CODEN: LUTFD2/(TFRT-5381)/1-126/(1988)

Filterhantering på externa massminnesmedier för litet processdatorsystem

Mikael Schmidt

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Juni 1988

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
June 1988

Document Number
CODEN: LUTFD2/(TFRT-5381)/1-126/(1988)

Author(s)
Mikael Schmidt

Supervisor
Per Hagander and Christer Åkerblom

Sponsoring organisation

Title and subtitle
Filhantering på externa massminnesmedier för litet processdatorsystem. (Filehandling for a small process control system.

Abstract

This report describes an implementation on how to handle files on different kinds of external memories related to a small process control system.

The communication with the external memory takes place over a serial channel and is independent on the type of memory. The system is written in FORTH. Two specific memory types are dealt with here, namely a PC and a backup unit (RAM-disc.)

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
126

Recipient's notes

Security classification

FILHANTERING PÅ EXTERNA MASSMINNESMEDIER

för litet processdatorsystem.

Ett Examensarbete av Mikael Schmidt.

Handledare : Per Hagander, Institutionen för Reglerteknik.
Christer Åkerblom, MICAB.

FÖRORD

Den rapport som följer behandlar mitt avslutande examensarbete på linjen för elektroteknik vid Lunds tekniska högskola. Detta arbete är utfört på MICAB, ett företag i forskarbyn Ideon i Lund. MICAB sysslar med utveckling av programmerbara styrsystem för industriellt bruk.

Jag vill här passa på att tacka Christer Åkerblom, som har varit min handledare på MICAB, och Per Hagander vid Institutionen för Reglerteknik på LTH som ställt upp som handledare för skolans räkning.

LUND 880615
Mikael Schmidt

INNEHÅLLSFÖRTECKNING :

1. Sammanfattning.

Examensarbetet är utfört vid MICAB, ett företag på Ideon i Lund. På MICAB har man utvecklat en processorbaserad styr- och regler- utrustning kallad Microflex10 (MF10, mer om den längre fram). Ex-jobbet har haft som mål att implementera kommandon i MF10 för hantering av filer på externa massminnesmedium samt att anpassa filhanteringsrutiner till dessa kommandon i två olika massminnestyper. Exempel på minnestyper är fristående floppydisk station, PC med skivminne, Winchesterminne, RAM-disk, Bubbelminne etc. De två massminnesmedium som är aktuella för den här implementeringen är dels en IBM-kompatibel PC dels en RAM-disk där den befintliga hårdvaran är framtagen på företaget.

Detta innebär en implementering i två steg :

I. Generella kommandon (FORTH-ord) i Microflex10 som skall vara oberoende av massminnesmedium.

II. En specifik del för varje medium där själva lagringen och hanteringen av filerna sker.

Syftet med att implementera filhantering på detta viset är att MF10 skall kunna operera direkt mot olika filer.

Ex. Man vill ha information om de olika processer som exekveras på MF10 och kan då, med vissa tidsintervall eller händelsestyrt, logga ut data på för ändamålet avsedda filer.

Eller kanske MF10 behöver exekvera en rutin som inte finns i dess bibliotek då skall MF10 själv kunna hämta ned den fil som innehåller den sökta koden och exekvera denna.

Det huvudsakliga arbetet har bestått i att implementera de specifika rutiner som varje medium kräver för att fungera mot MF10's kommandon.

Då en PC har använts som medium har rutinerna implementerats som Modula-2 program där det finns gränssnittsrutiner som kommunicerar direkt med operativsystemet (MS-DOS), vilket innebär att DOS's filhanteringssystem kan användas.

Då en RAM-disk används som minnesmedim måste både filstrukturen definieras och ett filhanteringssystem konstrueras. Då RAM-disken är baserad på samma processor (Rockwells R6511Q) som MF10, implementeras rutinerna i assemblyspråket för processorn.

2. Bakgrund.

Under arbetets gång har en del resurser tagits i anspråk och här följer en kort introduktion om den bakgrund och de förutsättningar och hjälpmedel som exjobbet har haft att utgå ifrån och att jobba med.

2.1 Microflex10 (MF10).

På MICAB har man utvecklat en processorbaserad styr- och regler- utrustning kallad Microflex10 (MF10) som alltså är den centrala enheten kring vilket allt rör sig, och från begränsningar hos MF10 har också problemställningen för detta examensarbete uppkommit. För närvarande är dock en ny modell, Microflex20, under utvecling men detta ex-jobb rör endast MF10. Systemet MF10 är uppbyggt kring Rockwells processor R6511Q. Själva regleringen och styrningen sker via ett antal analoga och digitala in/ut-gångar. Komunikationen med MF10 sker via två RS232-snitt, COM1 och CONSOLE, varav det ena (COM1) är försett med handskakning i hårdvaran, det andra saknar detsamma. Till snitten kan kopplas en överordnad PC eller annan kringutrustning som t.ex ett externminne.

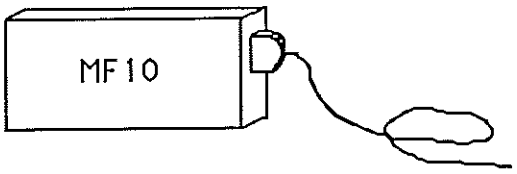


fig. 1. Microflex10

Programutvecklingen sker vanligtvis på en PC med hjälp av en vanlig texteditor, och när programmet är färdigt skickas källkoden ned till MF10, där den kompileras och läggs in i "ordlistan" över befintliga FORTH-ord. Program som är nedladdade på detta viset hamnar i MF10's statiska CMOS-RAM. Själva operativet är till största delen skrivet i FORTH, därför finns en motsvarande kompilator i PROM i MF10.

Då MF10 har tagit emot programvaran, t.ex. ett applikationsprogram, är den redo att exekvera detta. MF10 kan då vara installerad i en industriell tillämpning, ex. på ett fabriksgolv, och kan då arbeta på tre olika sätt :

1.) Som självständig enhet, d.v.s. ej kopplad till någon överordnad enhet.
2.) Ansluten till en PC som används för övervakning och via en operatör kan olika parametrar sätta och ändras som t.ex. börvärden.
3.) Flera MF10 kan kopplas samman via ett nätverk till en PC med liknande uppgifter som under punkt 2.

2.2 Microstore.

Microstore är en backupenhet som är utvecklad och framtagen på MICAB. Den är tänkt att användas som en portabel minnesenhet för att underlätta laddning av programvara till Microflexenheterna då någon eller flera av dem råkat ut för mjukvarufel vid t.ex. kraftiga störningar, kraftbortfall eller dyl.

Enheten är uppbygd kring samma processor som MF10, R6511Q (se nästa rubrik), och den anslutes till Microflex-enheten via RS232-snitt. På Microstore finns en s.k. nollkraftssockel i vilken en minneskapsel kan anslutas och lätt bytas ut, minnestypen

som används är E- eller E^2 -PROM, 32 kbyte. Den ursprungliga programvaran till backupenheten gör det möjligt att lagra programdata i form av objektkod i det EPROM som ansluts till enheten. Kommandona till Microstore ges via en knappsats bestående av tre tryckknappar, (L) Load, (D) Dump och (V) Verify. Vid lämplig knapptryckning kan backup-enheten få MF10 att sända iväg programdata, från i förväg definierade minnesareor, till Microstore-enheten där informationen lagras i minneskapseln ansluten till nollkraftssockeln.

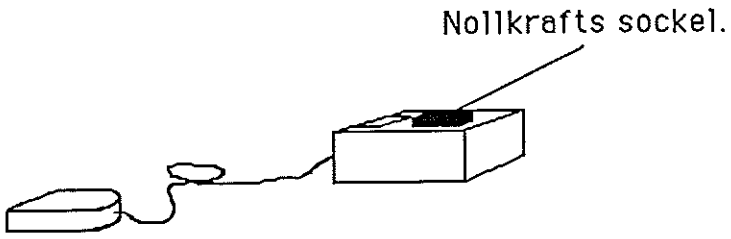


fig. 2. Backupenheten Microstore.

Vid motsatt förfarande, d.v.s. vid sändning av programdata från Microstore till MF10, hämtar backupenheten data från minneskretsen och sänder detta till värdenheten som placerar data på avsedd plats i minnet.

Ett exempel på användningen av Microstore med den ursprungliga programvaran :

Programutveckling sker på en PC med en ansluten MF10. Då programmet är klart laddas detta ned till MF10 där det kompileras och läggs i MF10's statiska RAM. Antag att det ute på ett fabriksgolv finns ett antal spridda Microflexenheter som alla behöver det applikationsprogram som just blivit klart på PC:n. Istället för att bära omkring på en tung PC laddas det kompilerade programmet från MF10 till Microstore som är en liten kompakt enhet, lätt att bära med sig. Backupenheten tas med ut på golvet och ansluts till en MF10 som laddas och kan sedan påbörja sin exekvering, backupenheten kopplas bort och kan anslutas till en ny MF10.

2.3 Processorn och assemblern.

Processorn är alltså en R6511Q (se app. C), en hårdvarumässigt utökad version av R6502. Den har 8-bitars datavägar och 16-bitars adressvägar och adresserna är disponerade enligt fig. 3.

Instruktionsuppsättningen är i princip samma för alla processorer i 6500-serien men hårdvarumässigt har R6511Q ett antal ingående komponenter som normalt måste realiseras med ett antal periferikretsar, så som :

* 192-byte statiskt RAM, så kallad "zero-page", som ligger på de hexadecimala adresserna 0040 - 00FF. Batteribackup ges via en "powerdown" pinne på kapseln. Minnet används normalt som stackutrymme vilket medför att stackpekaren endast blir på en byte. Även vid användning av pekare, d.v.s. vid indirekt adressering måste vägen tas via zero-page. Den indirekta adressen i programmet anges då som en byte, på den adressen i zero-page

finns den indirekta adressen som pekar ut var man hittar det sökta innehållet. Normalt räcker utrymmet till för detta men det kan i vissa tillämpningar vara en begränsning.

* Seriell in- och ut- matning.

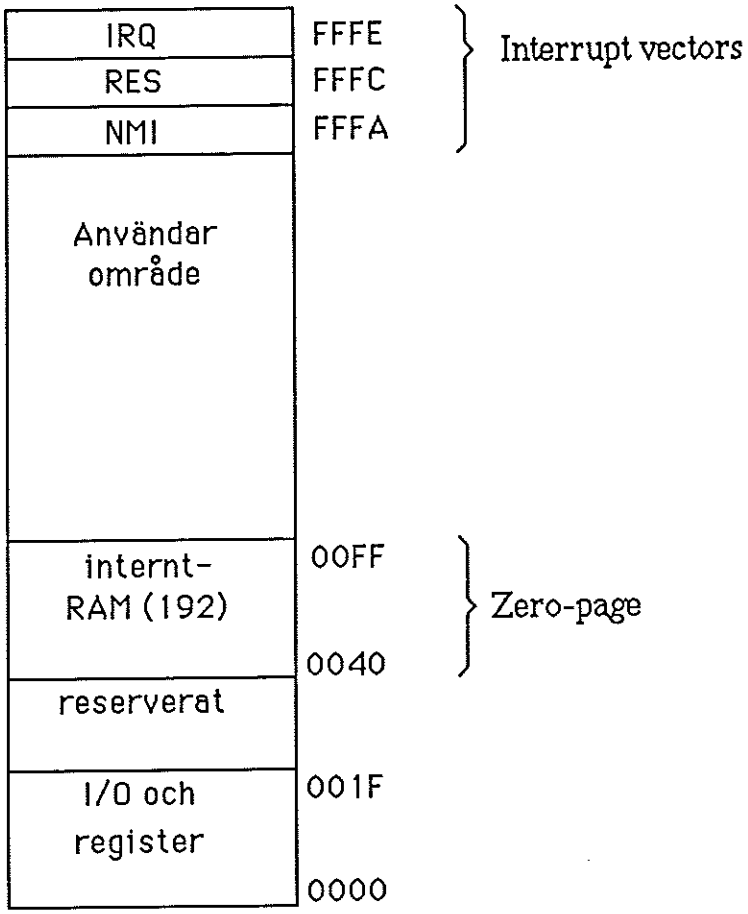
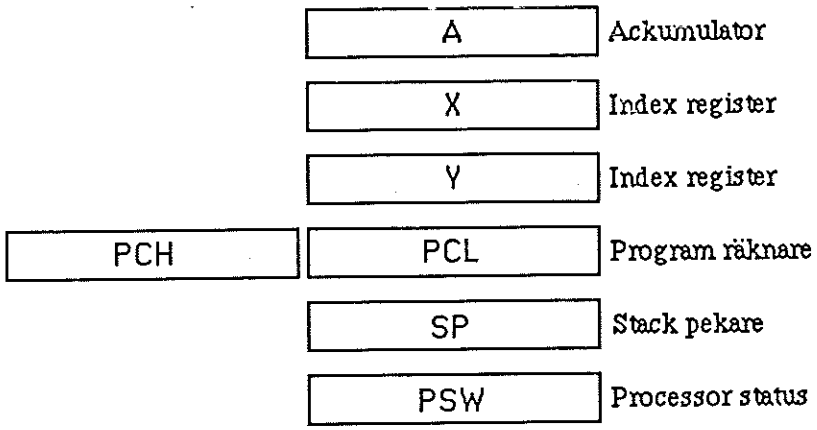


fig. 3. Minnes disposition för R6511Q.

Ytterligare skillnader finns men är av mindre intresse för denna implementering. Registerarkitekturen hos R6511Q är rätt sparsam (se fig. 4).

