this program. It includes a monitor simulating the major function switches of the PDP-8.

101. FPCMP

Floating Point Comparison routine compares two ANSI standard FORTRAN single precision floating point numbers and sets condition codes to allow conditional branch operations to take place. May be adapted to various precisions.

102. DA8CH

Performs the function of 8 channel digital to analog conversion by pulse width modulation. Useful for driving chart recorder or DC voltmeters.

103. DIABPD

Printer Driver for Diablo Hytype II. Outputs MDOS ASCII file from disk to printer.

104. HSPTD

High speed paper tape driver for Remex or equivalent high speed punch/reader combination. Allows generation, checking, and loading of hexadecimal format tapes.

105. PREMEX

Punches an MDOS disk file onto a high speed punch (Remex).

forts.n.sida

106. CRODRV

Loads a properly formatted source assembly file from cards into a source assembly file on MDOS disk.

107. RSORT

Ripple sort subroutine to sort variably dimensioned data blocks.

108. SIMUL

M6800 resident simulator. Simulates a user program by following each instruction and printing a register dump after every instruction.

109. FLAGX

ASCII to 5x7 matrix character converter written to interface with Remex paper tape punch.

110. LPDRV1

Line printer driver for MDOS using a 300 Baud RS232-C interface.

111. ALPHBT

Routine to output alphanumeric upper and lower case characters on the printer in large block style. Requires MDOS.

112. AUTOBD

This routine detects the baud rate of an EXORciser ACIA and programs the ACIA for proper 11 or 10 bit operation.

113. ARRAYP

128 x 128 bit array processor routines to initialize, read, and write to the array.

114. MAARX

This program provides a minimum set of program debug features including a disassembler and a memory editor which uses assembler op-codes. (Minimum Assembler And Resident System).

115. ACIACK

Tests most of the functions of the ACIA card.

116. DISKEY

A collection of 5 subroutines to perform LED display and keyboard input functions on the MEK6800D2 kit. (D2 kit)

117. KTIME

A short subroutine to use the D2 kit for a clock or elapsed timer.

118. RTCLK

A routine which performs functions of a real-time clock by counting interrupts generated by a 60 Hz. input. It tracks seconds, minutes, hours, days, months, and years.

119. PTCG

This program punches paper tape with alphanumeric, humanreadable characters, both upper and lower cases, and with both normal and reversed background (character holes on solid background or solid characters on hole background).

120. BIGBAN

This program prints very large block letters on a printer.

121. PRMBUG

This is a monitor/debug type program, retaining MIKBUG address compatibility, yet with improved I/O and commands.

122. MSTRMD

Version of the game MASTERMIND played on the D2 kit.

123. D2MON

This is a general-purpose monitor written for the D2 kit using an ASCII terminal. There are 18 commands. It is written to be programmed into two 1K EROMs.

124. PARITY

Subroutine to determine parity of an 8-bit word.

125. HEXCAL

Performs addition and subtraction in either base 10 or base 16, converts back and forth between the two bases. Uses the hexadecimal keyboard and displays of the D2 kit.

Morse—Ascii—Morse

Det program som vi skall titta på nu heter CWMK. Det kommer från Motorola User Group Library och har nummer 86. Programmet är inte enbart tänkt att glädja våra många radioamatörer. Tvärtom är det snarare vi andra som kan få en massa skoj. Om vi har en radio med kortvågsband skall vi med programmets hjälp kunna få klartextutmatning på CRT:n genom att avlyssna trafik mellan exempelvis båtar. Att omvandla ASCII tecken från tangentbordet till ekvivalenta MORSE tecken klarar programmet också. Innan man fått ut något i etern krävs naturligtvis lite mellanliggande radioutrustning. Låt oss se hur den översatta programbeskrivningen låter:

- Detta är ett program för MC6800 mikrodatorn som automatiserar sändning och mottagning hos en CW (International Morse Code) radio station.

I sändningsmod slår operatören in sina tecken via ett ASCII tangentbord. ASCII tecknen konverteras till sina Morse ekvivalenta tecken och placeras i en buffert. Under timing från en justerbar extern klocka (oscillator), tas morse tecknen från bufferten och sänds ut till en PIA utgång. Där kan denna signal användas för att nyckla en r.f. sändare eller lokal tongenerator.

I motagningsmode tas den till digital nivå omvandlade morsesignalen emot på en annan PIA anslutning. Mjukvaran konverterar den mottagna digitala signalen till motsvarande morsetecken. Detta omvandlas därefter till ett ASCII tecken som sänds till en ACIA för utskrift.

Mottagningsalgorithmen justerar för olika hastighet på den mottagna koden. Den är kapabel att klara hastigheter över ett område från mindre än 1/2 ord till över 250 ord i minuten (det står ord, skall det vara tecken??) utan justering av operatören.

Översättningstabell f. tecken

Datat som är lagrat i tabellen använder logisk 0 för punkter och logisk 1 för streck (det heter väl punkt och streck??). Data strängen termineras med en logisk 1 för att bestämma strängslut. Data är vänsterjusterat i 8-bits byten och nollutfyllt i ej utnyttjade positioner.

Tag som exempel morsetecknet för "P". Det är punkt-streck-streck-punkt. Detta representeras som 0110. En avslutningsbit (logisk 1) adderas till på högersidan och de återstående bitarna nollutfylls. Resultatet blir då 01101000. I hexadecimal form är motsvarigheten $68.

Tabellens ingångsläge är lagt i minnet på sådant sätt att dess lägre adresshalva direkt motsvarar ASCII koden. I exemplet ovan är 68 placerad på adress 0450. Den lägre halvan av adressen (50) är ASCII motsvarigheten till "P".

Tabellinnehåll med nollor är icke existerande morsetecken. Alla värden syns ej tillsammans med sina adresser pga. användningen av assembleroptionet NOG. Det ger kortare lista.

Hårdvaru interface

För att konvertera audiotonerna från en radiomottagare till de önskade digitala nivåerna måste ett interface användas. En sådan krets visas i okt,nov och december numren av QST tidningen 1975. (Publicerad av ARRL, Newington , CT). Om denna krets används, skall dess utgång inverteras och anslutas till 1 i fig 2 efter borttagning av +5 v. anslutningen. En logisk 1 är ekvivalent till "nyckel ned" tillstånd på pin PA7.

forts.n.sida

Specifikationer

- RAM: 0000-007F och A000-A07F
- programminne: 0400-07FF
- PIA: 8008-800B
- ACIA: 8010-8011

Behövligt minne är mindre än 256 bytes RAM och mindre än 1k bytes programminne (RAM/ROM).

Program index (funktion/radnr)

1. RAM och I/O adressdef 23-61
2. Morse kod/ASCII omv.tab 64-86
3. Restart/initierings rut 89-124
4. Executiv rutin 127-144
5. Interrupt polling och housekeeping rutin 160-173
6. Morse kod sänd rutin 175-326
7. Morse kod mottagn rutin 329-529
8. Textsträngar för utskrift av fasta meddelanden 532-538

OBSERVERA:

Starta programmet ($045B) med avbrottsmasken (I biten) satt i mikroprocessorns statusregister. Anslut icke använda DCD och CTS pinnarna på ACIA:n till logisk 0.

Så långt programbeskrivningen i User Group Library. Har man sin PIA/ACIA på andra adresser går det ganska lätt att snoka upp och ändra referenserna. De är inte så många.

Det är inte troligt att vi är i stånd att prova detta program före pressläggningen av MPU-laren. Inte så att själva programmet skulle innehålla fel. Utan det skulle varit roligt att visa ett komplett interface. Hinner vi så klämmer vi in ett som fig 1.

Programkoden kan levereras på KCS audiokassett i S1-format, MDOS diskett samt så småningom FLEX mini diskett. Tape håller vi med, diskett kopierar vi till din. Hör av dig - det är gratis.

TB1

```
00001                           NAM    CWMK
00002                           OPT    OT,S,NOG
00003                   ********************************************************
00004                   * MORSE CODE/ASCII SEND/RECEIVE PROGRAM
00005                   * FOR THE MOTOROLA 6800 MICROPROCESSOR
00006                   * WRITTEN BY:
00007                   *    DON L. JACKSON - W7KGU
00008                   *    JIM BAINTER - WA7VKZ
00009                   * VERSION 2.0 - FOR USE ON MOTOROLA
00010                   *  DESIGN EVALUATION KIT (MIKBUG).
00011                   ********************************************************
00012                   * I/O HARDWARE CONFIGURATION:
00013                   *    PIA ADDRESS - $8008
00014                   *       CA1 - INTERRUPT TIMING FOR XMIT CODE,
00015                   *             2-50 HZ.
00016                   *       PA0 - T/R SWITCH, 0=TX, 1=RCV
00017                   *       PA7 - RECEIVE CODE INPUT
00018                   *       PB0 - CODE OUT
00019                   *       PB1 - TRANSMIT LAMP
00020                   *       PB2 - RECEIVE LAMP
00021                   *    ACIA ADDRESS - $8010
00022                   ********************************************************
00023A 0000  0002  A CVCX   RMB    2        INDEX REG CONVERT STORE
00024A 0002  0002  A SAVEX  RMB    2
00025A 0004  0001  A COUNT  RMB    1
00026A 0005  0001  A RESMSK RMB    1        COSTA RESET MASK
00027A 0006  0001  A BUFLAG RMB    1        B7=1  DATA IN BUFFER
00028                 *                     B6=1 CRLF TO PRINTER
00029A 0007  0001  A CHCNT  RMB    1        CHAR. COUNTER FM KB
00030A 0008  0001  A COSTA  RMB    1        CODE OUTPUT STATUS
00031                 *B7 DIT FLAG,B6 DAH FLAG,B5 ELEMENT SPACE FLAG
00032                 *B4 WORD SPACE FLAG,B3 CHAR. SPACE FLAG
00033A 0009  0001  A LETYPE RMB    1        LAST ELEM TYPE DOT=0 DASH>
00034A 000A  0001  A HLETIM RMB    1        HALF LAST ELEM TIME
00035A 000B  0001  A TLETIM RMB    1        TWICE LAST ELEM TIME
00036A 000C  0001  A SPEEDK RMB    1        SPEED CONSTANT
00037A 000D  0001  A RCHAR  RMB    1        CHAR BEING RECEIVED
00038A 000E  0001  A LDATIM RMB    1        LAST DASH TIME
00039A 000F  0001  A TQLDAT RMB    1        3/4 LAST DASH TIME
00040A 0010  0001  A TLDAT  RMB    1        TWICE LAST DASH TIME
00041A 0011  0001  A KDTIM  RMB    1        KEYDOWN INTERVAL TIME
00042A 0012  0001  A KUTIM  RMB    1        KEYUP INTERVAL TIME
00043A 0013  0001  A CHCTR  RMB    1        REC. CHARACTER COUNTER
00044A 0014  0002  A RECX   RMB    2        REC. INDEX REG. TO ACCA.
00045A 0016  0002  A P1     RMB    2        REC. BUFFER POINTER
00046A 0018  0002  A P2     RMB    2        REC. BUFFER POINTER
00047A 001A  0065  A BUFBOT RMB    101      XMIT BUFFER BOTTOM
00048A 007F  0001  A BUFTOP RMB    1        XMIT BUFFER TOP
00049A A04A                 ORG    $A04A
00050A A04A  0009  A BBOT   RMB    9        BOTTOM OF RCV BUFFER
00051A A053  0001  A BTOP   RMB    1        TOP OF RCV BUFFER
00052A 8008                 ORG    $8008
00053A 8008  0001  A PIAAD  RMB    1
```