En nackdel med denna är att det förutom ackumulatorn endast finns två register, X och Y. Dessa bägge register kan inkrementeras, dekrementeras och användas vid indexerad minneshantering, men de kan inte användas vid aritmetiska operationer typ subtraheras från eller adderas till ackumulatorn.

Assemblern för R6511Q innehåller en del instruktioner som opererar direkt på minnesceller t.ex. INC M = inkrementering av minnescell och ROL M, LSR M = höger respektive vänsterskift av minnescell. Alla villkorliga hopp är relativa och om man vill göra ett hopp till en absolut adress finns instruktionen JMP (jump) som alltid är villkorslös.



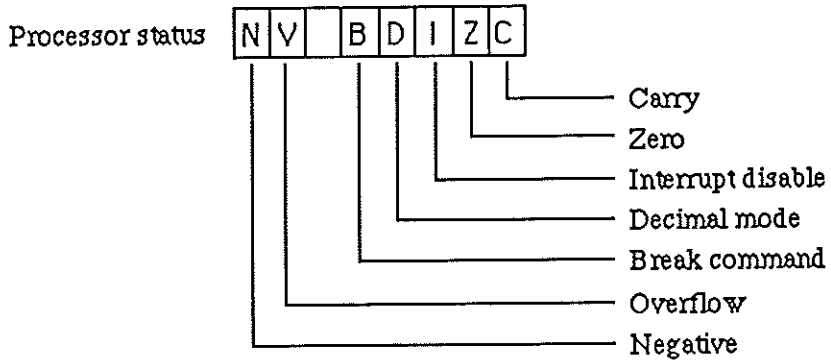


fig. 4. Register arkitekturen hos Rockwells R6511Q processor.

Det som annars är utmärkande för R6511Q är de många adresseringsmoderna som t.ex. för instruktionen LDA (Load Ackumulator) :

typ av adressering	instruktion
Immediate	LDA #operand
Zero page	LDA operand
Zero page, X	LDA operand, X
Absolute	LDA operand
Absolute, X	LDA operand, X
Absolute, Y	LDA operand, Y
(Indirect, X)	LDA (operand, X)
(Indirect), Y	LDA (operand), Y

2.4 FORTH.

FORTH är ett maskinnära, stackorienterat högnivåspråk, där man själv succesivt kan bygga sin programstruktur genom att definiera egna ord (ett FORTH-ord är det samma som ett programnamn eller ett kommando). Nya ord (kommandon) bygger på tidigare definierade ord som i sin tur bygger på tidigare ord etc. I botten finns en uppsättning basord (standardkommandon) typ aritmetiska ord, ex. "+", "-", "*", eller ord som opererar på stacken, ex. "SWAP", "DUP", "DROP" etc.

Detta sätt att definiera egna ord gör att för varje steg blir kommandona allt mer komplexa och det slutliga programmet som skall exekveras utgörs då bara av ett kommando/ord som anropas.

Exempel på ett FORTH-ord :

```
: TALSERIE  \ Skriver ut talen 0 - 100 på skärmen.
  100 0 DO I . LOOP ;
```

FORTH är ett kompakt språk som tack vare sin kombination av låg och högnivå också gör det snabbt att exekvera, väl lämpat för tidskritiska applikationer som styrning och reglering.

2.5 Debugger EBUG-800.

De nya rutiner som skrivits för Microstore är skrivna i assemblyspråket för R6511Q och för att möjliggöra programutveckling på rimlig tid måste programmet kunna avlusas. EBUG-800 är en universell debugger som kan anpassas till många olika processorer och emulerar det PROM som processorn normalt arbetar mot. Programmet, som är under utveckling, laddas ned i ett dynamiskt RAM som finns i debuggern och denna har då full insyn i programmet. Med hjälp av brytpunkter, visning av valda delar av minnesarean, möjlighet till ändringar i minnesceller och register etc. kan fel i programmet snabbt lokaliseras, justeras och på nytt provköras.

3. Problemställning.

Med hjälp av en editor av något slag kan man skapa, editera och hantera filer på PC-nivå. En fil, skapad på en PC, innehållande FORTH-ord kan sedan sändas ned till en Microflex-enhet där innehållet kompileras och läggs in i ordlistan. All kommandogivningen för nedsändandet av filen sker på PC-nivå. Så länge ett förutbestämt antal program skall exekveras, under förutsättning att dessa får plats i minnet, är denna metod fullt tillräcklig. Men antag att MF10 under exekveringens gång upptäcker att ytterligare rutiner behöves för att systemet skall uppföra sig som förväntat. Problemet är alltså att MF10 inte själv kan hämta ned dessa rutiner under exekveringens gång och målet är att implementera FORTH-ord i MF10 för hantering av filer på massminnesmedium. Det är alltså inte frågan om att i Microflexen implementera något slags filhanteringssystem utan all hantering sker på något externt medium.

Snittet mellan Microflex-enheten och det aktuella massminnesmediumet skall vara oberoende av minnestypen (fig. 5).

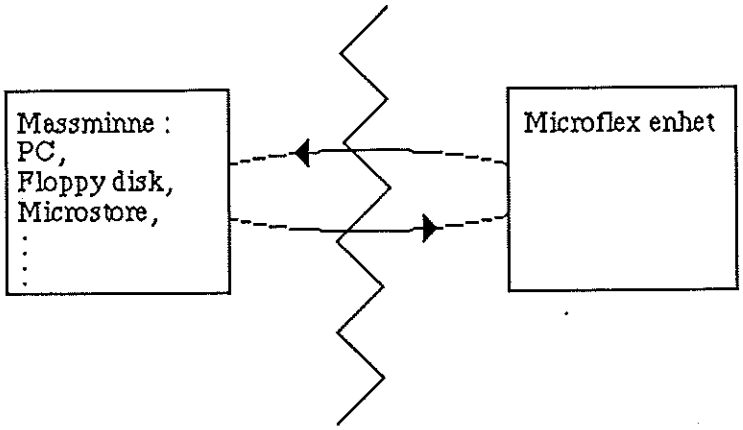


fig. 5. Snitt, oberoende av minnestyp.

Syftet med att kunna styra filhanteringen från MF10, ett exempel :

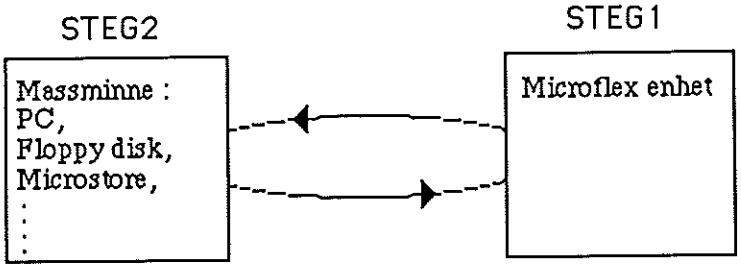
Antag att MF10 är installerad i sin industriella tillämpning, exekverande ett applikationsprogram för styrning av flera processer. Man önskar erhålla information om hur processerna beter sig och vill därför samla data om dem. Då ska MF10, händelse eller tidsstyrt, kunna öppna en godtycklig fil, logga ut data om processen, stänga filen och fortsätta exekveringen. På så sätt kan varje process få en egen fil att logga data på.

Ett annat exempel är att om MF10 exekverar ett program och en viss händelse indikerar att en rutin, som inte finns "inne", behöver exekveras skall MF10 kunna öppna den externa filen, hämta ned innhållet (FORTH- ord), kompilera och exekvera det.

4. Målformulering.

Målet med detta examensarbete är alltså att Microflex10 skall kunna "swappa" program under exekveringens gång, d.v.s. att kunna kasta ut ett program eller en del av ett program och på så sätt skapa utrymme för att kunna hämta ned ett nytt program för exekvering.

En metod att nå detta mål är att implementera en generell del, bestående av kommandon, i Microflex-enheten samt en specifik del för hantering av filer på varje massminnes- medium som är aktuellt att lagra filer på. Målet kan därför delas upp i delmål, STEG1 och STEG2.



Specifika rutiner för varje typ av massminnes-medium.

Generella kommandon i Microflex10 som inte är beroende av massminnes-mediet.

4.1 Delmål STEG1.

STEG1 består i att realisera den generella delen, d.v.s. att implementera den kod som MF10 behöver för att kunna styra filhanteringen på motsvarande medium.

Med generell menas här att en och samma implementering i MF10 skall kunna arbeta mot olika slags medium med samma resultat.

4.1.1 Kommandon.

MF10 skall kunna initiera en del grundläggande operationer på filer. Till varje operation är knutet ett kommando vilka exempelvis kan paras ihop enligt följande uppställning :

	Operation	Kommando
1.	Skapa en fil.	CREATEFILE
2.	Öppna en fil.	OPENFILE
3.	Stänga en fil.	CLOSEFILE
4.	Läsa en fil.	READFILE
5.	Skriva på en fil.	WRITEFILE
6.	Begära ned innehållet i en fil.	DOWNFETCH
7.	Begära ned innehållet i en fil.	SECUREFETCH
8.	Förstöra en fil.	DELFILE
9.	Visa innehållet i aktuellt bibliotek.	SHOWDIR

Kommandona implementeras som FORTH-ord och införlivas med biblioteksklistan (WORDS). Bakom varje kommando finns då kod som sänder iväg respektive kommando till aktuellt medium via ett RS232-snitt.

4.2 Delmål STEG2.

Det andra delmålet, STEG2, består i att implementera koden för den specifika delen. Implementeringen blir då knuten till den minnestyp man avser att använda. I detta fallet kommer kommer kod till två typer av minnesenheter att behöva konstrueras, nämligen till (1) en IBM-kompatibel PC med diskenheter och (2) en RAM-disk, Microstore, som är framtagen på MICAB. Detta innebär att delmål STEG2 i sin tur kan delas upp i två etappmål.

4.2.1 PC som specifikt massminnesmedium.

Microflex-enheten skall här jobba mot en IBM-kompatibel PC som redan har mycket av den specifika delen klar. Det finns ett operativsystem i MS-DOS, det finns redan väl definierade filer och kommandon som opererar på dessa och det finns uttag för floppy-disk skivor att lagra filer på. Vad som då behövs är rutiner som kan ta hand om tecken som kommer in på PC:ns seriekanal, RS232-snittet, dit MF10 är kopplad och som dessutom kan kommunicera med MS-DOS och dess filhanterings-kommandon.

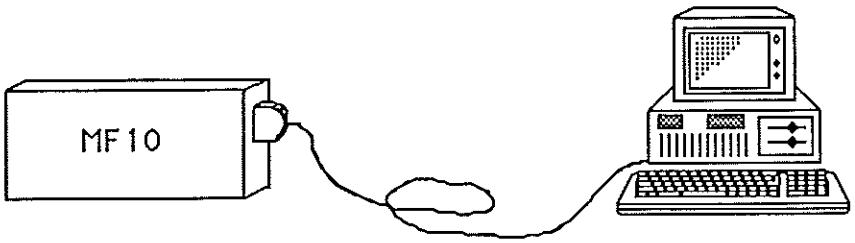


fig. 6. MF10 använder sig av en PC som massminnesmedium.

Modula-2 är ett av de programspråk som uppfyller dessa krav, där finns moduler innhållande procedurer som handhar RS232 kommunikationen, och det finns procedurer som utgör snitt mot MS-DOS (se fig. 7).

Det finns redan ett program, på PC nivå skrivet i Modula-2, som kommunicerar med MF10 via RS232-snittet. Kan programmet göras selektivt så att vissa tecken kan särbehandlas, kan filhanteringsrutiner motsvarande kommandona i MF10 initieras.

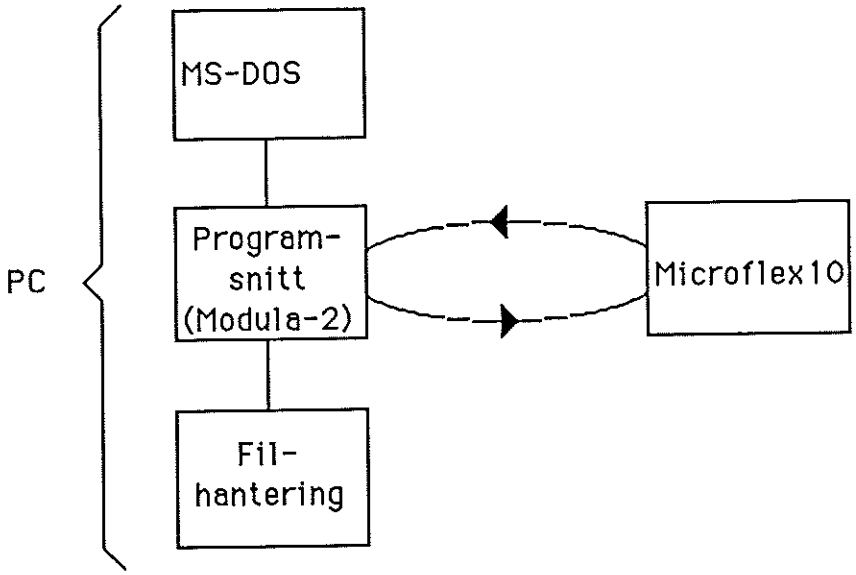


fig. 7. Ett Modula-2 program kan fås att utgöra en länk mellan Microflexen och MS-DOS.

4.2.2 Microstore som specifikt massminnesmedium.

Microstore (beskriven tidigare) är ytterligare en enhet som MF10 ska jobba mot men som inte har några filer definierade och därför inte heller några kommandon som opererar på filer. Rutinerna som från början finns implementerade i Microstore är avsedda för att kunna lagra programdata i E- eller E^2- PROM:et i form av icke relokerbar objektkod (= absolut kod). Då en laddning av en MF10 ska ske laddas *hela* innehållet i EPROM ned vilket medför både för- och nackdelar. En fördel är att lagringsformen för objektkod är mycket kompakt och laddningen går fort. Nackdelarna är att bara kompilerad programdata kan lagras och även lagring av små program kräver hela EPROM:et (32 kbyte) i anspråk då det saknas uppdelning i filer.

fig. 8. MF10 med backupenheten Microstore som massminne.

Ett önskemål är alltså att kunna lagra ASCII-filer (text-filer), t.ex. källkod eller data. För att den generella delen, som består av MF10:s filhanteringskommandon, skall fungera tillsammans med dessa önskemål krävs att en filstruktur definieras och ett filhanteringssystem konstrueras i Microstore.

5. Implementering.

Ingen hårdvara har behövt konstrueras utan all implementering är gjord i mjukvara. Programvaruutvecklingen till Microstore-enheten har skett på en Microflex-enhet där ROM:et har emulerats med hjälp av den tidigare beskrivna EBUG-800. Därför är uppläggningen av programmet bundet till MF10's minnesmappning (se fig. 9) i stället för till Microstore's.

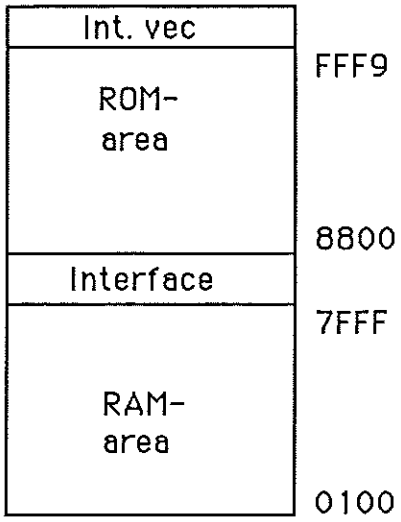


fig. 9. Minnesorganisation över användar området (se fig. 3) hos Microflex10.

Det E- eller E^2- PROM som är lagringsmediet på Microstore får ersättas av den statiska RAM-kapseln på utvecklings Microflexen. När programmet sedan överförs till Microstore mappas det om så det passar dess adressavkodning.

5.1 Generella kommandon i Microflex10.

Ett antal kommandoord måste implementeras i MF10 enligt 4.1.1. Kommandona implementeras som FORTH-ord som kompileras och införlivas i listan över exekverbara ord. Alla ord kan exekveras dels enskilt dels som delar i ett större program. Då ett kommando exekveras skickas först ett kontrolltecken (ctrl-D) iväg till externminnet via RS232-snittet, för att förbereda hanteringssystemet på att ett kommandotecken följer. Sedan skickas det hexadecimala värdet, enligt ASCII-tabellen, för det ordningsnummer som motsvarar kommandot. CREATEFILE skickar 31 = ASCII för "1", OPENFILE skickar 32 = ASCII "2" etc (se fig. 10).

Detta gör alla kommandon 1-9 och för alla utom två, nr. 6 DOWNFETCH och nr. 7 SECUREFETCH, är det också allt som görs. Förutom kontrolltecknet (ctrl-D) och kommandotecknet aktiverar ordet DOWNFETCH ett annat FORTH-ord i MF10, nämligen SCODE och ordet SECUREFETCH aktiverar FSCODE.

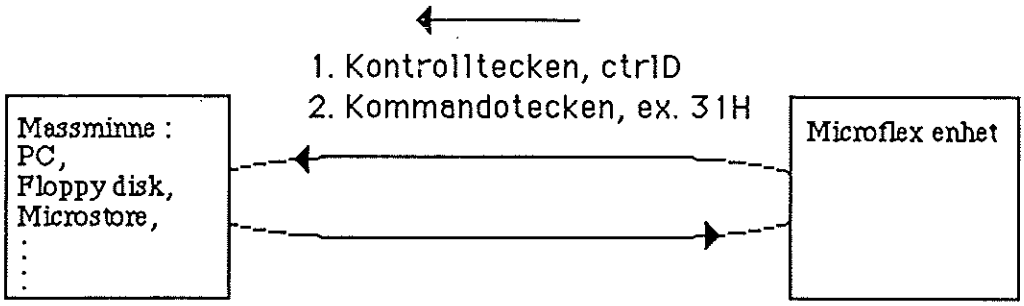


fig. 10. Teckenflöde vid kommandogivning från MF10.

DOWNFETCH beordrar den externa minnesenheten att skicka källkod till MF10. SCODE tar då hand om alla inkommande tecken och kompilerar dem radvis tills ett kontrolltecken från massminnet anländer som deaktiverar SCODE i MF10.

SECUREFETCH beordrar den externa minnesenheten att skicka ned källkod till MF10. Till skillnad från föregående kommando, som skickar tecken för tecken, packar SECUREFETCH in en rad i en ram som sedan skickas över (se 5.1.1). FSCODE tar hand om den inkommande ramen, avformatterar den och kompilerar innehållet (en rad).

Vid överföringen i bägge fallen sker handskakning mjukvarumässigt med hjälp av ett "XON-XOFF" protokoll som är ett sätt att synkronisera sändare och mottagare med varandra. Då SCODE i MF10 har tagit emot en teckenrad skickar den iväg ett XOFF-tecken (ett tecken i ASCII-tabellen med den hexadecimal representationen 13) till sändaren som då stänger av sändningen till MF10, detta för att SCODE i lugn och ro skall hinna kompilera den mottagna raden. När kompileringen är klar sänder SCODE iväg ett XON-tecken (hex repr. 11) för att sätta igång sändningen av en ny rad.

Namnen på kommandona är till stor del självinstruerande, här följer dock en kort redogörelse för innebörden.

1. CREATEFILE skapar en fil på motsvarande medium. Om filen redan existerar lämnas den öppen.
2. OPENFILE öppnar en existerande fil på motsvarande medium. Om filen inte existerar kan reaktionen bli olika beroende på medium.
3. CLOSEFILE stänger en öppen fil.
4. READFILE läser innehållet i en öppen fil och skriver ut det på skärmen. Fungerar således endast mot medium som har tillgång till bildskärm (PC).
5. WRITEFILE skriver ut information från MF10 på en öppnad fil.
6. DOWNFETCH hämtar innehållet på en öppnad fil och skickar ned det till Microflexen.
7. SECUREFETCH arbetar som DOWNFETCH med den skillnaden att här sänds innehållet i filen över i ramar med checksumma för säkrare överföring (se 5.1.1).

7. DELFILE raderar en fil som skall vara stängd innan operationen.
8. SHOWDIR används, liksom READFILE, endast mot medium som har en bildskärm att tillgå och visar då innehållet i aktuellt bibliotek.

Två ord som inte tidigare presenterats är EXEC" och FORGW". Dessa ord är inte knutna till själva filhanteringen utan är till för att underlätta exekveringen. Då ett nytt FORTH-ord skrivs krävs det att alla de ord som bygger upp det nya ordet är definierade sen tidigare (se 2.4). Då ett ord skrivs som har till uppgift att hämta ned ett ord från något externt massminne blir det problem att exekvera det nedhämtade ordet eftersom det inte finns i ordlistan från början (se Användarhandledningen). Detta problem kommer man runt med dessa bägge kommando.

9. EXEC" möjliggör exekvering av ord som definieras (läggs in i ordlistan) senare.
10. FORGW" gör det möjligt att ta bort ett ord ur ordlistan (WORDS) som ännu ej är definierat där.

Ytterligare två ord har lagts in i ordlistan, STX och ETX (Start of TeXt, End of TeXt). Orden begränsar den textmassa man måste skicka med vissa kommandon som till exempel OPENFILE som måste ha ett filnamn knutet till sig (se nedanstående exempel). Alla dessa ord införlivas i ordlistan över exekverbara kommandon.

Exempel :

När, vart och vilka tecken skickas egentligen (och varför) ? Kommandot DOWNFETCH får stå som exempel. Men för att exekveringen av DOWNFETCH skall vara meningsfull måste den filen som kommandot avser att hämta ned till MF10 vara en öppen fil. Därför får också OPENFILE stå som exempel (se fig. 11).
OPENFILE exekveras naturligtvis först och skickar iväg kontrolltecknet (ctrlD) för att förbereda mottagaren (massminnesenheten) att ett kommandotecken följer. Kommandotecknet för OPENFILE har det hexadecimala värdet 32H och då mottagaren identifierat tecknet förväntar den sig ett filnamn för att kunna hitta filen. Därför skickas filnamnet över från MF10 till minnesenheten men det är inte OPENFILE i sig självt som skickar det utan det bifogas till kommandot.
Nu är exekveringen av OPENFILE klar och filen med filnamnet FILENAME.EXT är öppen. Nu exekveras DOWNFETCH som börjar skicka kontrolltecknet (ctrlD) följt av kommandotecknet som för DOWNFETCH har den hexadecimala representationen 36H. När det tecknet just skickats iväg aktiverar DOWNFETCH ett nytt kommando i MF10, nämligen SCODE, som skall ta emot och kompilera innehållet i den fil som skall sändas ned från minnesenheten. Då mottagaren identifierat kommandotecknet börjar den skicka ned innehållet i filen rad för rad. Då SCODE i MF10 har tagit emot en rad skickar den ett XOFF-tecken som stänger av sändningen från minnesenheten så att SCODE får tid på sig att kompilera den nya raden. När raden är färdigkompilerad skickar SCODE ett XON-tecken som sätter igång sändningen av en ny rad. När filen är slut skickar minnesenheten ett kontrolltecken till MF10 som deaktiverar

SCODE och DOWNFETCH är exekverad. Den här programsnutten ser då ut så här :

```
OPENFILE STX ." FILENAME.EXT" ETX
DOWNFETCH
```

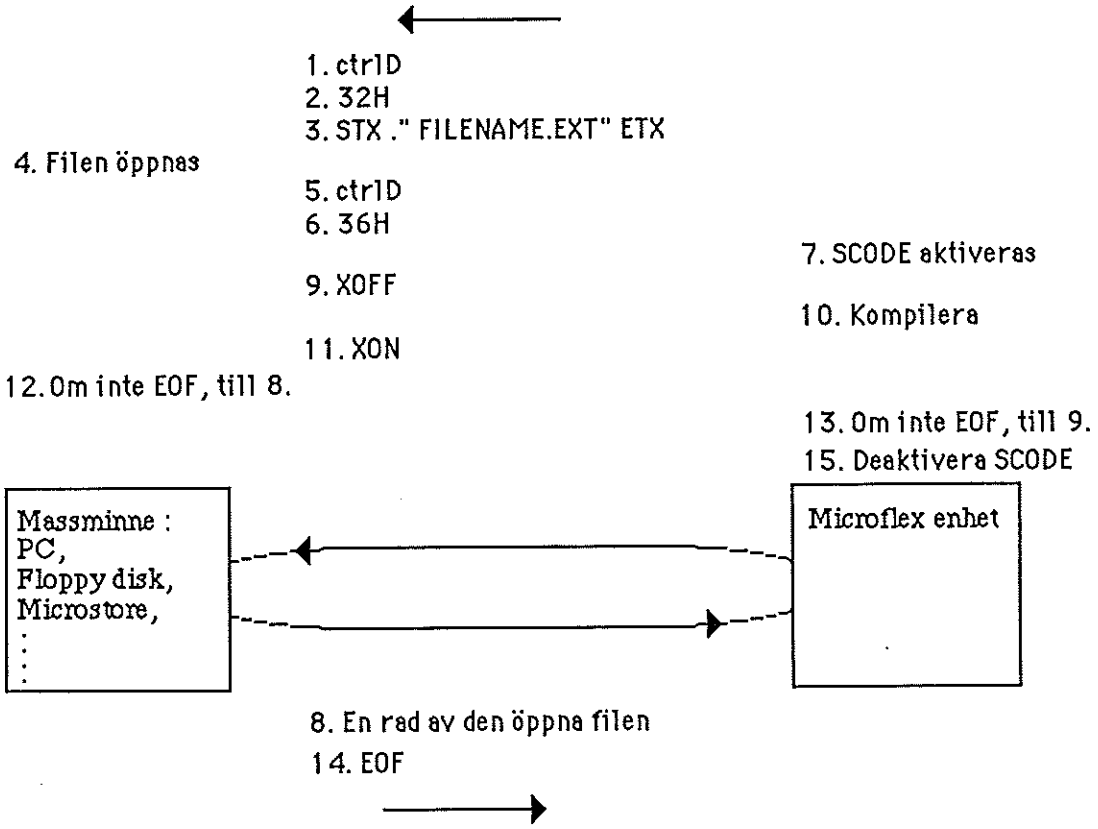


fig. 11. Flödet av tecken vid exekvering av kommandona OPENFILE och DOWNFETCH.

5.1.1 Överföring i ramar med checksumma.

Ovan nämns att SECUREFETCH använder sig av ramar med checksumma vilket kräver en närmare presentation.

För att öka detekterbarheten av fel vid den seriella kommunikationen mellan ett massminnesmedium och Miocroflex10 överförs data i s.k. ramar. En ram består av ett starttecken, information om hur många bytes data ramen överför, själva data och sist en checksumma som är två-komplementsumman av info-byten om ramens längd och datadelen. En ram kan ha följande utseende (se även fig 12) :

DATA RAM

Byte nr :

1	SOH, Start Of Header. Hexadecimal representation = 01H.
2..3	Antalet binära byte i datadelen i denna ram, två ASCII-tecken.
4..x	Data bytes, två ASCII-tecken per byte.
x+1..x+2	Checksumma, två ASCII-tecken.