forts.n.sida

```
00054A 8009    0001  A PIAAC  RMB    1
00055A 800A    0001  A PIABD  RMB    1
00056A 800B    0001  A PIABC  RMB    1
00057A 8010                   ORG    $8010
00058A 8010    0001  A ACIAC  RMB    1
00059A 8011    0001  A ACIAD  RMB    1
00060          A07F  A STACK  EQU    $A07F
00061          A000  A IRQVEC EQU    $A000

00063A 0400                   ORG    $0400
00064                  *               KN  BK  AR  SK   ACK
00065A 0400    00    A CODE   FCB    0,$B4,$8B,$54,$16,0,$14,0
00066                  *              ERR         SP
00067A 0408    01    A        FCB    1,0,0,0,0,$21,0,0
00068A 0410    00    A        FCB    0,0,0,0,0,0,0,0
00069                  *                    AS
00070A 0418    00    A        FCB    0,0,0,$44,0,0,0,0
00071                  *               SP   "         '
00072A 0420    21    A RTAB   FCB    $21,0,$4A,0,0,0,0,$7A
00073                  *               (   )      ,   -   .   /
00074A 0428    B6    A        FCB    $B6,$B2,0,0,$CE,$86,$56,$94
00075                  *               0   1   2   3   4 5 6   7
00076A 0430    FC    A        FCB    $FC,$7C,$3C,$1C,$0C,4,$84,$C4
00077                  *               8   9   :   ;    BT(=)  ?
00078A 0438    E4    A        FCB    $E4,$F4,$E2,$AA,0,$8C,0,$32
00079                  *                 A   B   C   D   E   F   G
00080A 0440    00    A        FCB    0,$60,$88,$A8,$90,$40,$28,$D0
00081                  *               H   I   J   K   L   M   N   O
00082A 0448    08    A        FCB    $08,$20,$78,$B0,$48,$E0,$A0,$F0
00083                  *               P   Q   R   S   T   U   V   W
00084A 0450    68    A        FCB    $68,$D8,$50,$10,$C0,$30,$18,$70
00085                  *               X   Y   Z
00086A 0458    98    A        FCB    $98,$B8,$C8

00088                  ***RESTART ROUTINE***
00089A 045B CE 007F   A RESRT  LDX    #$7F     CLR RAM 1-7F
00090A 045E 6F 00     A L1     CLR    0,X
00091A 0460 09                 DEX
00092A 0461 26 FB 045E         BNE    L1
00093A 0463 CE 0F01   A        LDX    #$0F01
00094A 0466 DF 0C     A        STX    SPEEDK   INZ SPEED CONS & RCHAR
00095A 0468 86 80     A        LDAA   #$80
00096A 046A 97 7F     A        STAA   BUFTOP   INZ BUFTOP
00097A 046C CE A053   A        LDX    #BTOP    INZ BUFFER POINTERS
00098A 046F DF 16     A        STX    P1
00099A 0471 09                 DEX
00100A 0472 DF 18     A        STX    P2
00101A 0474 CE 0400   A        LDX    #CODE
00102A 0477 DF 00     A        STX    CVCX     INZ CVCX
00103A 0479 CE 007F   A        LDX    #BUFTOP
00104A 047C DF 02     A        STX    SAVEX
00105A 047E CE 04E6   A        LDX    #POLL    INZ IRQ VECTOR IN MIKBUG
00106A 0481 FF A000   A        STX    IRQVEC
```

```
00107A 0484 CE 8008  A        LDX    #PIAAD
00108A 0487 6F 01    A        CLR    1,X       CLR PIAAC
00109A 0489 6F 03    A        CLR    3,X       CLR PIABC
00110A 048B 8E 0007  A        LDS    #$0007    CA1=+ & ALLOWED
00111A 048E AF 00    A        STS    0,X       (PIAAD & PIAAC)
00112A 0490 8E FF34  A        LDS    #$FF34
00113A 0493 AF 02    A        STS    2,X       (PIABD & PIABC)
00114A 0495 A6 00    A        LDAA   0,X       CLR IRQA FLAGS
00115A 0497 A6 02    A        LDAA   2,X       CLR IRQB FLAGS
00116A 0499 86 03    A        LDAA   #3
00117A 049B A7 08    A        STAA   8,X       RESET ACIA
00118A 049D 86 81    A        LDAA   #$81      7 DATA, EVEN PARITY, 2 STO
00119A 049F A7 08    A        STAA   8,X       INZ ACIAC
00120A 04A1 8E A07F  A        LDS    #STACK    INZ STACK POINTER
00121A 04A4 8D 29 04CF        BSR    SCRLF     PRINT CR/LF
00122A 04A6 CE 074B  A        LDX    #MSG1
00123A 04A9 8D 27 04D2        BSR    SCS       PRINT MSG1
00124A 04AB 0E                CLI              CLEAR INTERRUPT FLAG

00126                  *** EXECUTIVE PROGRAM ***
00127A 04AC 96 06    A EXEC   LDAA   BUFLAG    CHECK FOR CRLF
00128A 04AE 48                ASLA
00129A 04AF 2A 08 04B9        BPL    EX2       NO CRLF
00130A 04B1 8D 1C 04CF        BSR    SCRLF
00131A 04B3 96 06    A        LDAA   BUFLAG
00132A 04B5 84 BF    A        ANDA   #$BF      ZERO B6 IN BUFLAG
00133A 04B7 97 06    A        STAA   BUFLAG
00134A 04B9 96 07    A EX2    LDAA   CHCNT
00135A 04BB 81 41    A        CMPA   #65
00136A 04BD 23 0D 04CC        BLS    EX1
00137A 04BF DE 02    A        LDX    SAVEX     66 CHAR FM KB
00138A 04C1 A6 00    A        LDAA   0,X
00139A 04C3 81 21    A        CMPA   #$21      LOOK FOR WORD SPACE
00140A 04C5 26 05 04CC        BNE    EX1       NO WORD SPACE
00141A 04C7 8D 06 04CF        BSR    SCRLF     CR/LF TO PRINTER
00142A 04C9 7F 0007  A        CLR    CHCNT
00143A 04CC 3E           EX1  WAI
00144A 04CD 20 DD 04AC        BRA    EXEC

00146                  ***SUBR TO SEND CR/LF OR CHAR STRING TO PRINTER**
00147A 04CF CE 07B1  A SCRLF  LDX    #CRLF
00148A 04D2 A6 00    A SCS    LDAA   0,X
00149A 04D4 81 04    A        CMPA   #4
00150A 04D6 27 0D 04E5        BEQ    RTSR      END OF CHAR STRING
00151A 04D8 F6 8010  A SCR1   LDAB   ACIAC
00152A 04DB C5 02    A        BITB   #2        TST FOR TX REG. EMPTY
00153A 04DD 27 F9 04D8        BEQ    SCR1      NOT EMPTY
00154A 04DF B7 8011  A        STAA   ACIAD     SEND CHAR TO PRINTER
00155A 04E2 08                INX
00156A 04E3 20 ED 04D2        BRA    SCS
00157A 04E5 39           RTSR RTS

00159                  ***IRQ INTERRUPT ROUTINE***
```