SLUT RAM
Byte nr :

1	SOH, Start Of Header. Hexadecimal representation = 01H.
2..3	Antalet binära byte i denna ram, skall var "00".
4..5	Checksumma, skall vara "00".

Då en rad av filen skall skickas packas tecknen in i ramen ett i taget samtidigt som checksumman räknas ut. Vid själva överförandet av raden skickas hela ramen över på en gång, mottagaren avformatterar den mottagna ramen och kontrollerar att ramen är korrekt överförd. Om så är fallet skickar mottagaren ett positivt meddelande (ACK=Acknowledgement) till sändaren som då kan förbereda sändningen av nästa ram (och rad). Om däremot checksumman inte skulle överensstämma med summan av det överförda innehållet skickar mottagaren ett negativt meddelande (NAK=Negative Acknowledgement) till sändaren som då gör en omsändning av samma ram. Detta kan i princip upprepas ett oändligt antal gånger men bör avbrytas efter ett förutbestämt antal gånger då förbindelsen anses vara bruten.

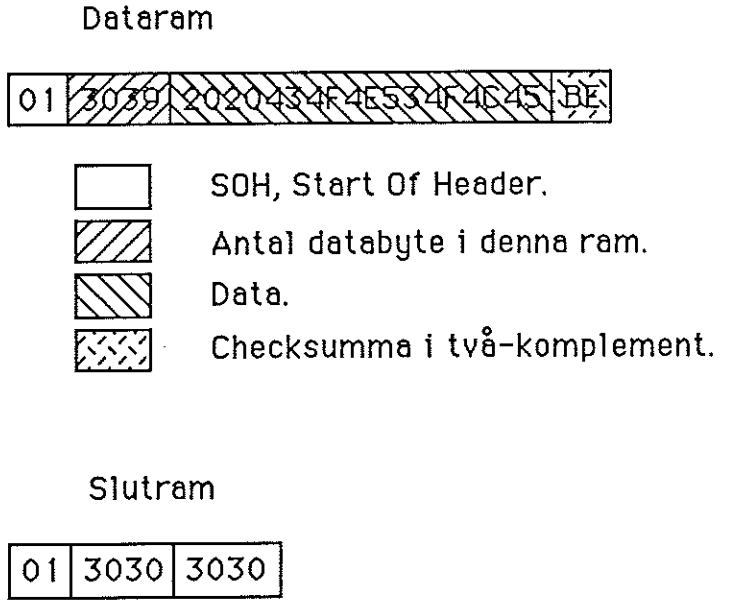


fig. 12. Uppbyggnaden av en dataram och slutram. Observera att varje byte skickas som två ASCII-tecken.

Om nu kommandot SECUREFETCH är ett säkrare sätt att överföra data på varför då DOWNFETCH ? I en störig miljö kanske det är ett måste att använda säker kommunikation, speciellt vid stora filöverföringar, men det tar ca 50% längre tid. Vid störningsfri omgivning är alltså DOWNFETCH att föredra.

5.2 Specifika rutiner med PC som massminne.

COMIC heter det program på PC:n som ursprungligen utgjorde och fortfarande, med en del modifieringar, utgör snittet mellan MF10 och en överordnad PC. COMIC tar emot alla tecken som kommer från MF10, skriver ut dem på skärmen, och sänder ned alla från tangentbordet kommande tecken till MF10. Modifieringen består i att låta COMIC testa varje inkommande tecken och känna av om det inkomna

tecknet är ett vanligt tecken som skall skrivas ut på skärmen eller om det är ett filhanterings-kontrolltecken (ctrl-D).

Då kontrolltecknet mottagits vet programmet att det kommer ett kommandotecken och då detta mottagits är det dags att exekvera motsvarande rutin. Alla kommandon i MF10 har en motsvarande filhanteringsrutin på PC:n. Dessa rutiner är, liksom COMIC, skrivna i program språket MODULA-2 och har lagts i en egen modul "FileHandler" (se appendix) som har kontakt med COMIC via exporterade variabler och procedurer.

De flesta kommandoorden i MF10 kan exekveras enskilt från tangentbordet eller ingå i ett program t. ex ett applikationsprogram. Detta kräver att filhanterings-paketet kan skilja dessa bägge fall åt. Skillnaden består i att då ett kommando exekveras från tangent bordet skrivs också filnamnet in via tangentbordet i motsats till om kommandot exekveras från MF10 då filnamnet omges av "STX" och "ETX", två detekterbara tecken.

Alla i MF10 implementerade kommandon kan exekveras enskilt men en del är inte lämpade att exekveras inne i en programkörning som t.ex. READFILE och SHOWDIR som genererar utskrifter på skärmen. SHOWDIR liksom EXEC" och FORGW", är för övrigt inte möjliga att exekvera på detta sätt.

5.2.1 Kort om procedurerna i FileHandler.

Procedurerna i modulen "FileHandler" är skrivna i programspråket Modula-2 och det finns en procedur för varje filhanteringskommando i Microflexen. "FileHandler" är alltså namnet på en undermodul till programmet COMIC på PC:n. COMIC anropar alltså någon procedur i "FileHandler" (se fig. 13).

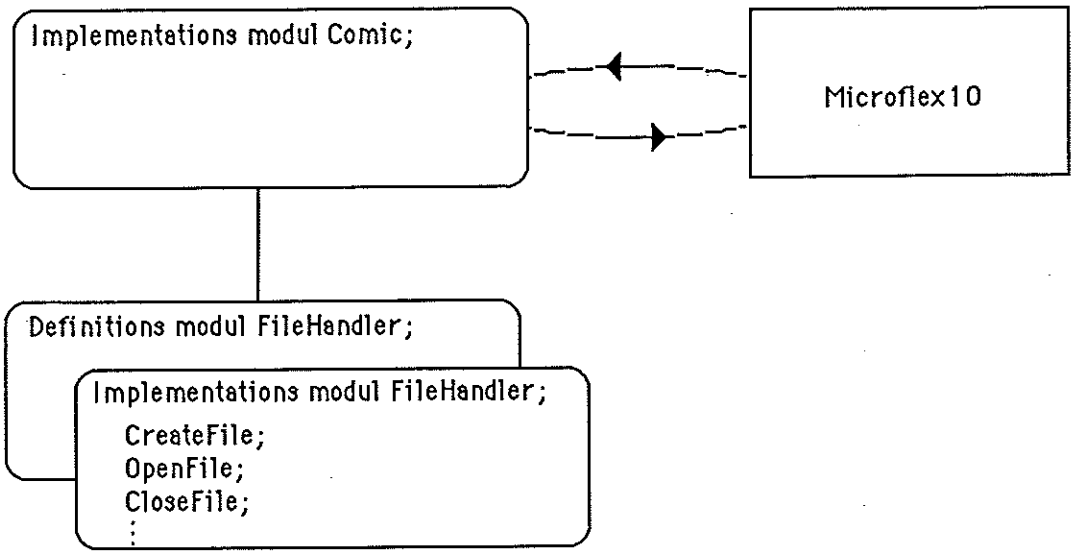


fig. 13. Samband mellan Comic, FileHandler och Microflex10.

1. CreateFile; (FILENAME.EXT)
   Söker först efter filen med filnamnet FILENAME.EXT och om filen redan existerar lämnas den öppen. Om filen inte existerar skapas den och lämnas öppen.

2. OpenFile; (FILENAME.EXT)
   Söker efter den givna filen och om den finns öppnas den. Om den inte existerar gives ett erbjudande om att skapa den (Y/N). Besvaras detta jakande skapas en fil med det givna filnamnet som sedan lämnas öppen.

3. CloseFile;
   Endast en fil tillåts vara öppen åt gången och CloseFile stänger denna.

4. ReadFile;
   Läser innehållet i den öppnade filen och skriver ut det på skärmen. Används endast vid exekvering från tangentbord.

5. WriteFile;
   Läser tecken som kommer in på RS232-snittet, anslutet till MF10, och skriver dem på den öppna filen.

6. DownFetch;
   Sänder ned innehållet i en öppen fil till Microflexen där det tas om hand och kompileras radvis med hjälp av SCODE. Arbetar med XON-XOFF protokoll. Då hela filen är överförd stängs den.

7. SecureFetch;
   Sänder ned innehållet i en öppen fil radvis, där vaje rad skickas i en ram (se 5.1.1). Ramen tas emot och avformatteras av FSCODE varpå den antingen kompileras eller begärs omsänd. Arbetar med XON-XOFF protokoll. Filen stängs efter överföringen.

8. DelFile; (FILENAME.EXT)
   Tar bort en stängd fil specifierad av FILENAME.EXT.

9. ShowDir; (Options)
   Visar innehållet i aktuellt bibliotek på skärmen. Options är vanliga MS-DOS options typ /W, /P, wildcharacters etc.

Förutom dessa 9 hanteringsrutiner finns fem underprogram som kompletterar programmet.

FileError;
Innehåller diverse felutskrifter och anropas då en operation mot en fil har misslyckats.

WriteFileName;
Skriver ut filnamnet i strängen "FileNameString" men skippar resterande blanktecken.

ReadFileName;
Läser in filnamnet på den fil som avses att operera på. Kommer filnamnet från ett exekverande program på MF10 så inleds det med "STX" i motsats till om det kommer från tangentbordet vid enskild exekvering.

Ececute;
Mottager en textsträng. Skickar tillbaka en mottagen sträng ut på seriekanalen följt av ett CR.

ForgetWord;
Mottager en textsträng. Skickar ut ordet "FORGET" följt av den mottagna strängen och ett CR.

5.3 Specifika rutiner med Microstore som massminne.

Microstore, beskriven tidigare (se 2.3), är det andra massminnesmediet som är aktuellt i denna implementering. I motsats till PC:n har Microstore inget operativsystem, inga definierade filer och därför inte heller något filhanteringssystem.

Eftersom snittet mellan MF10 och massminnesmediet skall vara oberoende av minnestypen måste Microstore fås att reagera på samma sätt som PC:n (se fig. 14).

För att detta skall vara möjligt måste Microstore kunna lagra information i form av filer vilket i sin tur kräver att en filstruktur definieras.

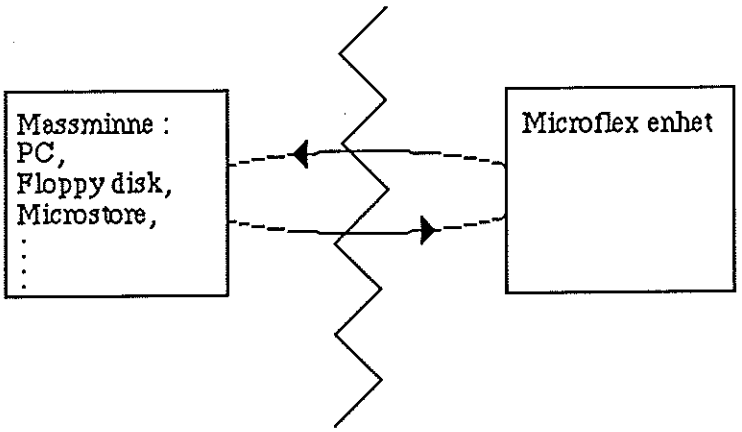


fig. 14. Kommandona i MF10 skall vara oberoende av massminnestyp.

5.3.1 Lagringskretsar.

Tidigare har nämts att de kretsar som Microstore lagrar data i är av typen 32 kbyte E- eller E^2PROM, men om ett filhanteringssystem skall vara meningsfullt att konstruera kan inte EPROM användas eftersom ett sådant bränns i sin helhet vid ett och samma tillfälle och kan endast raderas i uv-ljus. E^2PROM (Electrical Erasable Progamable Read Only Memory) är betydligt smidigare att arbeta med och är, som namnet antyder, elektriskt raderbart. Då skrivning av data till en enskild minnescell görs raderas denna automatiskt innan skrivningen utförs. Möjligheten att kunna skriva i enskilda minnesceller vid olika tillfällen är en förutsättning för ett filhanteringssystem, E^2PROM är alltså enda möjligheten.

Ett alternativ finns dock, statiskt RAM med inbyggd batteribackup (se avsnitt 6).

5.3.2 Läsrutin.

Ett program som exekverar i en Microflex, skrivet för att arbeta mot en PC, innehåller utskrifter som kräver en bildskärm. Om backupenheten skall kunna arbeta mot en godtycklig Microflex måste den kunna reagera på vissa tecken, så som

filkommandon, och ignorera andra tecken som ingår i utskrifter e.dyl.
Om en läsrutin implementeras som hela tiden läser vad som kommer på seriekanalen kan man få den att bli selektiv, d.v.s. känna igen de tecken som har någon innebörd för den och kasta resterande tecken.

5.3.3 Fildefinition.

En fildefinition på ett medium med sådana begränsnigar (litet minnesutrymme) kan med fördel vara enkel för att inte data *om* filerna skall ta upp för mycket plats av det värdefulla minnesutrymmet. En fil består lämpligen av en kropp och ett huvud. Kroppen innehåller själva informationen som man vill spara medan huvudet innehåller information om filen till exempel hur stor den är.
Ett minimum av vad ett filhuvud bör innehålla är :

1. Namn på filen.
2. Filens storlek.

Det kan också vara lämpligt att ha

3. Indexblock.

- Filkroppen är den del där själva lagringen av data sker och för att enklare kunna administrera minnes arean delas denna upp i block (segment) av storleken 512 byte/block. En filkropp byggs då upp av ett antal block som tillsammans motsvarar den ungefärliga storleken hos innehållet man vill spara. Detta innebär att det sista blocket i filkroppen inte blir fullt utan kommer att innehålla en outnyttjad del som i genomsnitt är av storleken 512/2 = 256 byte. Ju mindre blockstorlek desto högre blir utnyttjandegraden av filkroppen men det innebär också mer administrativt arbete. För att markera att ett block är upptaget utgör den första byten i varje block en flagga (occupied) som är satt om blocket är ockuperat.

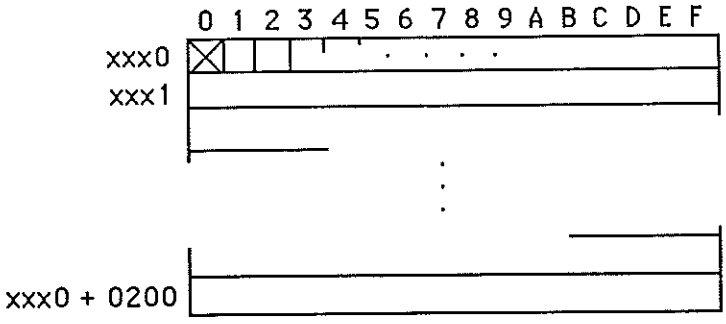


fig. 15. Varje minnesblock är 512 byte stort (= 0200 hexadecimalt) och första byten (markerad med kryss) i varje block utgörs av en flagga som talar om ifall blocket är ockuperat eller ej.

- Filhuvudet:
- Filnamnet är filens identitet och läggs in i en sträng bestående av N stycken tecken där N= 20.
- Filens storlek upptar 2 byte och anges som antalet av filen i anspråktagna block. Sist i filen, d.v.s. i det sista logiska blocket, ligger ett EOF-tecken för att markera slutet på filen (End-Of-File).

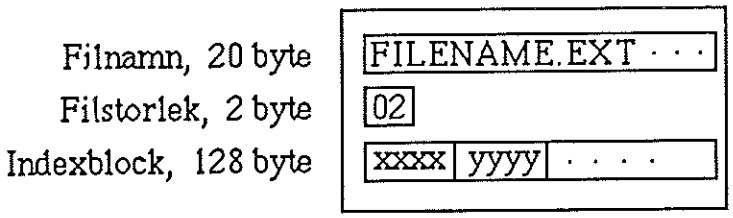


fig. 16. Utseendet hos ett filhuvud som har reserverat två minnesblock med start adresserna "xxxx" resp. "yyyy".

Ett filhuvud upptar ett utrymme på 150 byte

- Indexblock upptar 128 byte av ett filhuvud och är en vektor av typen "array [0..127] of integer;". Om indexblocket delas upp i enheter om två bytes var kan varje enhet rymma en 16-bitars adress, d.v.s. en pekare till ett block i minnet. Minnescell nr. 0 och minnescell nr.1 kommer att innehålla en pekare till block nr. 0 (se fig. 17).

Fördelen med indexblock är att logiskt konsekutiva block inte behöver vara fysiskt konsekutiva. Hade en logiskt sammanhängande fil tvunget behövt vara fysiskt sammanhängande skulle minnet efter ett tag vara fragmenterat, d.v.s. fullt av luckor som inte hade kunnat utnyttjas.

Med ovan nämda numeriska värden kan varje fil hålla 128 / 2 · 512 = 32 kbyte. Varje filhuvud kommer att få ett utseende enligt figur 16.

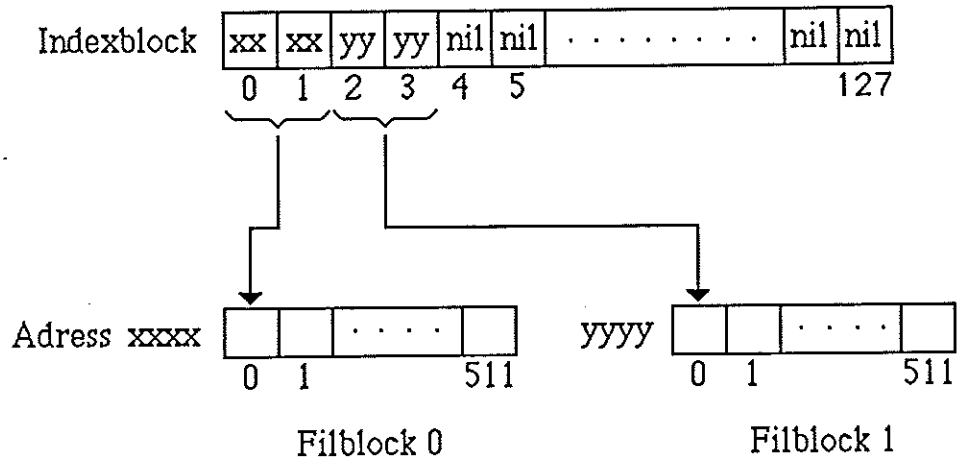


fig. 17. Uppläggningen av indexblock.

5.3.4 Uppläggning av filer.

Vid arbete med filer är det naturligast att kunna reservara plats för ett obegränsat antal filer men med tanke på att Microstore har en sådan begränsad minnesstorlek

(32 kbyte) är det rimligt att begränsa maximala antalet filer som varje minneskapsel kan hålla. Det blir då betydligt enklare att administrera eftersom man redan från början kan reservera utrymme för max antal filhuvuden. I denna implementering är maximalt antal filer satt till 8 stycken och den reserverade minnesarean för filhuvudena blir då 8·150=1200 byte.

Ytterligare minnnesutrymme av PROM:et måste tas i anspråk för att kunna administrera och informera om t.ex. hur många filer PROM:et för närvarande håller och vilka utav blocken som är upptagna etc. Denna information är generell och är överordnad filhuvudena och består av flaggor, konstanter, variabler och pekare och är alltså knuten till just den kapsel där filerna finns lagrade. Dessa parametrar upptar ett utrymme på 19 byte och totalt måste då reserveras utrymme, för filhuvuden plus filparametrar, på 1219 byte.

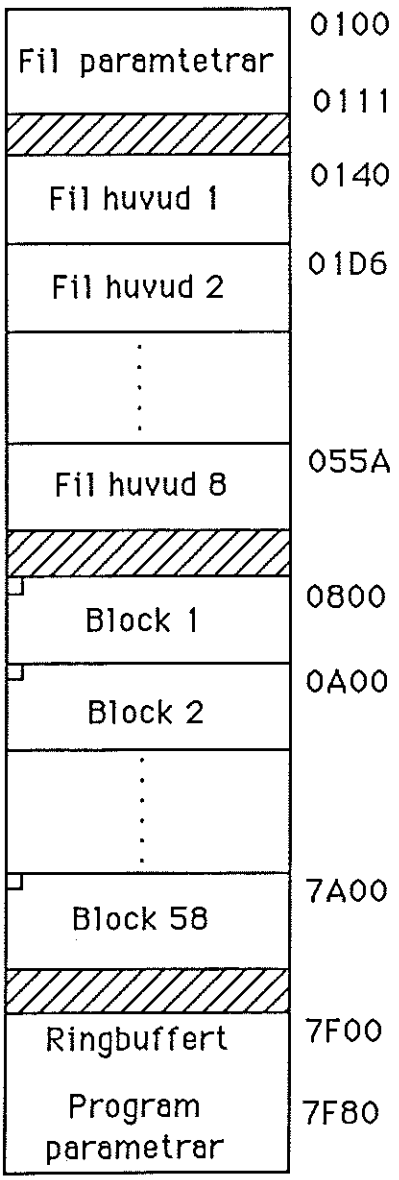


fig. 18. Minnes organisation över användarområdet (d.v.s. RAM:et) hos den emulerade modellen.

Sedan måste utrymme för programparametrar reserveras. Exempel på sådana är Booleska variabler, ringbuffert, konstanter etc. Dessa parametrar är inte knutna till några speciella filer utan är generella och behövs endast så länge programmet exekveras. Därför läggs dessa i systemets RAM-kapsel (på den emulerade modellen har ju RAM-kapseln ersatt EPROM:et hos Microstore så därför måste program-parametrarna samsas om utrymmet med filer och filparametrar).

Dispositionen av minnesarean får ett utseende enligt figur 18.

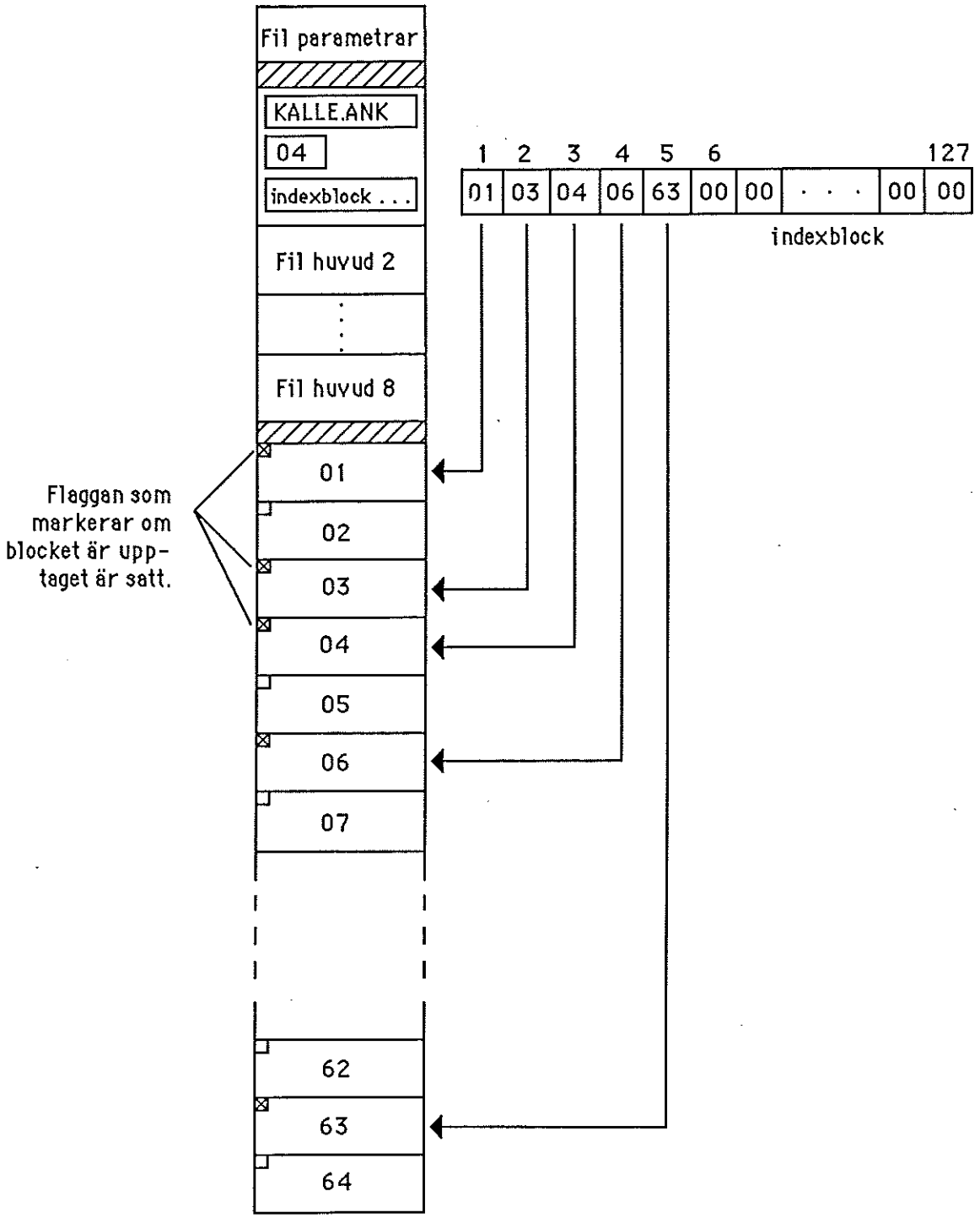


fig. 19. Exempel på filutseende. Minnesblocken refereras här endast till med hjälp av ordningsnummer, ej att förväxla med adresser.

Ett exempel över hur en fil kan vara uppbyggd.

En bild får illustrera hur filen är uppbyggd (se fig.19). Vi kan kalla filen "KALLE.ANK" och den upptar fem minnesblock. På grund av olika orsaker ligger inte blocken fysiskt konsekutivt i minnet men indexblocket håller ordning på den logiska ordningen mellan blocken. Den första byten i varje minnesblock utgörs av en flagga (occupied-flag) som talar om ifall blocket ifråga är ockuperat eller ej och när filen tas bort nollställs endast flaggorna i de block som var knutna till filen.
Om filen skulle behöva förlängas med ytterligare data söker systemet upp det första lediga blocket,lägger adressen till blocket i Indexblocket, ett-ställer flaggan(occupied-flag) samt räknar upp filstorleksvariabeln med ett. Celler i Indexblocket som inte pekar på något block är nollställda.

5.3.5 Mjukvaran till Microstore.

Den mesta tiden av exjobbet har gått åt till att konstruera och koda programmet till Microstore. Programutvecklingen har alltså skett på en PC med hjälp av en vanlig texteditor och kodats i R6511Q-assembler (se 2.3). Efter assembleringen av programmet skickas det ned till EBUG:en som emulerar ROM:et varpå testning kan påbörjas.

De rutiner i backupenheten som genererar tecken till mottagaren för utskrift på en bildskärm kan vid första anblicken vara meningslösa då backupenheten vanligtvis är kopplad till en Microflex som saknar bildskärm. På PC:n finns det interfaceprogram som kommunicerar direkt med dessa rutiner i Microstore. Detta innebär att man kan koppla ihop en PC med backupenheten via serie kanalen och operera direkt på filerna i mediet. Mer om interfaceprogrammen på PC:n under punkt 5.4.