forts.n.sida

```
00160A 04E6 F6 8009  A POLL   LDAB   PIAAC    SAVE INTR. FLAGS.
00161A 04E9 B6 8008  A        LDAA   PIAAD
00162A 04EC 46                RORA
00163A 04ED 25 0A 04F9        BCS    RECM     REC MODE
00164A 04EF 5D          POLL1 TSTB             CHECK FOR TX INTR.
00165A 04F0 2B 0E 0500        BMI    POLL2    (CA1) YES
00166A 04F2 F6 8010  A        LDAB   ACIAC    CHECK ACIA RDRF
00167A 04F5 56                RORB
00168A 04F6 25 6D 0565        BCS    COMMR1
00169A 04F8 3B                RTI

00171A 04F9 96 06    A RECM   LDAA   BUFLAG
00172A 04FB 2B F2 04EF        BMI    POLL1    DATA IN TX BUFFER
00173A 04FD 7E 0603  A RECI   JMP    REC      GO TO REC ROUTINE

00175A 0500 F6 800A  A POLL2  LDAB   PIABD
00176A 0503 CA 02    A        ORAB   #2       XMIT LAMP ON
00177A 0505 C4 FB    A        ANDB   #$FB     REC LAMP OFF
00178A 0507 F7 800A  A        STAB   PIABD
00179A 050A 96 08    A        LDAA   COSTA    GET CODE OUTPUT STATUS
00180A 050C 2A 07 0515        BPL    CFDAH    CHECK FOR DAH
00181A 050E C6 7F    A        LDAB   #$7F
00182A 0510 D7 05    A        STAB   RESMSK   DIT RESET
00183A 0512 5F          CES1  CLRB             DIT BEING SENT
00184A 0513 20 29 053E        BRA    CKCNT    TEST COUNT

00186A 0515 48          CFDAH ASLA             TEST FOR DAH
00187A 0516 2A 08 0520        BPL    CFES     CHECK FOR ELEMENT SPACE
00188A 0518 C6 BF    A        LDAB   #$BF
00189A 051A D7 05    A        STAB   RESMSK   DAH RESET
00190A 051C C6 04    A        LDAB   #4       DAH BEING SENT
00191A 051E 20 1E 053E        BRA    CKCNT

00193A 0520 48          CFES  ASLA             TEST FOR ELEMENT SPACE
00194A 0521 2A 0A 052D        BPL    CFWS     CHECK FOR WORD SPACE
00195A 0523 C6 DF    A        LDAB   #$DF
00196A 0525 D7 05    A        STAB   RESMSK   ELEMENT SPACE RESET
00197A 0527 8D 2A 0553        BSR    SOZERO
00198A 0529 C6 01    A        LDAB   #1
00199A 052B 20 11 053E        BRA    CKCNT

00201A 052D 48          CFWS  ASLA
00202A 052E 2A 08 0538        BPL    CFCS     CHECK FOR SPACE
00203A 0530 C6 EF    A        LDAB   #$EF
00204A 0532 D7 05    A        STAB   RESMSK   WORD SPACE RESET
00205A 0534 C6 08    A        LDAB   #8       WORD SPACE BEING SENT
00206A 0536 20 06 053E        BRA    CKCNT

00208A 0538 C6 F7    A CFCS   LDAB   #$F7
00209A 053A D7 05    A        STAB   RESMSK
00210A 053C C6 04    A        LDAB   #4
00211A 053E D1 04    A CKCNT  CMPB   COUNT
00212A 0540 27 04 0546        BEQ    CK1
```

```
00213A 0542 7C 0004  A        INC    COUNT
00214A 0545 3B         RETRN  RTI

00216A 0546 7F 0004  A CK1    CLR    COUNT
00217A 0549 96 08    A        LDAA   COSTA
00218A 054B 94 05    A        ANDA   RESMSK
00219A 054D 97 08    A        STAA   COSTA
00220A 054F 26 F4 0545        BNE    RETRN
00221A 0551 20 14 0567        BRA    GNEL

00223                  ***SEND OUT A ZERO***
00224A 0553 B6 800A  A SOZERO LDAA   PIABD
00225A 0556 84 FE    A        ANDA   #$FE
00226A 0558 B7 800A  A        STAA   PIABD
00227A 055B 39                RTS

00229                  ***SEND OUT A ONE***
00230A 055C B6 800A  A SOONE  LDAA   PIABD
00231A 055F 8A 01    A        ORAA   #1        SET BIT ZERO
00232A 0561 B7 800A  A        STAA   PIABD
00233A 0564 39                RTS

00235A 0565 20 53 05BA COMMR1 BRA    COMMR

00237                  ***GET NEW ELEMENT FOR OUTPUT***
00238A 0567 96 7F    A GNEL   LDAA   BUFTOP
00239A 0569 81 80    A        CMPA   #$80      TEST FOR CHAR END
00240A 056B 26 22 058F        BNE    GNEL1
00241A 056D 86 08    A        LDAA   #$08      SEND OUT CHAR. SPACE
00242A 056F 97 08    A        STAA   COSTA
00243A 0571 8D E0 0553        BSR    SOZERO
00244A 0573 DE 02    A        LDX    SAVEX
00245A 0575 8C 007F  A        CPX    #BUFTOP
00246A 0578 26 07 0581        BNE    GNELR
00247A 057A 96 06    A        LDAA   BUFLAG    BUFFER EMPTY
00248A 057C 84 7F    A        ANDA   #$7F
00249A 057E 97 06    A        STAA   BUFLAG    RESET FLAG
00250A 0580 3B                RTI
00251                    *
00252A 0581 8D 27 05AA GNELR  BSR    MOVUP
00253A 0583 96 7F    A        LDAA   BUFTOP
00254A 0585 81 21    A        CMPA   #$21      LOOK FOR SPACE
00255A 0587 27 16 059F        BEQ    SWS
00256A 0589 7D 0008  A        TST    COSTA
00257A 058C 27 01 058F        BEQ    GNEL1
00258A 058E 3B                RTI
00259                    *
00260A 058F 78 007F  A GNEL1  ASL    BUFTOP
00261A 0592 25 04 0598        BCS    SODAH     SEND OUT DAH
00262A 0594 86 A0    A        LDAA   #$A0      SEND OUT DIT & ELEMENT SPA
00263A 0596 20 02 059A        BRA    SOEL
00264                    *
00265A 0598 86 60    A SODAH  LDAA   #$60      SEND OUT DAH &ELEMENT SPAC
```

```
PAGE  006   CWMK

00266A 059A 97 08    A SOEL   STAA   COSTA    SET STATUS
00267A 059C 8D BE 055C        BSR    SOONE
00268A 059E 3B                RTI
00269                    *
00270A 059F 86 80    A SWS    LDAA   #$80
00271A 05A1 97 7F    A        STAA   BUFTOP
00272A 05A3 86 10    A        LDAA   #$10
00273A 05A5 97 08    A        STAA   COSTA    SEND OUT WORD SPACE
00274A 05A7 8D AA 0553        BSR    SOZERO
00275A 05A9 3B                RTI
00276                    *
00277A 05AA CE 0065  A MOVUP  LDX    #BUFTOP-BUFBOT
00278A 05AD A6 19    A MOV1   LDAA   BUFBOT-1,X
00279A 05AF A7 1A    A        STAA   BUFBOT,X
00280A 05B1 09                DEX
00281A 05B2 26 F9 05AD        BNE    MOV1
00282A 05B4 DE 02    A        LDX    SAVEX    UPDATE INPUT POINTER
00283A 05B6 08                INX
00284A 05B7 DF 02    A        STX    SAVEX
00285A 05B9 39          MOVRT  RTS

00287                    ***COMM REC INTERRUPT ROUTINE***
00288A 05BA DE 02    A COMMR  LDX    SAVEX
00289A 05BC 7C 0007  A        INC    CHCNT    INC CHAR COUNTER
00290A 05BF B6 8011  A        LDAA   ACIAD    GET KB DATA
00291A 05C2 8C 001A  A        CPX    #BUFBOT
00292A 05C5 27 30 05F7        BEQ    BUFFUL   MEMORY BUFFER FULL
00293A 05C7 C5 18    A        BITB   #$18     CK FOR FE OR OVRN
00294A 05C9 26 30 05FB        BNE    ERROR
00295A 05CB 81 0D    A        CMPA   #$0D     CK FOR CR
00296A 05CD 27 1D 05EC        BEQ    SCR      SEND CR
00297A 05CF 81 5A    A        CMPA   #$5A
00298A 05D1 22 2C 05FF        BHI    BADCH    ILLEGAL CHAR
00299A 05D3 97 01    A STM    STAA   CVCX+1
00300A 05D5 DE 00    A        LDX    CVCX
00301A 05D7 E6 00    A        LDAB   0,X      DATA CONVERTED TO MORSE CO
00302A 05D9 27 24 05FF        BEQ    BADCH    KICK OUT BAD CHAR'S
00303A 05DB DE 02    A        LDX    SAVEX
00304A 05DD 09                DEX
00305A 05DE E7 00    A        STAB   0,X      STORE DATA IN BUFFER
00306A 05E0 DF 02    A        STX    SAVEX    MOVE POINTER DOWN
00307A 05E2 D6 06    A        LDAB   BUFLAG
00308A 05E4 CA 80    A        ORAB   #$80     SET FLAG BIT 7
00309A 05E6 D7 06    A        STAB   BUFLAG
00310A 05E8 B7 8011  A PRNT   STAA   ACIAD    PRINT CHARACTER
00311A 05EB 3B                RTI
00312                    *
00313A 05EC D6 06    A SCR    LDAB   BUFLAG
00314A 05EE CA 40    A        ORAB   #$40     SET BIT 6
00315A 05F0 D7 06    A        STAB   BUFLAG
00316A 05F2 7F 0007  A        CLR    CHCNT    RESET COUNTER TO 0
00317A 05F5 20 DC 05D3        BRA    STM
00318                    *
```

```
00319A 05F7 86 2A    A BUFFUL LDAA   #'*      PRINT * FOR BUFFUL
00320A 05F9 20 ED 05E8        BRA    PRNT
00321                 *
00322A 05FB 86 3F    A ERROR  LDAA   #'?      PRINT ? FOR ERROR
00323A 05FD 20 E9 05E8        BRA    PRNT
00324                 *
00325A 05FF 86 07    A BADCH  LDAA   #7       RING BELL FOR BAD CHAR
00326A 0601 20 E5 05E8        BRA    PRNT

00328                 ***START OF RECEIVE ROUTINE***
00329A 0603 B6 800A  A REC    LDAA   PIABD
00330A 0606 8A 04    A        ORAA   #4        REC LAMP ON
00331A 0608 84 FD    A        ANDA   #$FD      XMIT LAMP OFF
00332A 060A B7 800A  A        STAA   PIABD
00333A 060D 86 FF    A KEYUP  LDAA   #$FF      RESET INTERVAL COUNTER
00334A 060F 81 FE    A KULOOP CMPA   #$FE      INTERVAL COUNTER AT MAX?
00335A 0611 27 03 0616        BEQ    NOINC     DO NOT INCR IF MAX
00336A 0613 4C                INCA             INCREMENT INTERVAL COUNTER
00337A 0614 20 0A 0620        BRA    KUCONT

00339A 0616 97 12    A NOINC  STAA   KUTIM     SAVE KU INTERVAL TIME
00340A 0618 D6 0D    A        LDAB   RCHAR     GET RCVD CHAR
00341A 061A C1 01    A        CMPB   #1        ANYTHING THERE?
00342A 061C 27 02 0620        BEQ    KUCONT
00343A 061E 8D 4E 066E        BSR    COMPKU

00344A 0620 F6 8008  A KUCONT LDAB   PIAAD     CHECK INPUT
00345A 0623 2B 08 062D        BMI    KD        BRANCH IF KEYDOWN
00346A 0625 8D 35 065C        BSR    TIMER     TIME DELAY
00347A 0627 56                RORB             CHECK T/R SWITCH
00348A 0628 25 E5 060F        BCS    KULOOP
00349A 062A 7E 04E6  A        JMP    POLL      RETURN TO TX ROUTINE

00351A 062D 97 12    A KD     STAA   KUTIM     SAVE KU INTERVAL TIME
00352A 062F 8D 3D 066E        BSR    COMPKU
00353A 0631 86 FF    A KEYDWN LDAA   #$FF      RESET INTERVAL TIMER
00354A 0633 81 FE    A KDLOOP CMPA   #$FE      INTERVAL COUNTER AT MAX?
00355A 0635 27 01 0638        BEQ    MAXKD     DO NOT INCR IF MAX
00356A 0637 4C                INCA             INCREMENT INTERVAL COUNTER
00357A 0638 F6 8008  A MAXKD  LDAB   PIAAD     CHECK INPUT
00358A 063B 2A 04 0641        BPL    KU        BRANCH IF KEYUP
00359A 063D 8D 1D 065C        BSR    TIMER
00360A 063F 20 F2 0633        BRA    KDLOOP

00362A 0641 97 11    A KU     STAA   KDTIM     SAVE KD INTERVAL TIME
00363A 0643 81 04    A        CMPA   #$4       KD INTERVAL TIME TOO LOW?
00364A 0645 24 05 064C        BCC    CKHI      BRANCH IF NOT TOO LOW
00365A 0647 96 0C    A        LDAA   SPEEDK
00366A 0649 44                LSRA             DIVIDE SPEED CONSTANT BY 2
00367A 064A 20 07 0653        BRA    UNZERO

00369A 064C 81 7F    A CKHI   CMPA   #$7F      KD INTERVAL TIME TOO HIGH?
00370A 064E 25 07 0657        BCS    CMPTKD    BRANCH IF OK
00371A 0650 96 0C    A        LDAA   SPEEDK
```

```
00372A 0652 48                ASLA            MULTIPLY SPEED CONSTANT BY
00373A 0653 8A 01    A UNZERO ORAA   #1       ASCERTAIN SPEED CONSTANT I
00374A 0655 97 0C    A        STAA   SPEEDK   NOT SET TO ZERO
00375A 0657 8D 32 068B CMPTKD BSR    COMPKD
00376A 0659 7E 060D  A        JMP    KEYUP

00378                   ***SUBROUTINE TO CREATE TIME DELAY***
00379A 065C 37          TIMER  PSHB            SAVE B
00380A 065D 36                 PSHA            SAVE A
00381A 065E D6 0C    A         LDAB   SPEEDK
00382A 0660 86 40    A DELOP2  LDAA   #$40
00383A 0662 4A          DELOOP DECA
00384A 0663 26 FD 0662         BNE    DELOOP
00385A 0665 5A                 DECB
00386A 0666 26 F8 0660         BNE    DELOP2
00387A 0668 BD 06FB  A         JSR    PACH     PRINT CHAR IN BUFFER
00388A 066B 32                 PULA            RESTORE A
00389A 066C 33                 PULB            RESTORE B
00390A 066D 39                 RTS

00392                   ***SUBROUTINE TO COMPUTE KU***
00393A 066E 91 0F    A COMPKU CMPA   TQLDAT
00394A 0670 25 18 068A         BCS    MOREL    BRANCH IF KUTIM < TQLDAT
00395A 0672 96 0D    A         LDAA   RCHAR    GET CHAR BEING RECEIVED
00396A 0674 81 01    A         CMPA   #1
00397A 0676 27 08 0680         BEQ    CKFSP
00398A 0678 8D 67 06E1         BSR    GAFT     GET ASCII FROM TABLE
00399A 067A 8D 46 06C2         BSR    PRINT
00400A 067C 86 01    A         LDAA   #1
00401A 067E 97 0D    A         STAA   RCHAR    READY FOR NEW CHAR
00402A 0680 96 10    A CKFSP   LDAA   TLDAT    GET TWICE LAST DASH TIME
00403A 0682 91 12    A         CMPA   KUTIM    COMPARE WITH KU INTERVAL
00404A 0684 24 04 068A         BCC    MOREL    BRANCH IF TLDAT >= KU INTE
00405A 0686 86 20    A         LDAA   #$20     ASCII SPACE
00406A 0688 8D 38 06C2         BSR    PRINT    PRINT SPACE
00407A 068A 39          MOREL  RTS

00409                   ***SUBROUTINE TO COMPUTE KD***
00410A 068B 96 0B    A COMPKD  LDAA   TLETIM   GET TWICE LAST ELEM TIME
00411A 068D 91 11    A         CMPA   KDTIM    COMPARE WITH KD INTERVAL
00412A 068F 25 10 06A1         BCS    DASHEL   BRANCH IF TLETIM < KD INTE
00413A 0691 96 0A    A         LDAA   HLETIM   GET HALF LAST ELEM TIME
00414A 0693 91 11    A         CMPA   KDTIM    COMPARE WITH KD INTERVAL
00415A 0695 24 04 069B         BCC    DOTEL    BRANCH IF HLETIM >= KD INT
00416A 0697 96 09    A         LDAA   LETYPE   CHECK LAST ELEMENT TYPE
00417A 0699 26 06 06A1         BNE    DASHEL   BRANCH IF LAST ELEM WAS DA
00418A 069B 7F 0009  A DOTEL   CLR    LETYPE   MAKE LAST ELEM TYPE=00=DOT
00419A 069E 0C                 CLC
00420A 069F 20 14 06B5         BRA    ADDEL
00421A 06A1 7C 0009  A DASHEL  INC    LETYPE   MAKE LAST ELEM TYPE=DASH
00422A 06A4 96 11    A         LDAA   KDTIM    GET KD INTERVAL
00423A 06A6 97 0E    A         STAA   LDATIM   STORE IN LAST DASH TIME
00424A 06A8 16                 TAB             SAVE IN ACCUM B
```

```
00425A 06A9 44          LSRA            DIVIDE KD INTERVAL BY 2
00426A 06AA 97 0F    A  STAA  TQLDAT    SAVE 1/2 KD
00427A 06AC 44          LSRA            DIVIDE 1/2 KD BY 2
00428A 06AD 9B 0F    A  ADDA  TQLDAT    ADD 1/2 TO 1/4 KD INTERVAL
00429A 06AF 97 0F    A  STAA  TQLDAT    STORE RESULT
00430A 06B1 58          ASLB            MULTIPLY KD INTERVAL BY 2
00431A 06B2 D7 10    A  STAB  TLDAT     STORE RESULT
00432A 06B4 0D          SEC
00433A 06B5 79 000D  A ADDEL ROL  RCHAR   ADD NEW ELEM TO CHARACTER
00434A 06B8 96 11    A  LDAA  KDTIM     GET KD INTERVAL
00435A 06BA 16          TAB             SAVE IN ACCUM B
00436A 06BB 44          LSRA            DIVIDE KD BY 2
00437A 06BC 97 0A    A  STAA  HLETIM    STORE 1/2 KD INTERVAL
00438A 06BE 58          ASLB            MULTIPLY KD BY 2
00439A 06BF D7 0B    A  STAB  TLETIM    STORE TWICE KD INTERVAL
00440A 06C1 39          RTS

00442                  ***SUBROUTINE TO PRINT RCVD CHAR***
00443A 06C2 7C 0013  A PRINT  INC   CHCTR    INCREMENT CHAR COUNTER
00444A 06C5 D6 13    A  LDAB  CHCTR
00445A 06C7 C1 3C    A  CMPB  #60       CHECK CHAR COUNT
00446A 06C9 25 13 06DE  BCS   OK        BRANCH IF LESS THAN 60
00447A 06CB C1 47    A  CMPB  #71
00448A 06CD 25 0B 06DA  BCS   CKSP      BRANCH IF LESS THAN 71
00449A 06CF 36        ENDLIN PSHA       SAVE A
00450A 06D0 8D 66 0738    BSR   CRLFR   PRINT CR AND LF
00451A 06D2 7F 0013  A    CLR   CHCTR    RESET CHARACTER COUNTER
00452A 06D5 32            PULA          RESTORE A
00453A 06D6 81 20    A    CMPA  #$20     IS CHARACTER A SPACE?
00454A 06D8 27 06 06E0    BEQ   DONE     BRANCH IF YES
00455A 06DA 81 20    A CKSP   CMPA  #$20
00456A 06DC 27 F1 06CF    BEQ   ENDLIN   END THE LINE IF SPACE
00457A 06DE 20 2A 070A OK      BRA   SDATA    STORE CHARACTER
00458A 06E0 39        DONE     RTS

00460                  ***SUBR. TO GET ASCII CHAR FROM CODE TABLE***
00461A 06E1 0D         GAFT     SEC          CHANGE FORMAT OF RCHAR.
00462A 06E2 49                  ROLA
00463A 06E3 48         GAFT1    ASLA
00464A 06E4 24 FD 06E3          BCC   GAFT1
00465A 06E6 CE 045A  A          LDX   #RESRT-1
00466A 06E9 A1 00    A STAB1    CMPA  0,X
00467A 06EB 27 09 06F6          BEQ   TABM     FOUND MATCH
00468A 06ED 09                  DEX
00469A 06EE 8C 0421  A          CPX   #RTAB+1  END OF TABLE?
00470A 06F1 26 F6 06E9          BNE   STAB1    NO!
00471A 06F3 86 5F    A          LDAA  #'_      RETURN "_" FOR NO MATCH
00472A 06F5 39                  RTS
00473                    *
00474A 06F6 DF 14    A TABM     STX   RECX
00475A 06F8 96 15    A          LDAA  RECX+1   X(LOW) TO ACCA.
00476A 06FA 39                  RTS
```