Huvudprogrammet (se appendix) består av detekteringsrutiner för kontrolltecken och kommandotecken samt de rutiner som har sin direkta motsvarighet i något av de 8 kommandon som finns i MF10.

fig. 20. MF10 med Microstore som massminnesenhet.

WAITCOMMAND är den första detekteringsrutinen och den ligger och väntar på ett ctrl-D tecken, alla andra tecken som anländer via seriekanalen kastas.

READCOMMAND. Då ett ctrl-D mottagits av ovanstående rutin måste kommandot identifieras och READCOMMAND ligger och väntar på kommandotecken så att motsvarande hanteringsrutin kan sparkas igång.

CREATEFILE söker efter en fil med filnamnet "FILENAME.EXT", och om den inte finns och om det finns plats skapas ett filhuvud. Filnamnet läggs in på sin avsedda plats i filhuvudet , storleken på filen nollställs och pekarna i indexblocket

sätts till nil.

OPENFILE söker efter en fil specificerad av "FILENAME.EXT". Om den existerar öppnas den genom att en pekare, "OpenFileP" belägen i "zero-page", sätts att peka på det aktuella filhuvudet. Om filen inte existerar sätts "OpenFileP = nil".

CLOSEFILE sätter pekaren "OpenFileP" till nil som är ett "omöjligt värde", dvs ett värde som inte har någon motsvarighet i en möjlig adress (nil = 0000H).

READFILE. På PC:n finns ett par interfaceprogram med syfte att kunna kommunicera direkt med Microstore. Det interfaceprogram som kommunicerar med rutinen READFILE på backupenheten heter "ReadFile" och är skriven i Modula-2.
READFILE skickar upp tecken till PC:n som skriver ut dessa på skärmen och på så sätt kan man lista en fil lagrad på Microstore. För att synkronisera sändningen mellan backupenheten och PC:n används "XON-XOFF" protokoll.

WRITEFILE läser inkommande tecken på seriekanalen och placerar in dem i den öppna filen .
Om Microstore är ansluten till MF10 kan WRITEFILE anropas då man önskar att logga ut data på en fil eller skriva ut någon annan information.
Om backupenheten istället kopplas direkt till en PC som innehåller Modula-2 programmet "DumpFile", som anropar WRITEFILE, kan man ladda ned en fil direkt från PC:n till backupenheten.
WRITEFILE uppfyller också valmöjligheten att kunna , då en redan existerande fil på nytt har öppnats, skriva över det redan befintliga innehållet i filen eller att fortsätta skriva i filen där man sist slutade (förlänga filen).

DOWNFETCH tar innehållet i en fil och skickar det till värdenheten MF10. Till skillnad från READFILE, som i princip gör detta, anpassar DOWNFETCH teckenflödet så att det skall passa till FORTH-kompilatorn i MF10. Arbetar med "XON-XOFF" protokoll.

SECUREFETCH överför innehållet i en fil radvis till värdenheten MF10. Varje rad packas in i en ram som överförs i sin helhet. Detta ökar säkerheten vid överföringen med tar längre tid i anspråk. Arbetar med "XON- XOFF" protokoll.

DELFILE söker upp en stängd fil med filnamnet "FILENAME.EXT". Existerar filen elimineras den genom att alla blocken som filen ockuperat lämnas tillbaka till systemet och platsen för filens filhuvud markeras som ledigt.

SHOWDIR anropas, liksom READFILE, endast då backupenheten är ansluten till en PC. SHOWDIR anropas från "ShowDir", ett Modula-2 program på PC:n, och då listas alla filer som det aktuella biblioteket håller upp på skärmen.

Ovanstående rutiner är alla direkt knutna till ett kommando i Microflexen eller på PC:n. Förutom dessa rutiner finns ett antal subrutiner som anropas ifrån rutinerna ovan, eller på något annat sätt är en del av programmet.

INIT gör nödvändiga initeringar för att processorn skall fungera som planerat. Initierar stackpekaren, register för kommunikation, räknare för korrekt

överföringshastighet (9600 baud), samt nollställer diverse variabler och pekare. INIT kollar också om lagrings kapseln är en ny oanvänd eller om det redan finns filer i den. Detta görs genom att läsa två byte i kapseln som ska, om kapseln redan är initierad, innehålla en speciell kod. Om kretsen befinns vara oanvänd anropas nästa rutin PROMINIT.

PROMINIT anropas om den aktuella kretsen är oanvänd. Flaggorna som indikerade att kretsen inte var initierad sätts. Vidare initieras de filvariabler som är knutna till filerna i just den kapseln, till exempel sätts occupied-flaggorna på alla block till FALSE.

GETCHAR läser ett tecken från ringbufferten och placerar detta i variabeln "CHAR". I rutinen finns en löpvariabel "Wait" som räknas upp varje gång GETCHAR finner ringbufferten vara tom och som nollställs vid motsatsen. Då "Wait" uppnått en viss förutbestämd gräns (256 ggr) anses det föreligga ett fel på överföringen och felindikation erhålles, fungerar som en slags timeout-period.
Om sändningen till backupenheten är avstängd, på grund av att ringbufferten varit på väg att svämma över (se rutinen INTHANDLER), slår GETCHAR på sändningen då antalet lediga platser i ringbufferten överstiger 25 st.

SENDCHAR sänder via seriekanalen iväg tecknet som ligger i variabeln "CHAR".

SENDA sänder iväg innehållet i ackumulatorn.

MSDELAY utför en dummy-loop under 1 millisekund.

ERROR anropas för felindikering.

READFILENAME läser in filnamnet på den fil som avses att operera på och lägger det i strängvariabeln "FileName". Om filnamnet som mottages är anpassade efter MS-DOS normer kan det hända att det "hänger med" biblioteks och under-biblioteks namn typ "\BIB1\BIB2\FILENAME.EXT". READFILE ignorerar backslash "\" och alla tecken före backslash.

COMPARE jämför det av READFILE inlästa filnamnet med filnamnet i en av filerna i biblioteket.

COMPARENAME anropar COMPARE en gång för varje fil som måste sökas igenom innan antingen den rätta filen hittas eller tills hela biblioteket sökts igenom.

LASTBLOCK letar upp sista blocket i den aktuella filen och sätter en pekare "ActBlockP" att peka på det. Om filen är tom, d.v.s. att inga block är ockuperade, kommer pekaren att få ett felaktigt värde. Denna kontroll måste göras i den anropande rutinen.

NEWBLOCK letar upp ett ledigt block att lägga till filen genom att en pekare "ActBlockP" sätts att peka på det och samtidigt markeras blocket upptaget. Om inga lediga block finns sätts "ActBlockP = nil" (nil = 0000H, ett omöjligt värde).

CLEARBLOCK. Då en fil tas bort måste alla blocken filen disponerade lämnas

tillbaka till systemet. Detta görs genom att sätta den Booelska variabeln "occupied" i varje block till FALSE.

CONHEX gör om ett tecken till ASCII-representation.

SENDFRAME skickar iväg en ram till mottagaren.

EXFO_READ läser in en sträng och lägger den i en buffert.

EXFO_WRITE skriver ut den inlästa strängen.

EXECUTE sänder tillbaks den mottagna strängen följt av ett CR.

FORGETWORD sänder ut ordet "FORGET" följt av den tidigare inlästa strängen och ett CR.

CLEARDIR bör endast användas i nödfall då systemet har kraschat. Tar då bort alla filer och återlämnar alla block till systemet. På så sätt kan kretsen nollställas och användas igen. Rutinen anropas ifrån ett program "ClearDir" på PC:n.

NMIHANDLER. Var 20:e millisekund fås in till processorn ett avbrott som inte kan maskas bort (NMI - Non Maskable Interrupt). Detta tas omhand genom att återvända till det ställe i programmet som vid tiden för avbrottet stod i tur att exekveras. Adressen dit återfinns på stacken.

INTHANDLER. Då ett tecken anländer till Microstore genereras ett interrupt i processorn som då söker upp aktuell hoppadress i IRQ-vektorn. Adressen är den till INTHANDLER som tar hand om tecknet och lägger in det i ringbufferten (128 byte stor). Om antalet lediga platser i ringbufferten understiger 4 st stänger rutinen av sändningen till backupenheten med "XON-XOFF".

SLUT fungerar endast som markering att programmet är slut för det program som genererar symbollistan på PC:n.

5.4 Interface-program på PC.

Som nämts tidigare finns möjligheten att via seriekanalen koppla in backup-enheten direkt till en PC. På PC:n finns fem stycken program (se appendix) som gör det möjligt att etablera direktkontakt med Microstore. Alla programmen är skrivna i Modula-2 och aktiverar en motsvarande rutin i backupenheten.

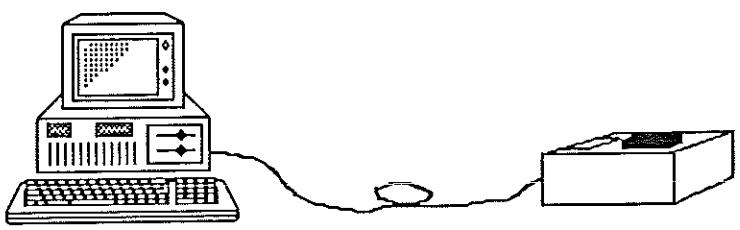


fig. 21. PC:n kopplad direkt till backupenheten.

SHOWDIR. Listar biblioteket, i den aktuella E^2PROM kapseln, på bildskärmen i form av filernas namn och deras storlek.

DUMPFILE. Tar en fil på PC:n och sänder ned den till Microstore-enheten via seriekanalen. Överföringshastighet = 9600 baud.

READFILE. Listar ut en fil i Microstore på skärmen. För att synkronisera överföringen används "XON-XOFF" protokoll.

DELFILE. Anropar rutinen "DELFILE" i Microstore som tar bort en fil.

CLEARDIR. Får endast användas i nödfall. Om någon oupptäckt bugg i mjukvaran gör att systemet kraschar finns möjligheten att åtminstone rädda lagringskretsen så att den går att använda igen. CLEARDIR tar bort alla filer och återlämnar alla minnesblock till systemet.

6. Avslutning.

Microflex10 är den enhet kring vilket det mesta rör sig även om exjobbet har haft sin tonvikt på Microstore. MF10,uppbyggd på ett kretskort, är i grundutförandet s.a.s. en "fast konstruktion". Den nya styr- och reglerutrustningen Microflex20 är av racktyp, d.v.s. att det finns uttag för flera moduler i racken.

En av dessa moduler är en modifierad variant av den ursprungliga Microstore. En av de viktigare skillnaderna består i att ytterligare en typ av minneskrets, statiskt CMOS-RAM, kan anslutas till den modifierade backupenheten. Dessa minneskretsar har formen av ett tjock kreditkort och innehåller, förutom 32 kbyte RAM, batteri och kretsar till batteribackup. Denna möjlighet gör systemet betydligt snabbare att använda. Ett helt kartotek av sådana kretsar kan byggas upp var och en innehållande data eller filer som kan tas fram och användas vid lämpliga tillfällen.

Även E^2PROM:et finns i formen av ett kreditkort och kan anslutas till samma uttag som ovan nämda RAM då dessa arbetar med samma spänningar. Däremot är EPROM-kretsarna ute ur bilden vid användande av detta filhanteringssystem (se 5.3.1).

En annan viktig skillnad är att i MF20 kommer överföringen mellan den modifierade Microstore och CPU-kortet att ske parallellt vilket gör systemet betydligt snabbare än vid seriell överföring.

APPENDIX

A. Användarhandledning.

B. Flödesscheman.

C. Datablad.

D. Programlistor.

APPENDIX A

Användarhandledning.

ANVÄNDARHANDLEDNING.

1. Uppstart.

Anslut först, med en RS232-anpassad kabel, mottagarenhetens (vanligtvis en Microflex10) serieport till PC:ns seriesnitt.

För att kunna använda sig av filhanteringspaketet behövs tillgång till ett antal filer på PC:n. Leta därför upp filerna

I. SCODE.4TH
II. FSCODE.4TH
III. FILECOMM.4TH
IV. EXEC.4TH
V. COMIC2.EXE
VI. DOWNFILE.EXE

och ladda ned filerna I., II., III. och IV. till mottagaren, Microflex10, i nu nämd ordning. Detta görs genom att köra programmet i fil VI. Skriv därför

DOWNFILE <cr> .

Filerna kan skickas ned med hjälp av en batchfil "FBATCH.BAT"

2. Orden.

Etablera direktkontakt med MF10 genom att skriva

COMIC2 <cr>

och kontrollera innehållet i WORDS. Ordlistan över exekverbara ord och variabler i MF10 skall nu ha utökats med tolv (26) nya ord. De nya orden är

1. UT \ Variabel till SCODE.
2. SCODE \ Ord som tar hand om inkommande tecken \ som skall kompileras.
3. TEMP \ Variabel till FSCODE.
4. NUM \ Dito.
5. LOP \ Dito.
6. CSUMI \ Dito.
7. CSUME \ Dito.
8. ACK \ ACKowledgement.
9. NAK \ Negative AcKnowledgement.
10. EXPCT \ Underprogram till FSCODE.
11. CONV \ Dito.
12. SUMTIB \ Dito.
13. FSCODE \ Avformatterar inkommande ramar och \ kompilerar innehållet.
14. CREATEFILE \ Skapar en fil.
15. OPENFILE \ Öppnar en fil.
16. CLOSEFILE \ Stänger en fil.
17. READFILE \ Läser en fil.
18. WRITEFILE \ Skriver i en fil.
19. DOWNFETCH \ Begär ned innehållet i en fil. Aktiverar \ SCODE.
20. SECUREFETCH \ Begär ned innehållet i en fil. Aktiverar \ FSCODE
21. DELFILE \ Tar bort en fil.

22. SHOWDIR \ Visar innehållet i aktuellt bibliotek.
23. STX \ Start of TeXt.
24. ETX \ End of TeXt.
25. EXEC" \ Gör det möjligt att exekvera ett ord som \ ännu inte finns i ordlistan.
26. FORGW" \ Gör det möjligt att ta bort ett ord som är \ definierat senare.

De elva kommandona, 14 - 22 samt 25 - 26 ovan, opererar mot något av massminnesmedierna PC eller Microstore.

3. PC som massminnesmedium.

Då en PC används som massminnesmedium är seriesnitten ihopkopplade som ovan. Kommandona kan nu exekveras på två olika sätt, dels enskilt dels som en del i ett större program.

A. ENSKILD EXEKVERING.

Vid enskild exekvering måste först direktkontakt etableras med MF10 genom att, som ovan, exekvera PC programmet COMIC2. Via tangentbordet kommunicerar man nu direkt med MF10.

Till en del av kommandona hör ett filnamn "FILENAME.EXT" eller en valmöjlighet "OPTION". För filnamnet gäller att det får vara 31 tecken långt och det inkluderar även biblioteksnamn typ "\BIB1\BIB2\FILENAME.EXT". För "OPTION" gäller samma regler, 31 tecken långt inalles.

Vid exekvering av ett kommando skrivs

KOMMANDO <cr> .

	Skriv.	Åtgärd.
-	CREATEFILE <cr>	Förfrågan om filnamnet på filen som skall skapas fås, "Filename : "
	FILENAME.EXT <cr>	Om "FILENAME.EXT" redan är en existerande fil öppnas den, annars skapas den och lämnas öppen.
-	OPENFILE <cr>	"Filename : "
	FILENAME.EXT <cr>	Existerar filen öppnas den. Om den inte existerar lämnas ett erbjudande om att skapa filen, (Y/N).
-	CLOSEFILE <cr>	Stänger den öppna filen. Endast en fil kan vara öppen åt gången.
-	READFILE <cr>	Om ingen fil är öppen fås ett felmeddelande, annars listas den öppna filen ut på skärmen.
-	WRITEFILE <cr>	Om ingen fil är öppen fås ett felmeddelande, annars fås en förfrågan, "Overwrite or Continue the file ? (O/C) : ", om man vill skriva från början i filen (och då ev. skriva över gammalt innehåll)

eller om man vill fortsätta skriva i slutet av filen (där man senast slutade).

textmassa . . . . Lägger in allt som skrivs från tangentbordet i den öppna filen.

- DOWNFETCH <cr> Om ingen fil är öppen fås felmeddelande men ordet SCODE aktiveras. För att ta sig ur SCODE (deaktivera) skriv : "QUIT".
DOWNFETCH aktiverar SCODE och beordrar PC:n att skicka ned innhållet i den öppna filen till MF10. SCODE tar emot innehållet radvis, kompilerar det och lägger in det i ordlistan över befintliga FORTH-ord. Då hela innehållet är överfört stängs filen.

- SECUREFETCH <cr> Om ingen fil är öppen fås felmeddelande men ordet FSCODE aktiveras. För att ta sig ur FSCODE (deaktivera) skriv : "ctrlA 0000".
SECUREFETCH ger samma resultat som ovanstående kommando med den skillnaden att här skickas innehållet över i ramar med checksummekontroll. Detta kommando tar ca 50% längre tid än DOWNFETCH.

- DELFILE <cr> "Filename : "
FILENAME.EXT <cr> Om någon fil är öppen fås ett felmeddelande, annars tas den specificerade filen bort.

- SHOWDIR <cr> "Option :"
OPTION <cr> OPTION kan vara de vanliga tillägg som görs till MS-DOS kommandot DIR, såsom diskettstation, biblioteksnamn, /W, /P, wildcharacters etc. Default-Option är "*.*" . Listar det av OPTION önskade biblioteket.

- EXEC" <cr> Saknar innebörd vid enskild exekvering.

- FORGW" <cr> Saknar innebörd vid enskild exekvering.

B. EXEKVERING I PROGRAM.

När man definierar nya FORTH-ord använder man sig av tidigare definierade ord. På samma sätt kan också filhanteringsprogrammet användas, d.v.s. som delar i nya ord.

Då kommandona exekveras enskilt skrivs allting som hör till ett kommando såsom filnamn, textmassa, options, in ifrån tangentbordet. När kommandona exekveras i ett program kommer allt detta ifrån Microflex-enheten. För att PC:n skall kunna skilja dessa bägge fall åt omges allt som är knutet till ett kommando med STX och ETX.

Ex.1 CREATEFILE STX ." FILENAME.EXT" ETX
WRITEFILE STX 43 EMIT ." textmassa . . . . " 100 0 DO I . LOOP ETX

Kommandon	Kommentarer
- CREATEFILE	Används, med ovanstående modifiering (STX och ETX), som under punkt A.
- OPENFILE	Dito.
- CLOSEFILE	Här behövs ej STX och ETX.
- READFILE	Kan användas men saknar ofta relevans i ett program.
- WRITEFILE	Då detta kommando används i ett program måste man skicka med information om var i filen man vill skriva in data, i början eller i slutet (Overwrite/Continue). Denna information måste följa direkt på ordet STX och skrivs som det *hexadecimala värdet* , enligt ASCII-tabellen, av stora C eller O följt av kommandot EMIT (se Ex.1).
- DOWNFETCH	Kommandot hämtar ned innehållet i en öppen fil på PC:n, ofta bestående av FORTH-ord. Vill man sedan exekvera det nya ordet måste man lägga till ordet i samma fil. Ex. 2 : P1 \ Skriver ut talen 0 - 100 på skärmen. 100 0 DO I . LOOP ; P1 OBS att i filer som hämtas ned på detta sättet får *ej innehålla något av filhanteringskommandona som presenteras i denna handledning*. Denna typ av nestling klarar inte programvaran av för närvarande. För att ta sig runt detta problem används orden EXEC" och FORGW" (se dessa ord). Kommandot stänger den öppna filen efter överförandet.
- SECUREFETCH	Här gäller samma regler som för DOWNFETCH.
- DELFILE	Se till att alla filer är stängda innan detta kommando exekveras och bifoga filnamnet omgivet av STX, ETX.
- SHOWDIR	Detta kommando kan endast exekveras direkt från tangentbordet och går inte att använda som ett ord i ett program.
- EXEC"	Gör det möjligt att exekvera ord i ordlistan som ännu ej är definierade. Används med fördel tillsammans med orden DOWNFETCH och SECUREFETCH. Ex. 3 : P1 \ Skriver ut talen 0 - 100 på skärmen. 100 0 DO I . LOOP ; \ Här exekveras inte P1 som i ex. 2 ovan.

```
: P2  \ Hämtar ned P1 ovan.
OPENFILE STX ." P1.4TH" ETX
DOWNFETCH
" P1" EXEC" ;  \ P1 exekveras här.
```

- FORGW" — Gör det möjligt att ta bort ett senare definierat ord ur ordlistan (WORDS) i MF10. Programmet P2 i ex. 3 ovan kan då utökas enligt följande.

Ex. 4
```
: P2  \ Hämter ned P1.
OPENFILE STX ." P1.4TH" ETX
DOWNFETCH
" P1" EXEC"    \ P1 exekveras här.
" P1" FORGW" ; \ P1 tas bort ur ordlistan.
```

4. Microstore som massminnesmedium.

Då backupenheten Microstore skall användas som massminne ansluts den RS232-anpassade kabeln mellan ett av seriesnitten hos MF10 och seriesnittet på backupenheten. Kommandona i MF10 kan exekveras enskilt men på ett lite annorlunda sätt. Dock kan alla FORTH-program i MF10, som innehåller filhanterings- kommandon, som är skrivna med tanke på att kunna användas mot en PC som massminne, också användas mot Microstore. Undantaget är READFILE som går att exkvera mot en PC men inte mot Microstore då ingen bildskärm finns tillgänglig.

A. ENSKILD EXEKVERING.

För att kunna exekvera kommandona enskilt då MF10 arbetar mot Microstore måste PC:n anslutas till COM1, MF10's ena seriesnitt, och backupenheten ansultas till CONSOLE, MF10's andra seriesnitt. Etablera direktkontakt med MF10 som tidigare via kommandot

COMIC2 <cr>.

För att sedan kunna exekvera kommandona mot Microstore måste MF10 ställas att arbeta mot CONSOLE, dit bakupenheten är ansluten. Detta görs genom att skriva

CONSOLE <cr>.

Till skillnad mot när MF10 arbetade mot en PC som massminne fås här inga ledtexter och man får själv hålla reda på vilka filer som finns tillgängliga. Ett exempel på hur en exekvering kan gå till.

Ex. 3.
```
CONSOLE <cr>
OPENFILE STX .( FILENAME.EXT) ETX <cr>
DOWNFETCH <cr>
COM1 <cr>
```

Eftersom inga ledtexter finns måste konstruktionen "STX .(filnamn) ETX" användas.

Kommando.	Kommentar.
- CREATEFILE	Skapar en fil och lämnar den öppen. Om filen

		redan existerar öppnas den. Filnamnet måste omges av STX-ETX (se Ex. 3.).
-	OPENFILE	Öppnar en fil, om filen inte finns fås ingen reaktion. Filnamnet omges av STX-ETX (Ex.3).
-	CLOSEFILE	Stänger en fil.
-	READFILE	Kan ej användas i denna tillämpnig.
-	WRITEFILE	Skriver ut tecken på den öppna filen. Direkt efter STX måste valet O/C (Overwrite/Continue) göras prcis som tidigare beskrivet (43 EMIT eller 4F EMIT).
-	DOWNFETCH	Hämtar innehållet i en fil på Microstore, laddar ned det i MF10 och kompilerar det. Här fås ingen kvittens av när laddningen är klar. Försöker man utföra kommandot DOWNFETCH utan att filen finns kan systemet låsa sig.
-	SECUREFETCH	Som DOWNFETCH.
-	DELFILE	STX .(filnamn) ETX.
-	SHOWDIR	Kan ej användas i denna tillämpning.
-	EXEC"	Saknar innebörd vid enskild exekvering.
-	FORGW"	Saknar innebörd vid enskild exekvering.

B. EXEKVERING I PROGRAM.

Ett program skrivet för MF10 skall kunna exekvera, utan modifieringar, på samma sätt både mot en PC och mot Microstore med ett undantag. Kommandot READFILE *kan* exekveras mot PC men denna möjlighet finns ej då programmet arbetar mot Microstore då ingen bildskärm finns. I övrigt gäller samma regler.

5. PC - Microstore.

Anslut PC:ns seriesnitt med backupenhetens seriesnitt. Det finns nu möjligheter att kommunicera direkt med Microstore. För det ändamålet krävs tillgång till ytterligare fem (5) filer :

SHOWDIR.EXE
READFILE.EXE
DUMPFILE.EXE
DELFILE.EXE
CLEARDIR.EXE

Var och en av dessa utgör ett kommando som opererar på filer i Microstore.

	Skriv.	Åtgärd.
-	SHOWDIR <cr>	Listar upp biblioteket,som hålls av den aktuella minneskretsen, i form av filernas namn och deras storlek.
-	READFILE <cr>	"Enter filename :"
	FILENAME.EXT <cr>	Listar upp innehållet i den sökta filen på skärmen.
-	DUMPFILE <cr>	"Enter filename :"

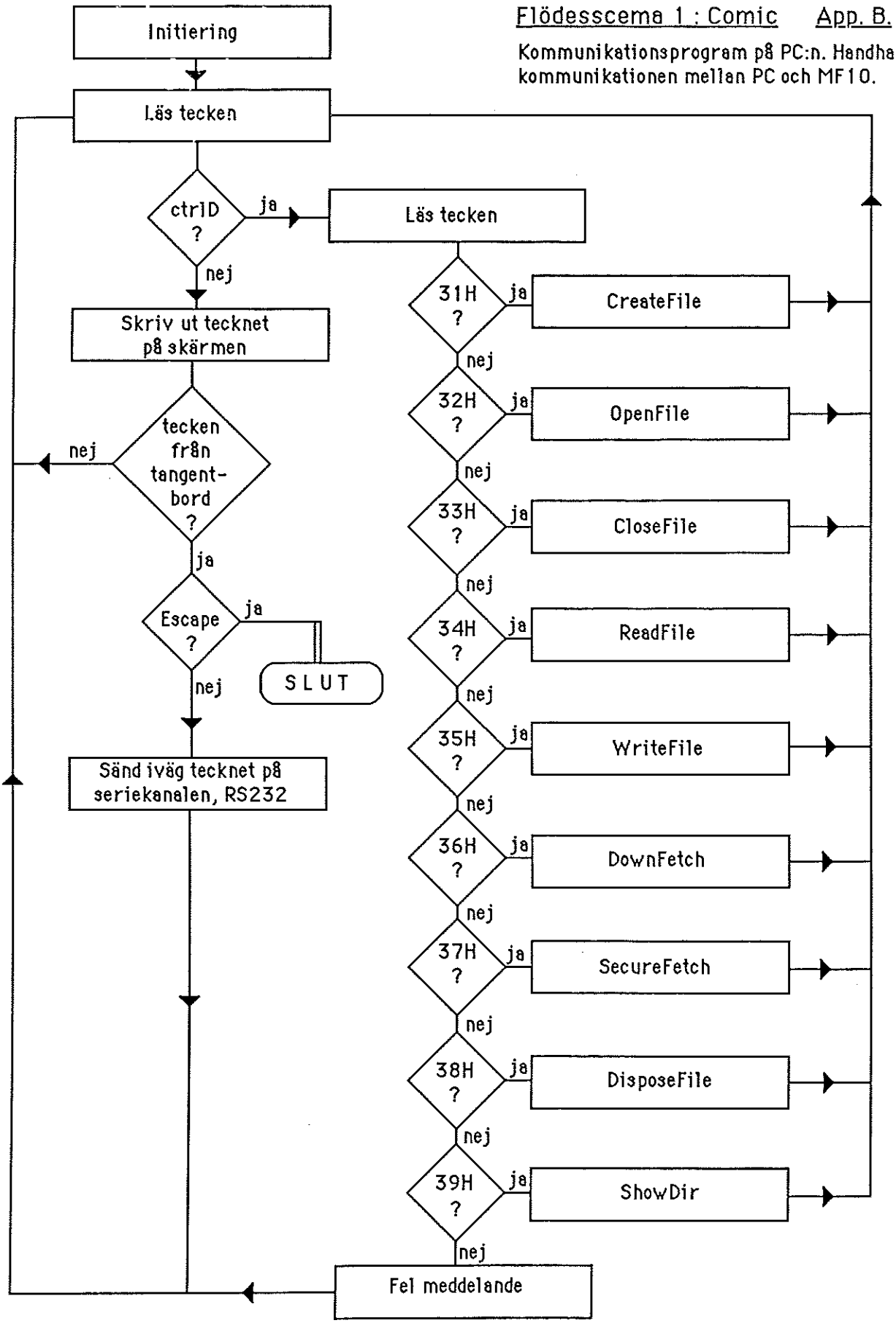
	FILENAME.EXT <cr>	Letar upp den sökta filen på PC:n och skickar ned till Microstore.
-	DELFILE <cr>	"Enter filename :"
	FILENAME.EXT <cr>	Tar bort den sökta filen från biblioteket i den aktuella minneskapseln på Microstore.
-	CLEARDIR <cr>	Används med försiktighet. Tar bort alla filer ur det aktuella biblioteket.