forts.n.sida

```
PAGE  010   CMMK

00478                    ***SUBROUTINE TO PRINT 1 ASCII CHAR***
00479A 06FB F6 8010  A PACH   LDAB   ACIAC
00480A 06FE C5 02    A        BITB   #2
00481A 0700 27 07 0709        BEQ    PACH1    TST FOR TX REG. EMPTY.
00482A 0702 8D 1D 0721        BSR    GDATA
00483A 0704 24 03 0709        BCC    PACH1    EMPTY
00484A 0706 B7 8011  A        STAA   ACIAD    PRINT CHARACTER
00485A 0709 39        PACH1   RTS

00487                    ***SUBROUTINE TO STORE DATA IN REC BUFFER***
00488A 070A DE 18    A SDATA  LDX    P2
00489A 070C 9C 16    A        CPX    P1
00490A 070E 27 0F 071F        BEQ    BFUL
00491A 0710 8C A049  A        CPX    #BBOT-1
00492A 0713 26 03 0718        BNE    SDATA1
00493A 0715 CE A053  A        LDX    #BTOP
00494A 0718 A7 00    A SDATA1 STAA   0,X
00495A 071A 09                DEX
00496A 071B DF 18    A        STX    P2
00497A 071D 0C                CLC             NOT FULL
00498A 071E 39                RTS
00499                  *
00500A 071F 0D         BFUL   SEC             BUFFER FULL
00501A 0720 39                RTS

00503                    ***SUBROUTINE TO GET DATA FROM REC BUFFER***
00504A 0721 DE 16    A GDATA  LDX    P1
00505A 0723 09                DEX
00506A 0724 9C 18    A        CPX    P2
00507A 0726 27 0E 0736        BEQ    EMPTY
00508A 0728 8C A049  A        CPX    #BBOT-1
00509A 072B 26 03 0730        BNE    GDATA1
00510A 072D CE A053  A        LDX    #BTOP
00511A 0730 A6 00    A GDATA1 LDAA   0,X
00512A 0732 DF 16    A        STX    P1
00513A 0734 0D                SEC             NOT EMPTY
00514A 0735 39                RTS
00515                  *
00516A 0736 0C         EMPTY  CLC             EMPTY
00517A 0737 39                RTS

00519                    ***SUBROUTINE TO SEND CR/LF TO REC BUFFER***
00520A 0738 CE 07B1  A CRLFR  LDX    #CRLF
00521A 073B A6 00    A CRR    LDAA   0,X
00522A 073D 81 04    A        CMPA   #4
00523A 073F 27 09 074A        BEQ    CRR1
00524A 0741 DF 14    A        STX    RECX
00525A 0743 8D C5 070A        BSR    SDATA
00526A 0745 DE 14    A        LDX    RECX
00527A 0747 08                INX
00528A 0748 20 F1 073B        BRA    CRR
00529A 074A 39         CRR1   RTS
```

```
00531                    *** CHARACTER STRINGS ***
00532A 074B    4D    A MSG1   FCC    'MORSE CODE SEND/RECEIVE '
00533A 0763    50    A        FCC    'PROGRAM FOR MOTOROLA '
00534A 0778    36    A        FCC    '6800 MICROPROCESSOR'
00535A 078B    0D    A        FCB    $0D,$0A,0,0,0,0
00536A 0791    56    A        FCC    'VER. 2.0  ** W7KGU ** WA7VKZ **'
00537A 07B0    0A    A        FCB    $0A
00538A 07B1    0D    A CRLF   FCB    $0D,$0A,0,0,0,0,4

00540A 07F8                   ORG    CODE+$3F8
00541A 07F8    04E6  A        FDB    POLL,RESRT,RESRT,RESRT VECTORS
00542                         END
TOTAL ERRORS 00000
```

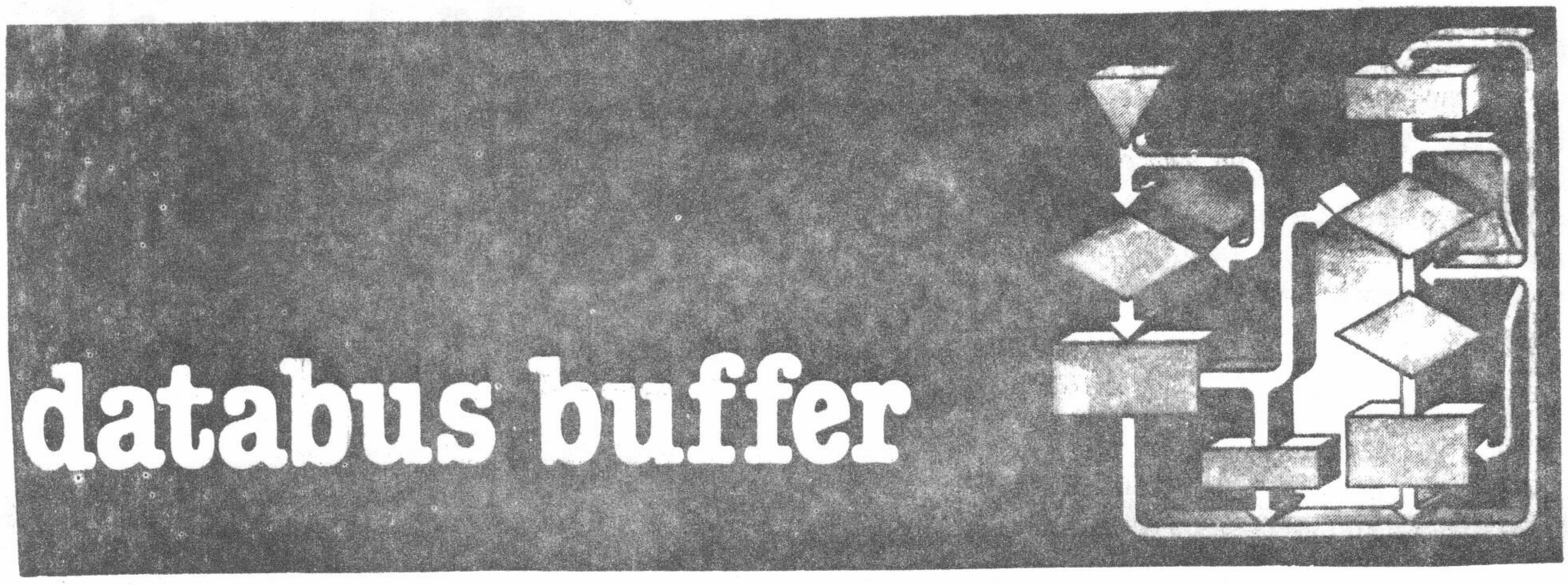
```
8010 ACIAC  00058*00151 00166 00479
8011 ACIAD  00059*00154 00290 00310 00484
06B5 ADDEL  00420 00433*
05FF BADCH  00298 00302 00325*
A04A BBOT   00050*00491 00508
071F BFUL   00490 00500*
A053 BTOP   00051*00097 00493 00510
001A BUFBOT 00047*00277 00278 00279 00291
05F7 BUFFUL 00292 00319*
0006 BUFLAG 00027*00127 00131 00133 00171 00247 00249 00307 00309
            00313 00315
007F BUFTOP 00048*00096 00103 00238 00245 00253 00260 00271 00277
0512 CES1   00183*
0538 CFCS   00202 00208*
0515 CFDAH  00180 00186*
0520 CFES   00187 00193*
052D CFWS   00194 00201*
0007 CHCNT  00029*00134 00142 00289 00316
0013 CHCTR  00043*00443 00444 00451
0546 CK1    00212 00216*
053E CKCNT  00184 00191 00199 00206 00211*
0680 CKFSP  00397 00402*
064C CKHI   00364 00369*
06DA CKSP   00448 00455*
0657 CMPTKD 00370 00375*
0400 CODE   00065*00101 00540
05BA COMMR  00235 00288*
0565 COMMR1 00168 00235*
068B COMPKD 00375 00410*
066E COMPKU 00343 00352 00393*
0008 COSTA  00030*00179 00217 00219 00242 00256 00266 00273
0004 COUNT  00025*00211 00213 00216
07B1 CRLF   00147 00520 00538*
0738 CRLFR  00450 00520*
073B CRR    00521*00528
074A CRR1   00523 00529*
0000 CVCX   00023*00102 00299 00300
```

forts.n.sida

```
06A1 DASHEL 00412 00417 00421*
0662 DELOOP 00383*00384
0660 DELOP2 00382*00386
06E0 DONE   00454 00458*
069B DOTEL  00415 00418*
0736 EMPTY  00507 00516*
06CF ENDLIN 00449*00456
05FB ERROR  00294 00322*
04CC EX1    00136 00140 00143*
04B9 EX2    00129 00134*
04AC EXEC   00127*00144
06E1 GAFT   00398 00461*
06E3 GAFT1  00463*00464
0721 GDATA  00482 00504*
0730 GDATA1 00509 00511*
0567 GNEL   00221 00238*
058F GNEL1  00240 00257 00260*
0581 GNELR  00246 00252*
000A HLETIM 00034*00413 00437
A000 IRQVEC 00061*00106
062D KD     00345 00351*
0633 KDLOOP 00354*00360
0011 KDTIM  00041*00362 00411 00414 00422 00434
0631 KEYDWN 00353*
060D KEYUP  00333*00376
0641 KU     00358 00362*
0620 KUCONT 00337 00342 00344*
060F KULOOP 00334*00348
0012 KUTIM  00042*00339 00351 00403
045E L1     00090*00092
000E LDATIM 00038*00423
0009 LETYPE 00033*00416 00418 00421
0638 MAXKD  00355 00357*
068A MOREL  00394 00404 00407*
05AD MOV1   00278*00281
05B9 MOVRT  00285*
05AA MOVUP  00252 00277*
074B MSG1   00122 00532*
0616 NOINC  00335 00339*
06DE OK     00446 00457*
0016 P1     00045*00098 00489 00504 00512
0018 P2     00046*00100 00488 00496 00506
06FB PACH   00387 00479*
0709 PACH1  00481 00483 00485*
8009 PIAAC  00054*00160
8008 PIAAD  00053*00107 00161 00344 00357
800B PIABC  00056*
800A PIABD  00055*00175 00178 00224 00226 00230 00232 00329 00332
04E6 POLL   00105 00160*00349 00541
04EF POLL1  00164*00172
0500 POLL2  00165 00175*
06C2 PRINT  00399 00406 00443*
05E8 PRNT   00310*00320 00323 00326
```

```
000D RCHAR  00037*00340 00395 00401 00433
0603 REC    00173 00329*
04FD RECI   00173*
04F9 RECM   00163 00171*
0014 RECX   00044*00474 00475 00524 00526
0005 RESMSK 00026*00182 00189 00196 00204 00209 00218
045B RESRT  00089*00465 00541 00541 00541
0545 RETRN  00214*00220
0420 RTAB   00072*00469
04E5 RTSR   00150 00157*
0002 SAVEX  00024*00104 00137 00244 00282 00284 00288 00303 00306
05EC SCR    00296 00313*
04D8 SCR1   00151*00153
04CF SCRLF  00121 00130 00141 00147*
04D2 SCS    00123 00148*00156
070A SDATA  00457 00488*00525
0718 SDATA1 00492 00494*
0598 SODAH  00261 00265*
059A SOEL   00263 00266*
055C SOONE  00230*00267
0553 SOZERO 00197 00224*00243 00274
000C SPEEDK 00036*00094 00365 00371 00374 00381
06E9 STAB1  00466*00470
A07F STACK  00060*00120
05D3 STM    00299*00317
059F SWS    00255 00270*
06F6 TABM   00467 00474*
065C TIMER  00346 00359 00379*
0010 TLDAT  00040*00402 00431
000B TLETIM 00035*00410 00439
000F TQLDAT 00039*00393 00426 00428 00429
0653 UNZERO 00367 00373*
```

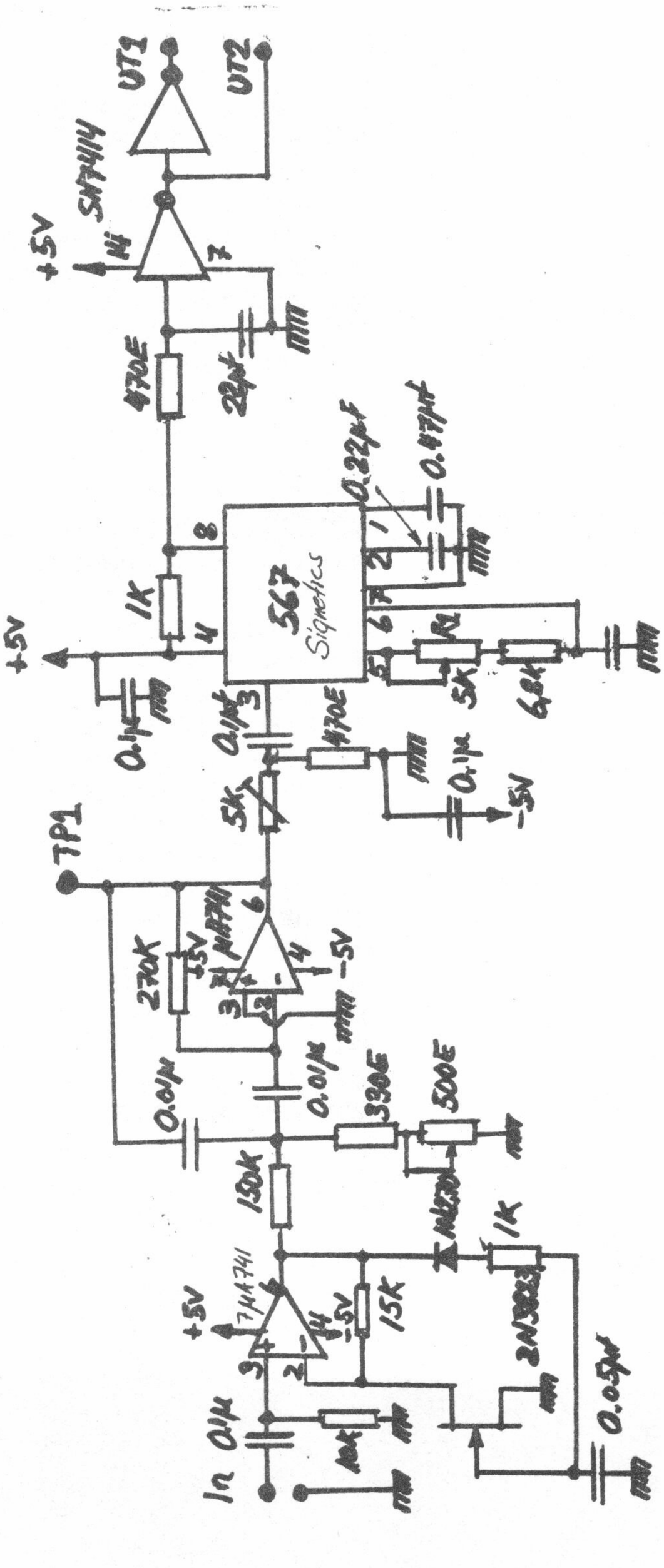
forts.n.sida



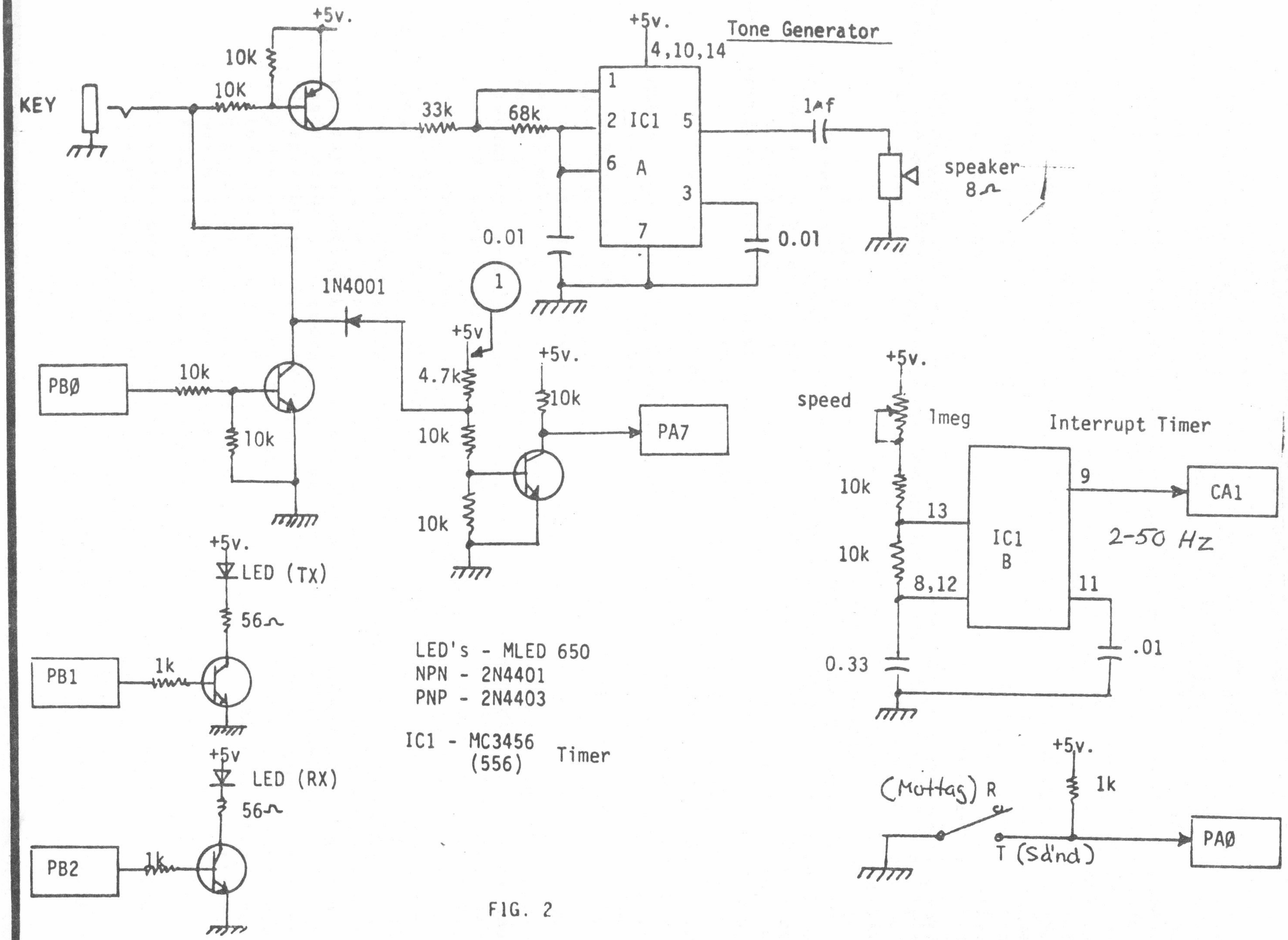
I samband med det program som finns beskrivet på annan plats i detta nummer för morseomvandling, kan kanske följande vara av intresse. Det är en omvandlare från LF-signaler till TTl-nivåer, delvis beskriven i Popular Electronics mars nummer. Detta behövs för datorns interface mot omgivningen.

Konstruktionen är gjord på följande sätt:
Den första OP-förstärkaren tillsammans med FET-transistorn fungerar som begränsare.
Nästa OP-ic är ett bandpassfilter med centerfrekvens 1200 Hz.
Därpå följer PLL-kretsen som demodulator. Demodulatorn är genom R1 inställd till 1200 Hz.
Den färdiga signalen kan sedan erhållas inv./icke inv. på de båda utgångarna. För att radion skall fungera vid sändningsklass A1 erfodras att radion har BFO.*

Någon samkörning av hårdvara-mjukvara har tyvärr inte varit möjlig, då tiden inte medgivit detta. Vi är därför intresserade av att de som bygger detta hör av sig med synpunkter. Vi redovisar resultatet i nästa nummer.
Att konstruktionen i sig fungerar är dock givetvis kontrollerat. Problem kan alltså endast uppstå i gränssnittet.



* Sändningsklass A1 är telegrafi utan modulering, alltså endast bärvåg till-från. Genom att inkoppla en BFO (beat-frekvens-osc.) erhålls genom interferens en hörbar ton. Denna kan också demoduleras av PLL-kretsen.



LED's - MLED 650
NPN - 2N4401
PNP - 2N4403

IC1 - MC3456 Timer
(556)

FIG. 2

Ascii–Baudot–Ascii

Förutom morse programmet har vi ännu en godbit för radiopularna. Det är ett program från User Group Library som heter CODEC. Det består av två macrorutiner och ett testprogram. De två macrorutinerna kan omvandla ASCII till Baudot resp. Baudot till ASCII. I testprogrammet visar man hur macrorutinerna används till att ta in ASCII från ett tangentbord, lagra dem i en buffert, därefter omvandla dem till Baudot tecken som i sin tur ånyo omvandlas till ASCII och slutligen skrivs ut på terminalen. Någon anpassning till yttre världen för mottagningen och sändningen av Baudot tecknen är inte beskriven. En parameter till macrorutinen CONBA är den adress från vilken man kan hämta ett mottaget Baudot tecken. Ut kommer sedan ASCII i A-acc.

Den hårdvara som behövs för att omvandla tonskiftskoden i etern till 5-bits Baudot har vi inte tillgänglig. Där hoppas vi faktiskt på lite respons från våra radioamatörer. När vi har något bra att visa kommer det i MPU-laren.

Vad är förresten en macro.? Det kanske är många som inte har sett sådana tidigare. Först och främst behöver man en macroassembler. Exempelvis Motorolas RASM under MDOS. Den kräver minimum 24k i RAM. En macro kan ses som en förlängning av språket. Med dess hjälp kan man få in rutiner i sitt program i vilka storheter tilldelas värden beroende på hur anropet av macro:n ser ut. Tag som exempel rutinen CONBA. I den skall användas följande information:

- ADDR adress att hämta Baudot tecken på.
- SHIFT adress till RAM cell för flaggning om Baudot siffra eller bokstav
- XLOAC adress till 2-bytes area där adress till omvandlingstabellen ligger.

I macrokroppen CONBA ser vi en mängd ställen där bokstaven "Ö" föregår ett annat värde. Detta "Ö" är egentligen en back-slash "\". Med svensk teckengenerator blir det ett Ö i stället. Nåväl, om det efter ö:et står en numeral betyder det att en parameter från macroanropet skall insättas därefter. Numeralen är helt enkelt ordningsnumret i parameterlistan. Exemplet i programmet:

```
00310A B609      CONBA X,$3,$4
```

betyder att överallt där det nu står Ö1 skall det insättas X, där det står Ö2 skall det i stället bli $3 osv.

```
  LDAA  Ö1 blir LDAA  X
  STAA  Ö3+1 blir STAA $4+1
```

Där det står exempelvis .Ö4 som på rad 88 BRA .Ö4 skall vid assembleringen ett unikt programläge utläggas.

```
  BRA  .Ö4 blir nu BRA .00009
```

Med option kan man slippa se den expanderade koden om så önskas. En macro skulle kunna skrivas för att ersätta den i M6800 saknade instruktionen PSHX. I sin kod skriver man sedan endast PSHX när man vill ha X-reg ut på stacken. Macrogeneratorn lägger då ut den förutbestämda koden.

Det här var egentligen en utvikning från det ursprungliga ämnet ASCII/Baudot. Men den kanske underlättar tolkningen av listorna. I texten står lite om hur man bär sig åt vid flyttning av koden. Exempelvis måste högsta adressbyten till omvandlingstabellen inknappas i CONBA och CONAB. Se rad 33-34. För övrigt skall lägre adressen alltid börja på 00. Den motsvaras som i morseprogrammet direkt av ett tecken.

Till ett senare nummer av MPU-laren kan vi bearbeta program och hårdvara mera.

TB1

```
PAGE  001   CODEC  *CONVERTS ASCII-BAUDOT VER1.0*

00001                       NAM    CODEC
00002                       TTL    *CONVERTS ASCII-BAUDOT VER1.0*
00003                       OPT    SYM,MEX,LLEN=80


00005                 *
00006                 *
00007                 *NAME:      JOHN HOYT
00008                 *COMP:      NAVELEX SECURITY ENG CENTER
00009                 *ADDR:      3801 NEBRASKA AVE NW
00010                 *           WASHINGTON, DC 20390
00011                 *PHON:      (202)282-0938
00012                 *
00013                 *
00014                 *
00015                 *THESE ARE BAUDOT TO ASCII & ASCII TO
00016                 *BAUDOT MACRO'S.
00017                 *
00018                 *PRIME USE IS IN INTRUP I/O ROUTINES TO
00019                 *CONVERT FRM/TO BAUDOT TERMINAL DEVICES
00020                 *FRM/TO ASCII TERMINAL DEVICES.
00021                 *
00022                 *LOWER CASE ASCII WILL BE CONVERTED TO
00023                 *UPPER CASE & ASCII CHARS THAT HAVE NO
00024                 *CORESPONDING BAUDOT CHAR WILL BE
00025                 *CONVERTED TO NULL'S (IE-ALL 0'S CHAR).
00026                 *
00027                 *LK-UP TABLE TAKES UP 128 BYTES OF (ROM) MEM.
00028                 *
00029                 *MAX SERVICE TIME IS 67 MC6800 CLK CYCLES.
00030                 *
00031                 *TO MOVE LK-UP TBL IN (ROM) MEMORY,
00032                 *CHANGE THE FOLLOWING:
00033                 *IN CONBA---ö.3 LDAB #$(NEW)
00034                 *IN CONAB---ö.4 LDAB #$(NEW)
00035                 *IN TABLE---ORG $(NEW)00
00036                 *WHERE (NEW) = 1ST 2 HEX CHARS OF NEW TBL LOAC.
00037                 *NOTE: TABLE MUST START @ ADDR ENDING IN 00!!!!!
00038                 *
00039                 *CHIPS OTHER THAN THE ACIA ARE NORMALLY REQUIRED
00040                 *FOR BAUDOT (5 BIT CODE) USE.
00041                 *
00042                 *
00043                 *
00044                 *EXAMPLE PROGRAM CONVERTS ONE LINE FRM THE
00045                 *CONSOLE KYBD TO BAUDOT STORED @ MEM ADDR $100
00046                 *ON, AND RECONVERTS THIS LINE BACK TO ASCII AT
00047                 *THE CONSOLE PRNTR AFTER A CR IS SENT. IF THE
00048                 *CONSOLE IS A TI 733 ASR THE EXAMPLE MUST BE
00049                 *RUN @ 300 BAUD ON THE EXORCISER AS NO FILL
00050                 *CHARS ARE SENT TO THE PRNTR. EXMPL NEEDS
00051                 *LESS THAN 1K RAM STRT'G @ $0000 & LESS THAN
00052                 *1K STRT'G @ $B500 TO RUN.
```

forts.n.sida

PAGE 002 CODEC *CONVERTS ASCII-BAUDOT VER1.0*

```
00054                   *MACRO CONBA - CONVERTS BAUDOT CHAR TO ASCII
00055                   *USE IS AS FOLLOWS: CONBA ADDR,SHIFT,XLOAC
00056                   *WHERE: ADDR=ADDR OF BAUDOT CHAR
00057                   *       SHIFT=LOCATION IN RAM(1 BYTE)INDICATING
00058                   *             IF FIGS OR LTTR BAUDOT CHAR.
00059                   *       XLOAC=2 BYTE LOCA IN RAM FOR LK
00060                   *             UP TABLE ADDR.
00061                   *
00062                   *IF A BAUDOT SHIFT CHAR IS INPUTED, THE
00063                   *A-ACMLTR = 0 AT END OF MACRO.
00064                   *
00065                   *SERVICE TIME RANGES FRM 10 CLK CYCLES TO DECODE
00066                   *A NULL TO 52 CLK CYCLES TO DECODE
00067                   *AN ASCII UPPER CASE LTR.
00068                   *
00069                   CONBA  MACR
00070                    LDAA ö0 ;GET BAUDOT CHAR
00071                    BEQ ö.4 ;GOTO ö.4 IF=0
00072                    CMPA #$1F ;LTRS?
00073                    BEQ ö.1
00074                    CMPA #$1B ;FIGS?
00075                    BEQ ö.2 ;IF SO GOTO ö.2
00076                    LDAB ö1 ;SEE IF LTRS OR FIGS
00077                    BEQ ö.3 ;IF FIGS GOTO ö.3
00078                    ORAA #$60 ;SET BITS 5 & 6 IF LTRS
00079                   ö.3 LDAB #$B5
00080                    STAB ö2
00081                    STAA ö2+1
00082                    LDX ö2
00083                    LDAA X ;ASCII CHAR
00084                    BRA ö.4 ;GOTO END
00085                   ö.1 LDAB #$FF ;SET ö1 TO $FF IF LTRS
00086                    STAB ö1
00087                    CLRA
00088                    BRA ö.4
00089                   ö.2 CLR ö1 ;SET ö1 TO $00 IF FIGS
00090                    CLRA
00091                   ö.4 EQU *
00092                    ENDM
```

```
00095                *MACRO CONAB - CONVERTS 7-BIT ASCII CHAR TO BAUDOT
00096                *USE IS AS FOLLOWS: CONAB ADDR,SHIFT,XLOAC
00097                *WHERE: ADDR=ADDR OF ASCII CHAR
00098                *       SHIFT=1 BYTE RAM LOCA TO INDICATE IF
00099                *             THE BAUDOT CHAR IS FIGS OR LTRS
00100                *       XLOAC=2 BYTE RAM LOAC FOR LK-UP TBL
00101                *
00102                *IF A BAUDOT SHIFT CHAR MUST BE SENT BOTH A & B
00103                *ACMLTRS WILL COMPARE & BOTH WILL HAVE THE SHIFT
00104                *CHAR THAT MUST BE SENT. THEN THIS MACRO MUST
00105                *BE RECALLED TO SEND THE BAUDOT CHAR.
00106                *
00107                *ANY ASCII CONTRL CHAR'S EXCEPT LF CR OR
00108                *BELL WILL BE DECODED AS NULL'S.
00109                *
00110                *THIS MACRO WILL TAKE BOTH LOWER AND UPPER CASE
00111                *ASCII CHAR'S AND CONVERT THEM TO UPPER CASE ON A
00112                *BAUDOT PRINTER (IE - a CONVERTS TO A).
00113                *
00114                *SERVICE TIMES RANGE FRM 10 CLK CYCLES TO DECODE
00115                *A NULL TO 67 CLK CYCLES TO DECODE A BAUDOT
00116                *FIGS SHIFT CHAR PRIOR TO A BAUDOT BELL CHAR.
00117                *
00118                CONAB  MACR
00119                 LDAA ö0 ;GET ASCII CHAR
00120                 BEQ ö.5
00121                 CMPA #$7F
00122                 BEQ ö.5
00123                 BITA #$40 ;LTRS?
00124                 BNE ö.2 ;IF SO GOTO ö.2
00125                 BITA #$60 ;CNTL CHAR?
00126                 BEQ ö.3 ;IF SO GOTO ö.3
00127                 CMPA #$20 ;SP?
00128                 BEQ ö.4
00129                 LDAB #$1B ;FIGS
00130                 CMPB ö1 ;SHIFT LOCATION IN RAM
00131                 BEQ ö.4
00132                ö.6 STAB ö1
00133                 TBA ;B=A
00134                 BRA ö.5 ;GOTO END
00135                ö.2 LDAB #$1F ;LTRS
00136                 CMPB ö1
00137                 BNE ö.6 ;SHIFT CHANGE (LTRS)
00138                 ORAA #$20 ;CONVERT LOWER CASE
00139                 EORA #$20 ;TO UPPER CASE
00140                ö.4 LDAB #$B5
00141                 STAB ö2
00142                 STAA ö2+1
00143                 LDX ö2
00144                 LDAA X ;BAUDOT CHAR
00145                 BRA ö.5 ;GOTO END
00146                ö.3 CMPA #$0A ;LF?
00147                 BNE ö.7
```



```
00148                 LDAA #$02 ;BAUDOT LF
00149                 BRA ö.5
00150                ö.7 CMPA #$0D ;CR?
00151                 BNE ö.8
00152                 LDAA #$08 ;BAUDOT CR
00153                 BRA ö.5
00154                ö.8 CMPA #$07 ;BELL?
00155                 BNE ö.9
00156                 LDAB #$1B ;FIGS
00157                 CMPB ö1
00158                 BEQ ö.A
00159                 BRA ö.6
00160                ö.A LDAA #$05 ;BAUDOT BELL
00161                 BRA ö.5
00162                ö.9 CLRA
00163                ö.5 EQU *
00164                 ENDM
```

forts.n.sida

```
PAGE  005   CODEC  *CONVERTS ASCII-BAUDOT VER1.0*

00167A B500                    ORG    $B500
00168                  *LOOK-UP TBL FOR ASCII-BAUDOT CONVERSION
00169                  *CHANGE THE ABOVE ORG STATEMENT TO MOVE
00170                  *LAST TWO DIGITS MUST BE 00. TABLE TAKES
00171                  *UP 128 BYTES OF ROM.


00173                  *BAUDOT FIGURES TO ASCII SECTION
00174A B500    00    A         FCB    $0
00175A B501    33    A         FCC    ö3ö
00176A B502    0A    A         FCB    $0A       ;LF
00177A B503    2D    A         FCC    ö- ö
00178A B505    07    A         FCB    $07       ;BELL
00179A B506    38    A         FCC    ö87ö
00180A B508    0D    A         FCB    $0D       ;CR
00181A B509    24    A         FCC    ö$4',!:(5")2#6019?&ö
00182A B51B    00    A         FCB    $0
00183A B51C    2E    A         FCC    ö./;ö
00184A B51F    00    A         FCB    $0


00186                    *ASCII TO BAUDOT FIGS SECTION
00187A B520    04    A         FCB    $04       ;SP
00188A B521    0D    A         FCB    $0D       ;!
00189A B522    11    A         FCB    $11       ;"
00190A B523    14    A         FCB    $14       ;#
00191A B524    09    A         FCB    $09       ;$
00192A B525    00    A         FCB    $00
00193A B526    1A    A         FCB    $1A       ;&
00194A B527    0B    A         FCB    $0B       ;'
00195A B528    0F    A         FCB    $0F       ;(
00196A B529    12    A         FCB    $12       ;)
00197A B52A    00    A         FCB    $0
00198A B52B    00    A         FCB    $0
00199A B52C    0C    A         FCB    $0C       ;,
00200A B52D    03    A         FCB    $03       ;-
00201A B52E    1C    A         FCB    $1C       ;.
00202A B52F    1D    A         FCB    $1D       ;/
00203A B530    16    A         FCB    $16       ;0
00204A B531    17    A         FCB    $17       ;1
00205A B532    13    A         FCB    $13       ;2
00206A B533    01    A         FCB    $01       ;3
00207A B534    0A    A         FCB    $0A       ;4
00208A B535    10    A         FCB    $10       ;5
00209A B536    15    A         FCB    $15       ;6
00210A B537    07    A         FCB    $07       ;7
00211A B538    06    A         FCB    $06       ;8
00212A B539    18    A         FCB    $18       ;9
00213A B53A    0E    A         FCB    $0E       ;:
00214A B53B    1E    A         FCB    $1E       ;;
00215A B53C    00    A         FCB    $0
00216A B53D    00    A         FCB    $0
00217A B53E    00    A         FCB    $0
```

```
00218A B53F    19    A       FCB    $19      ;?


00220                  *UPPER CASE ASCII TO BAUDOT LTRS SECTION
00221A B540    00    A       FCB    $0
00222A B541    03    A       FCB    $03      ;A
00223A B542    19    A       FCB    $19      ;B
00224A B543    0E    A       FCB    $0E      ;C
00225A B544    09    A       FCB    $09      ;D
00226A B545    01    A       FCB    $01      ;E
00227A B546    0D    A       FCB    $0D      ;F
00228A B547    1A    A       FCB    $1A      ;G
00229A B548    14    A       FCB    $14      ;H
00230A B549    06    A       FCB    $06      ;I
00231A B54A    0B    A       FCB    $0B      ;J
00232A B54B    0F    A       FCB    $0F      ;K
00233A B54C    12    A       FCB    $12      ;L
00234A B54D    1C    A       FCB    $1C      ;M
00235A B54E    0C    A       FCB    $0C      ;N
00236A B54F    18    A       FCB    $18      ;O
00237A B550    16    A       FCB    $16      ;P
00238A B551    17    A       FCB    $17      ;Q
00239A B552    0A    A       FCB    $0A      ;R
00240A B553    05    A       FCB    $05      ;S
00241A B554    10    A       FCB    $10      ;T
00242A B555    07    A       FCB    $07      ;U
00243A B556    1E    A       FCB    $1E      ;V
00244A B557    13    A       FCB    $13      ;W
00245A B558    1D    A       FCB    $1D      ;X
00246A B559    15    A       FCB    $15      ;Y
00247A B55A    11    A       FCB    $11      ;Z
00248A B55B    00    A       FCB    $0
00249A B55C    00    A       FCB    $0
00250A B55D    00    A       FCB    $0
00251A B55E    00    A       FCB    $0
00252A B55F    00    A       FCB    $0


00254                  *BAUDOT LTRS TO UPPER CASE ASCII SECTION
00255A B560    00    A       FCB    $0
00256A B561    45    A       FCC    öEö
00257A B562    0A    A       FCB    $0A      ;LF
00258A B563    41    A       FCC    öA SIUö
00259A B568    0D    A       FCB    $0D      ;CR
00260A B569    44    A       FCC    öDRJNFCKTZLWHYPQOBGö
00261A B57B    00    A       FCB    $0
00262A B57C    4D    A       FCC    öMXVö
00263A B57F    00    A       FCB    $0


00265                  *END OF TABLE
```

forts.n.sida

```
PAGE  007   CODEC  *CONVERTS ASCII-BAUDOT VER1.0*

00267                    *EXAMPLE PRGM TO SHOW USE OF ASCII-BAUDOT
00268                    *CONVERSION MACRO'S.
00269                    *
00270                    *REQUIRES <1K RAM STRT'G @ $0000
00271                    *AND <1K STRT'G @ $B500 TO RUN.
00272                    *
00273                    *TAKES IN ONE LINE FRM CONSOLE KYBD AND
00274                    *CONVERTS TO BAUDOT STORED AT MEM LOAC $100 ON.
00275                    *UPON RECEVING A CR, CONVERTS BAUDOT BACK
00276                    *TO ASCII & SENDS TO CONSOLE PRNTR.
00277                    *PRGM WILL THEN TAKE IN A NEW LINE, WRIT'G
00278                    *OVER THE OLD LINE.
00279                    *
00280                    *MUST USE @ 300 BAUD IF CONSOLE
00281                    *IS A TI 733 ASR (NO FILL CHARS)!!
00282                    *


00284A B580                      ORG    $B580
00285A B580 8E 07FF  A           LDS    #$7FF
00286A B583 7F 0000  A           CLR    $0
00287A B586 7F 0003  A           CLR    $3
00288A B589 86 01    A           LDAA   #$1       ;NO INT,7BIT,EP,2SB,X16CLK
00289A B58B B7 FCF4  A           STAA   $FCF4
00290A B58E CE 0100  A L3        LDX    #$100
00291A B591 DF 06    A           STX    $6
00292A B593 F6 FCF4  A STRT      LDAB   $FCF4     ;ACIA CR
00293A B596 C5 01    A           BITB   #1        ;REC DR FULL?
00294A B598 27 F9 B593           BEQ    STRT
00295          B59A  A L1        EQU    *
00296A B59A                      CONAB  $FCF5,$0,$1
     A B59A B6 FCF5  A           LDAA   $FCF5     ;GET ASCII CHAR
     A B59D 27 52 B5F1           BEQ    .00000
     A B59F 81 7F    A           CMPA   #$7F
     A B5A1 27 4E B5F1           BEQ    .00000
     A B5A3 85 40    A           BITA   #$40      ;LTRS?
     A B5A5 26 13 B5BA           BNE    .00001    ;IF SO GOTO .00001
     A B5A7 85 60    A           BITA   #$60      ;CNTL CHAR?
     A B5A9 27 25 B5D0           BEQ    .00002    ;IF SO GOTO .00002
     A B5AB 81 20    A           CMPA   #$20      ;SP?
     A B5AD 27 15 B5C4           BEQ    .00003
     A B5AF C6 1B    A           LDAB   #$1B      ;FIGS
     A B5B1 D1 00    A           CMPB   $0        ;SHIFT LOCATION IN RAM
     A B5B3 27 0F B5C4           BEQ    .00003
     A B5B5 D7 00    A .00004    STAB   $0
     A B5B7 17                   TBA              ;B=A
     A B5B8 20 37 B5F1           BRA    .00000    ;GOTO END
     A B5BA C6 1F    A .00001    LDAB   #$1F      ;LTRS
     A B5BC D1 00    A           CMPB   $0
     A B5BE 26 F5 B5B5           BNE    .00004    ;SHIFT CHANGE (LTRS)
     A B5C0 8A 20    A           ORAA   #$20      ;CONVERT LOWER CASE
     A B5C2 88 20    A           EORA   #$20      ;TO UPPER CASE
     A B5C4 C6 B5    A .00003    LDAB   #$B5
```

```
      A B5C6 D7 01    A        STAB   $1
      A B5C8 97 02    A        STAA   $1+1
      A B5CA DE 01    A        LDX    $1
      A B5CC A6 00    A        LDAA   X        ;BAUDOT CHAR
      A B5CE 20 21 B5F1        BRA    .00000   ;GOTO END
      A B5D0 81 0A    A .00002 CMPA   #$0A     ;LF?
      A B5D2 26 04 B5D8        BNE    .00005
      A B5D4 86 02    A        LDAA   #$02     ;BAUDOT LF
      A B5D6 20 19 B5F1        BRA    .00000
      A B5D8 81 0D    A .00005 CMPA   #$0D     ;CR?
      A B5DA 26 04 B5E0        BNE    .00006
      A B5DC 86 08    A        LDAA   #$08     ;BAUDOT CR
      A B5DE 20 11 B5F1        BRA    .00000
      A B5E0 81 07    A .00006 CMPA   #$07     ;BELL?
      A B5E2 26 0C B5F0        BNE    .00007
      A B5E4 C6 1B    A        LDAB   #$1B     ;FIGS
      A B5E6 D1 00    A        CMPB   $0
      A B5E8 27 02 B5EC        BEQ    .00008
      A B5EA 20 C9 B5B5        BRA    .00004
      A B5EC 86 05    A .00008 LDAA   #$05     ;BAUDOT BELL
      A B5EE 20 01 B5F1        BRA    .00000
      A B5F0 4F        .00007 CLRA
           B5F1  A .00000 EQU   *
00297A B5F1 DE 06    A        LDX    $6
00298A B5F3 A7 00    A        STAA   X
00299A B5F5 08                INX
00300A B5F6 DF 06    A        STX    $6
00301A B5F8 11                CBA
00302A B5F9 27 9F B59A        BEQ    L1
00303A B5FB 81 08    A        CMPA   #$08     ;BAUDOT CR
00304A B5FD 26 94 B593        BNE    STRT
00305A B5FF CE 00FF  A        LDX    #$0FF
00306A B602 DF 08    A        STX    $8
00307A B604 DE 08    A L2     LDX    $8
00308A B606 08                INX
00309A B607 DF 08    A        STX    $8
00310A B609                   CONBA  X,$3,$4
      A B609 A6 00    A        LDAA   X        ;GET BAUDOT CHAR
      A B60B 27 25 B632        BEQ    .00009   ;GOTO .00009 IF=0
      A B60D 81 1F    A        CMPA   #$1F     ;LTRS?
      A B60F 27 16 B627        BEQ    .00010
      A B611 81 1B    A        CMPA   #$1B     ;FIGS?
      A B613 27 19 B62E        BEQ    .00011   ;IF SO GOTO .00011
      A B615 D6 03    A        LDAB   $3       ;SEE IF LTRS OR FIGS
      A B617 27 02 B61B        BEQ    .00012   ;IF FIGS GOTO .00012
      A B619 8A 60    A        ORAA   #$60     ;SET BITS 5 & 6 IF LTRS
      A B61B C6 B5    A .00012 LDAB   #$B5
      A B61D D7 04    A        STAB   $4
      A B61F 97 05    A        STAA   $4+1
      A B621 DE 04    A        LDX    $4
      A B623 A6 00    A        LDAA   X        ;ASCII CHAR
      A B625 20 0B B632        BRA    .00009   ;GOTO END
      A B627 C6 FF    A .00010 LDAB   #$FF     ;SET $3 TO $FF IF LTRS
```

```
     A B629 D7 03    A        STAB   $3
     A B62B 4F                CLRA
     A B62C 20 04 B632        BRA    .00009
     A B62E 7F 0003  A .00011 CLR    $3        ;SET $3 TO $00 IF FIGS
     A B631 4F                CLRA
              B632   A .00009 EQU    *
00311A B632 27 D0 B604        BEQ    L2        ;SHIFT CHAR
00312A B634 F6 FCF4  A SND    LDAB   $FCF4
00313A B637 C5 02    A        BITB   #2        ;XMIT RDY?
00314A B639 27 F9 B634        BEQ    SND
00315A B63B B7 FCF5  A        STAA   $FCF5
00316A B63E 81 0D    A        CMPA   #$D       ;CR?
00317A B640 26 C2 B604        BNE    L2
00318A B642 7E B58E  A        JMP    L3


00320                         END
TOTAL ERRORS 00000

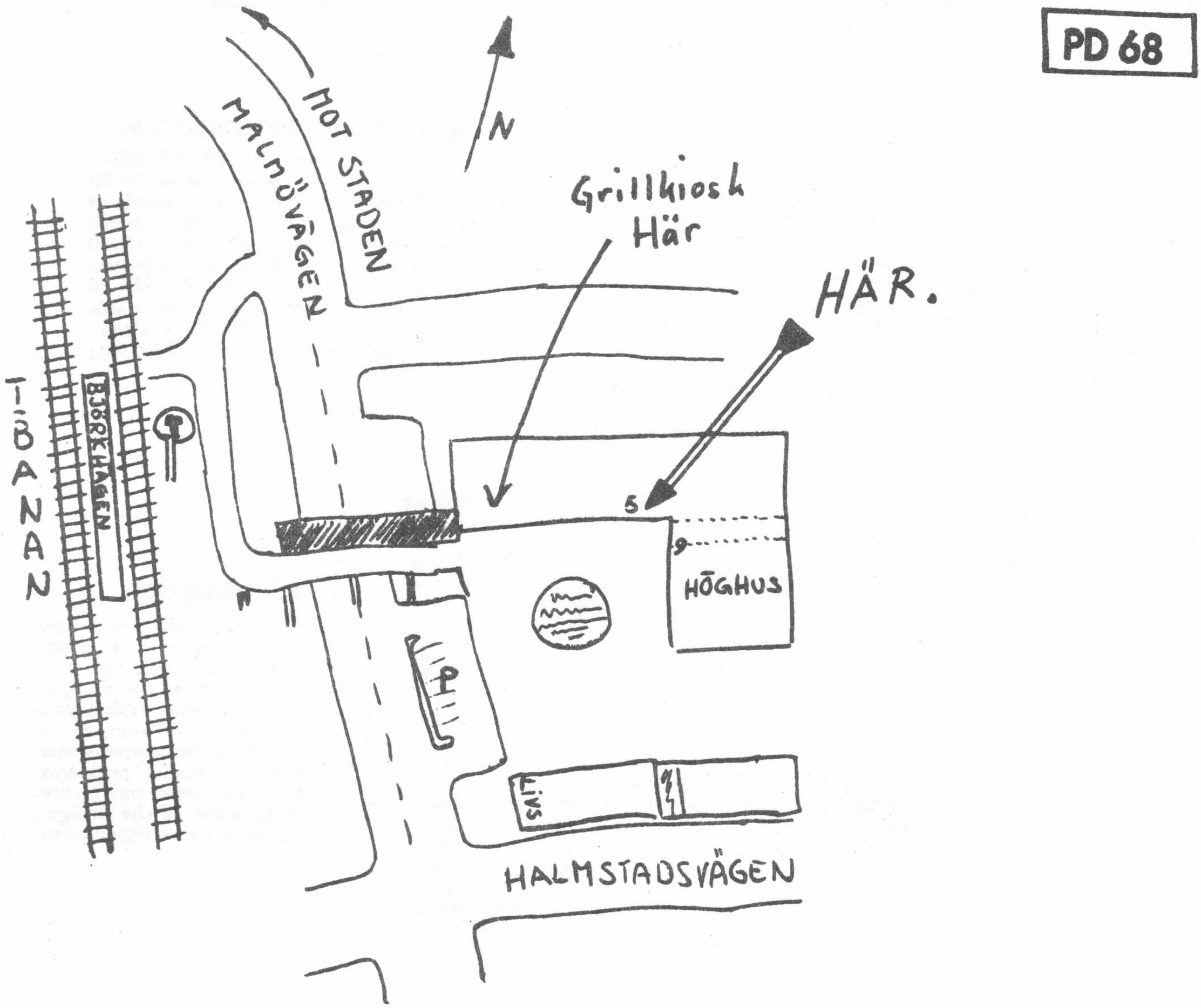

   B5F1 .00000 00296 00296 00296 00296 00296 00296 00296 00296*
   B5BA .00001 00296 00296*
   B5D0 .00002 00296 00296*
   B5C4 .00003 00296 00296 00296*
   B5B5 .00004 00296*00296 00296
   B5D8 .00005 00296 00296*
   B5E0 .00006 00296 00296*
   B5F0 .00007 00296 00296*
   B5EC .00008 00296 00296*
   B632 .00009 00310 00310 00310 00310*
   B627 .00010 00310 00310*
   B62E .00011 00310 00310*
   B61B .00012 00310 00310*
   B59A L1     00295*00302
   B604 L2     00307*00311 00317
   B58E L3     00290*00318
   B634 SND    00312*00314
   B593 STRT   00292*00294 00304
```



FÅR VI PÅMINNA OM VÅRA KLUBBMÖTEN.....

Torsdagen den 30 Augusti klockan 18.30 börjar våra klubbmöten.
Vi har bokat tid för Sept,Okt och November,sista torsdagen i
varje månad altså den 27/9,25/10,29/11. Adressen är Björkhagsplan 5..
Tunnelbanestation Björkhagen via linje 17..Gott om P-platser för
bilburna och korvkiosk i porten bredvid om middagen skulle råka
komma i kläm.

PD 68



TSC BASIC FOR THE 6800

Technical Systems Consultants, Inc. announced the availability of the TSC BASIC for the 6800. The program resides in 9.5K of memory and is currently the fastest floating point BASIC interpreter available for any 8 bit micro. All of the standard BASIC statements and functions are supported as well as many extended capabilities. Both floating point and string variables are provided with strings being fully dynamic or unrestricted in size. Variable names may be either the standard types or double letter combinations allowing limited variable name mnemonics.

Other features include single and double dimensioned arrays. Array references support subscripts of 0 unlike several other 6800 BASIC's available. Array size, loop nesting, subroutine nesting and string length are only limited by the amount of user memory available in the machine. A tremendous enhancement is provided by the 'IF.. THEN..ELSE' construct. The 'ELSE' clause promotes a more structured type programming style, thus improving readability and conciseness of the program. The input buffer allows lines as long as 127 characters to be entered to take advantage of the complex statement structures permitted with this statement. Other features include the HEX function which allows hexadecimal number representation while PI provides an easy reference to this often used constant. The floating point arithmetic done by BASIC is performed to seven digits accuracy internally, with all answers printed to six. The dynamic range of the numbers is in the range of 10 raised to the plus or minus 37th power.

Overall, TSC BASIC is a fast and powerful BASIC. It is easily adapted to run in any 6800 system having at least 12K of user RAM available from location 0000. A system with 16K or more of memory is recommended for serious applications. The BASIC is available on Kansas City Standard cassette along with a complete user's manual for $39.95.

Technical Systems Consultants, Inc., Box 2574, West Lafayette, IN 47906, (317) 463-2502.

CIRCLE 263 ON READER SERVICE CARD

6800 DEVELOPMENT SYSTEM

WINTEK's new DEVELOPMENT PAC II includes 48K RAM and an EROM programmer for $1895. Also included is a RS-232 serial I/O port with switch selectable baud rates, 300 baud and 2400 baud cassette interfaces, an editor/assembler, and industrial BASIC, and all other items required to configure a 48K development system. WINTEK Corp., 902 N. 9th Street, Lafayette, IN 47904; (317) 742-6802.

Version of Pascal for the micro market

A DUTCH company is now selling a version of Pascal specifically aimed at the personal computer market. Systems such as the SWTP 680, with sufficient hardware to support the TSC Flex 1.0 operating system, are more than adequate to run programs written in the high-level language.

Available from Lucidata, which is based in Voorburg, Holland, the software can compile and run Pascal programs in a minimum of 12K plus 4K bytes of memory. The compiler itself is written in a subset of Pascal and incorporates such functions as Files, Procedures, Recursion, and Sets, as well as looping and branch constructs. It also incorporates some of the favourite micro functions such as Peek, Poke and User.

One of the more interesting aspects of the software is the inclusion of a page mode facility. This is brought in automatically if there is insufficient real memory for a large program.

The Pascal compiler and the P-code interpreter/run-time system is provided on a mini-floppy disc in Flex 1.0 format.

First samples

FIRST wafers of Motorola's new 16-bit microprocessor, the 68000, have passed through the production process at the company's plant in Austin, Texas, with a fair measure of success. A spokesman for the company said that only "one or two" problems needed to be sorted out in the second iteration of the design, and samples are now expected in August.

6800 PRODUCTS at A-VIDD

SOFTWARE DYNAMICS EDIT

This powerful 6800 editor is written in the TECO style. It will work on either Miniflex, Flex V1.0, or V2.0.

Notable features include:

—Two input files, reselectable from EDIT
—One output file, reselectable from EDIT
—Multiple commands allowed on command lines
—Commands for both line and character editing
—Value registers
—Commands strings can be taken from a file
—Iteration commands for building custom editing commands
—Arithmetic, logicals, and relationals (<,<=,<> etc.)
—Labels for edit programming

Example:

```
[[@>='0&@<='9^1M]
[@=32^1D]9VL]
```

This example would scan an entire file and place a tab character in front of every line after the line number if any, and deleting all spaces at the beginning of each line.

Price **$100.00**

TECO® is a registered trademark of DEC.

Software Dynamics Compiler Basic for Flex including the MAL assembler and Run time package: **$329.00**

Dealer inquiries invited

6800 FORTH

Word oriented interactive compiler interpreter. Totally structured language where you define your own words and syntax.
On KC tape **$34.95**

6800 Hardware:

Smoke Signal 16K static memory board uses low power chip's for reliable service.
New low price **$299.00**

Microtime 6800

Plugs into any I/O port of the SWTPC 6800. Supplies user with time in hours, minutes, seconds, AM or PM, month, date and year. (ie: 5/1/79 12:00:00 PM PST.) Provisions for optional battery or external A/C power supply. Supplied with software to set time, interrogate clock and print time.

Assembled & Tested	**$95.00**
Bare Board w/Documentation	**$35.00**

A-VIDD electronics co.

2210 Bellflower Boulevard
Long Beach, CA 90815
(213) 598-0444
(714) 821-0870

65Ø2 FORTH
68ØØ FORTH
Z-8Ø FORTH

FORTH is a unique threaded language that is ideally suited for systems and applications programming on a micro-processor system. The user may have the interactive FORTH Compiler/Interpreter system running stand-alone in 4K to 6K bytes of RAM. The system also offers a built-in incremental assembler and text editor. Since the FORTH language is vocabulary based, the user may tailor the system to resemble the needs and structure of any specific application. Programming in FORTH consists of defining new words, which draw upon the existing vocabulary, and which in turn may be used to define even more complex applications. Reverse Polish Notation and LIFO stacks are used in the FORTH system to process arithmetic expressions. Programs written in FORTH are compact and very fast.

APPLE II COMPUTER	$34.95
PET 2001 COMPUTER	$34.95
TRS-80 COMPUTER	$34.95
SWTP COMPUTER	$34.95

PROGRAMMA INTERNATIONAL, Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010
(213) 384-0579

SDOS—A 6800 DISK OPERATING SYSTEM

SDOS is a Disk Operating System for 6800 microprocessors with at least 32K of RAM and floppy (or other) disk drive. SDOS can support 64K RAM and any number or mixture of disk drives on-line.

SDOS provides a friendly environment for application software or development tasks. It supports both random (accessible to the byte) and sequential disk files, device independent I/O, a user customizable command interpreter, and easy tailoring of the disk drivers so that SDOS can be adapted to virtually any disk hardware. SDOS can handle disk devices/files with storage capacities of up to 2.14 billion bytes. New device drivers can be added, operating with or without interrupts.

New disk files are created automatically as needed. Furthermore, SDOS does all space management on a dynamic basis, so files can grow or shrink as desired.

Versions of SDOS are now available for Midwest Scientific Instruments (MSI), Cincinnati Milacron Model 20, Electronic Product Associates Micro-68, Conrac, and Wavemate 6800 systems. Several other 6800 computer systems should have versions of SDOS available soon.

Software Dynamics, 2111 W. Crescent, Suite G, Anaheim CA 92801.

6800 FLEX™/SWTPC Software

- **Renumbering System by Dave Degler**

  Renumber your BASIC programs with this new FLEX™ utility. You'll wish you had it if you paint yourself into programming "corners". Needs no extra RAM beyond the program being renumbered. With operation notes. Available on FLEX™ minifloppy disk or SWTPC KCS cassette.

- **Some Common Basic Programs by Lon Poole and Mary Borchers**

  Now adapted to FLEX™ and SWTPC 8K BASICS!
  67 key programs from the popular book, which is necessary as the manual. Conversion notes included.

  Disk 1: 37 programs on finance, investments, mortgage amortization, plotting, intergration, more.

  Disk 2: 30 programs on matrix arithmetic, statistics, calendar dates, metrics, more.

  Available on FLEX™ minifloppy disk or SWTPC 8K KCS cassettes.
  The book, Some Common Basic Programs – $ 8.50

- **Weekly Payroll / Income Expense Ledger / Club's Mailing List / Church Membership and Pledge Records by Roger L. Smith**

  These BASIC programs have had years of use and will be valuable additions to your SWTPC software library. Operation notes included. Cassette editions store data on data tapes. Each program is on one FLEX™ minifloppy disk or SWTPC 8K KCS cassette.

  Prices: FLEX™ minifloppy disk — **$ 16.95 each**
  Kansas City Standard SWTPC 8K BASICS Cassette — **$ 10.95 each**

All software shipped off the shelf. Please include check or money order. International: add $ 4.00 per item for air mail postage. U.S. First Class: add $ 2.00.

S S I Publications

4327 East Grove / Phoenix, Arizona 85040

* Distributed to dealers by MICROMEDIA Marketing, 800-423-4265
See us at the COMPUTER FAIRE, booth 423 in San Francisco, May 11-13

FLEX™ is a trademark of Technical Systems Consultants, Inc.

PROGRAM LIBRARY

from

CASCADE ENTERPRISES

ON CASSETTE TAPE

- **If you use Solos™ or have an 8080 Sol™ System, G2™ Extended Basic is faster and better than others requires (15.5K) $49.95**
- **For our 6800 users G2™ Standard Basic specifically for the S.W.T.P.C.™ 6800 requires (7K) $34.95**
  Both G2 basics are by Microsoft™
- **We have the following games and entertainment which will run in these machines:**

THE GAME TAPES ARE ONLY $14.95 each	SOL ™	SORCERER	TRS 80™	APPLE™	SWTPC ™
G2 BEAT THE HOUSE	•	•	•	•	•
G2 CLINIC	•	•	•	•	•
G2 OUTWIT	•	•	•	•	•
G2 THE MARKET		•	•	•	•
G2 ADVENTURE			•	•	•
G2 BLOCKADE			•	•	•
G2 OIL TYCOON			•	•	
G2 PILOT					

G2 Beat the House includes Blackjack, Craps, Roulette and Slot Machine. Craps is a very big program. Ours left 3K usable in Level II 16K TRS-80™. Allows a family to play as a group against the computer.

Call us anytime except Sunday (916) 926-5154 or write

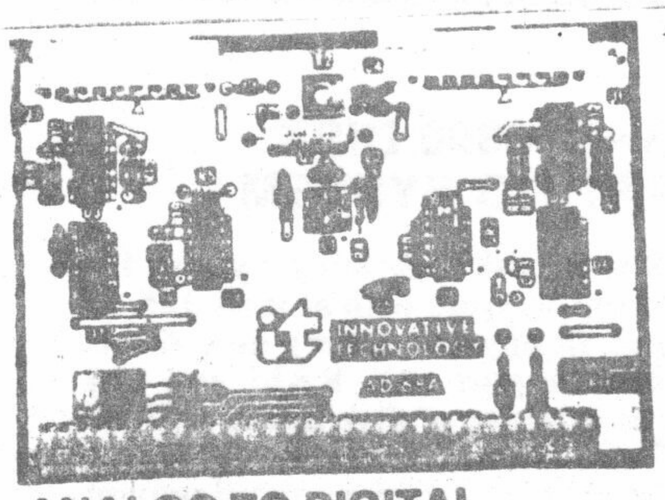
As an introductory offer—the first 500 orders get a free 10 minute blank cassette.

CASCADE ENTERPRISES
Box 213
Mount Shasta, CA 96067

Cal. Res. add 6% tax
Bank Cards add shipping & handling $2.00

☐ M/C ☐ B/A-Visa _____ Expires _____

Signature _____



ANALOG TO DIGITAL CONVERTER

The AD-68A is an eight channel analog to digital converter which occupies one I/O slot in a SWTPC 6800 type computer. It allows the computer to measure analog signals from 0 to 2.50 volts with a resolution of 10 millivolts. The AD-68A uses single ramp, elapsed time method of conversion with the computer providing control and counting functions through a 30 byte subroutine.

Unused analog inputs may be used to sense logic signals and switch closures. Regulated +5 volts and +/-12 volts are available at the input connectors to provide power for external circuits.

The AD-68A is currently being used in a variety of applications both in and outside of the home (especially in the university and industrial laboratory environment).

The AD-68A comes with complete documentation including the driver subroutine, BASIC program example, and schematic. $39.95 assembled and tested.

Innovative Technology, 510 Oxford Park, Garland TX 75043, 214-270-8393.

SOFTWARE CATALOG

A new catalog of microcomputer software products has been announced by TSC. Dozens of assembly language programs are described, mainly for the 6800 and 8080 microprocessor. The software includes editors, assemblers, text processors, math packages, games, utilities, and more. Most programs are sold with printed source listing and machine readable object code in the form of disk, cassette, or paper tape. Prices range from $3.75 to $75.00. Catalog: 25¢. Technical Systems Consultants, Inc., P.O. Box 2574, West Lafayette, IN 47906.

Nyhet!

Från MOTOROLA

MEX 6801 EVM
UTVECKLINGSMODUL

MEX 6801 EVM är ett datorkort uppbyggt kring MCU (Micro Computer Unit) 6801. Detta innehåller interface för terminal, 4K bytes statiskt RAM, socklar för I/O, PROM och timer samt wire wrap area för ytterligare expansion.

Debugprogrammet (LILBUG) innehåller 2K ROM med 32 instruktioner för bl. a.

- In- och utmatning till tape.
- Presentation och ändring i minne och register.
- Körning och stegning av program med kontrollpunkter.
- Avrop av Debug rutiner.

Ring och begär datablad

DISTRIBUTÖREN

08/49 25 05

INTERELKO AB
Sandsborgsvägen 55
122 33 Enskede
Kontorsväxel 08-49 26 60

SOMMAR-REALISATION

på delar till Ditt 6800-datorsystem....

Modifieringssats till Motorolas D2-kit	195,- (x)
Bus-IC (2x6880, 3x6887, 1x7430) för D2-kitet	68,- (x)
KCS-interface 6803 enligt RT 9, 1978	129,- (x)
Tangentbord Keytronic L 1648 med låda och bellkort	995,-
Videominne CÅ-6806 enligt RT 10, 1978	1100,-
Eprom-kort CÅ-6808 enligt RT 12, 1978 (för 18x2708)	499,- (x)
Bell-kort CÅ-6805 med högtalare	39,-
Eprom-programmerare CÅ-6807, för 2708, 2516, 2532	995,- (x)
EPROMMAR 2708 Fabr. Motorola, AMD, NS Pris Endast...	59,50/st(x)
Universalkort CÅ-6809 med klocka, timer, AD,DA,NCU.	1395,-
Floppykort CÅ-6817 för 1-4 enkelsidiga drivers	1250,-
Floppy Disk Drivers Shugart SA 400	2070,-

komponenter:

86-polig kontakt för 6800-bussen, lödstift för moderkort	29,- (x)
50-polig kontakt för D2-kitet (0,1 tums deln)	19,75 (x)
22-polig kontakt enkelrad, 0,156 tums deln. lödöron	5,25
2x18-pol.kontakt 0,156 tums deln. lödöron	5,25
samt ett flertal kontakter med varierande poltal och deln.	
kristaller 16,000 MHz	14,50
kristaller 1,8432 MHz	65,-

Begagnat och prototyper:

4k minneskort för 6800	500,-
Prototypen till universalkort 6809 (8 ing.AD-omv) fullt bestyckat med klocka, timer, NCU osv.	900,-
Prototypen till floppydisk-kontrollkortet 6817	900,-
Tangentbord G Risk 753 m låda och kabel m kontakter	495,-
Tangentbord med låda. 62 tangenter, ej avkodat	395,-
Videokort CÅ6806, demoex.	995,-
Nättrafo prim 220 V sek 6,8 volt 3VA	10,-
Nättrafo prim 220 V sek 5,5V, 2 x30 V, 110 V ca 50VA	20,-
Nättrafo prim 110/220 V, sek 15 V, 18 V ca 60 VA	22,-
Texas Silent 743 KSR med väska, manual och 5 rullar papper, V24-snitt, 110/300 baud, ÅÄÖ-tecken.	5000,-

OBS. Ovanstående specialpriser gäller t.o.m. 79-08-10 (x-markerade).
Vi har semesterstängt 9/7 - 5/8, och kan då endast i begränsad utsträckning expediera brevbeställningar. Moms ingår i alla priser.

TREVLIG SOMMAR ÖNSKAR VI PÅ CÅ-ELEKTRONIK !

INGENJÖRSFIRMA CÅ-ELEKTRONIK AB
Box 2010, 135 02 TYRESÖ
Ordertel. 08 - 742 34 01 må–to 12.30–16.30, fre 12.30–15.00
Postgiro 19 59 26-0 Bankgiro 490-8448

Alla priser inkl. 20,63% moms.
Frakt tillkommer.