APPENDIX B

Flödesscheman.

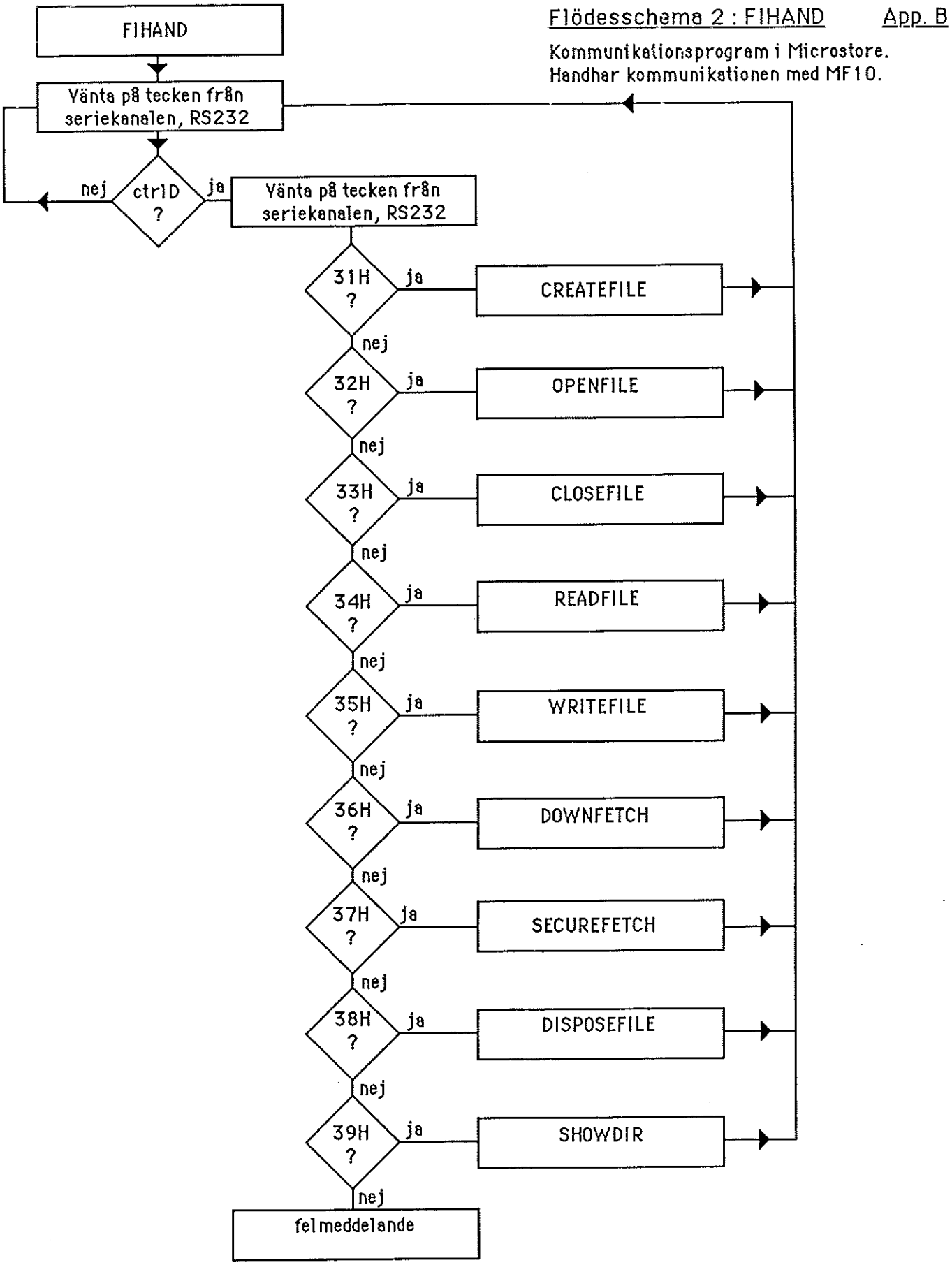
Flödesscema 1 : Comic App. B.

Kommunikationsprogram på PC:n. Handhar kommunikationen mellan PC och MF10.



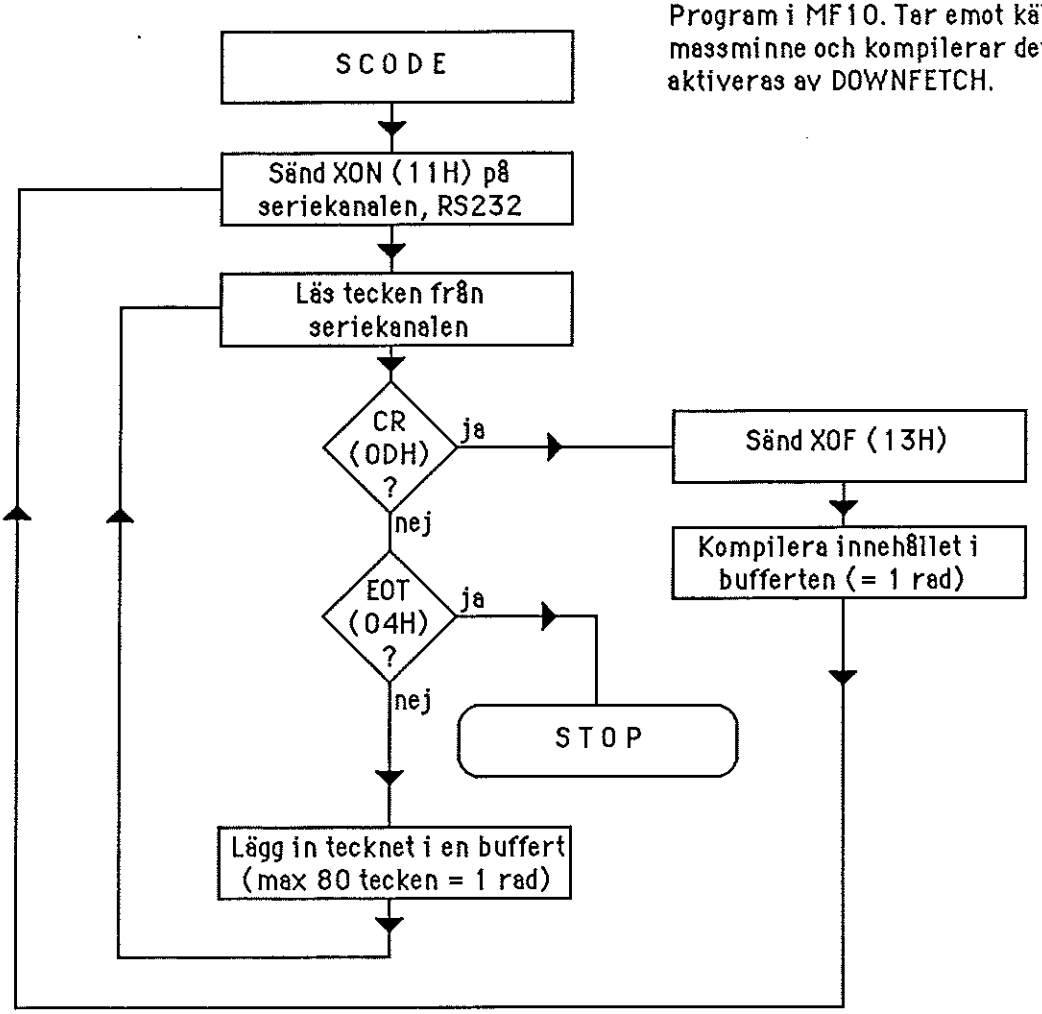
Flödesschema 2 : FIHAND

Kommunikationsprogram i Microstore.
Handhar kommunikationen med MF10.



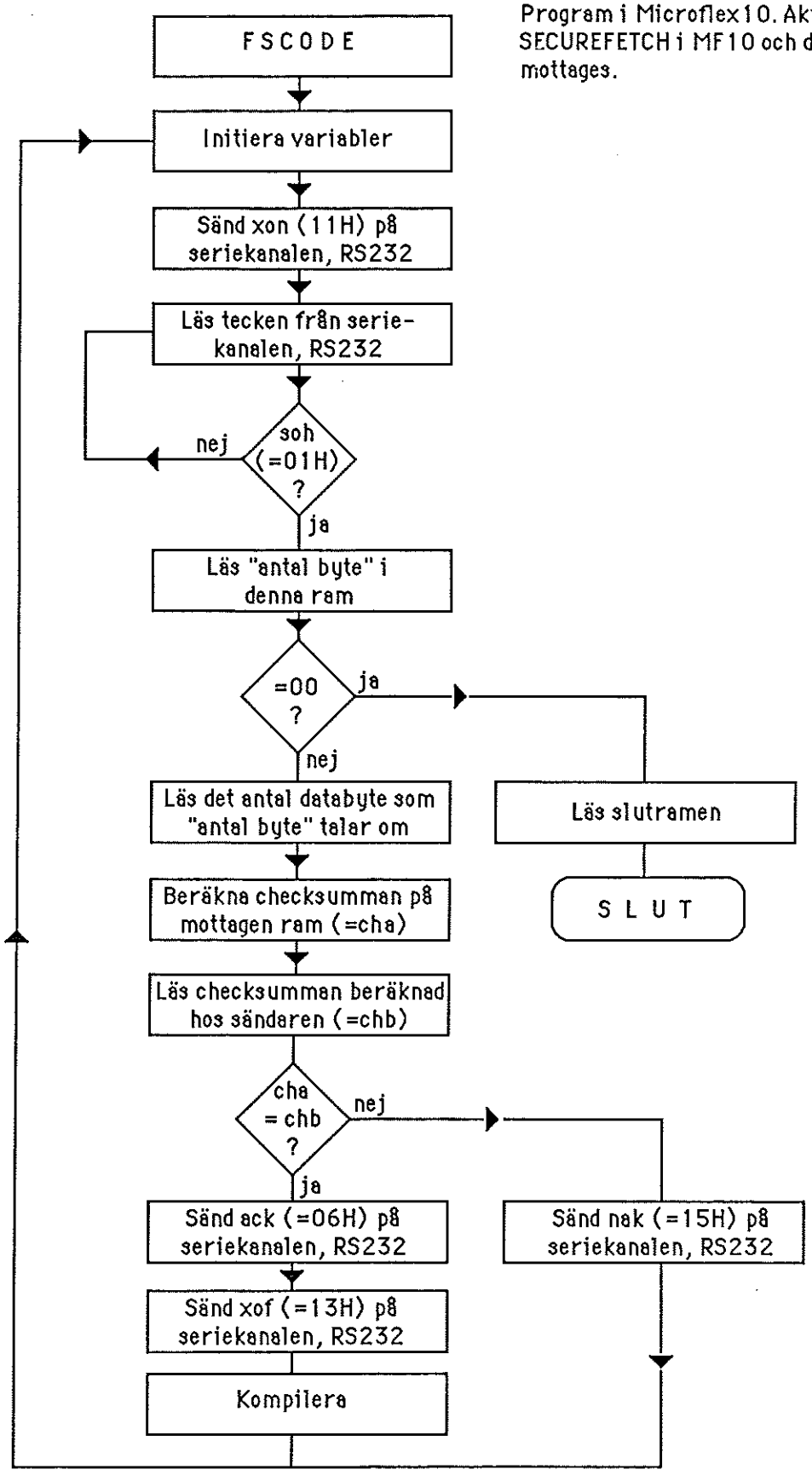
Flödesschema 3 : SCODE

Program i MF10. Tar emot källkod från externt massminne och kompilerar det radvis. SCODE aktiveras av DOWNFETCH.

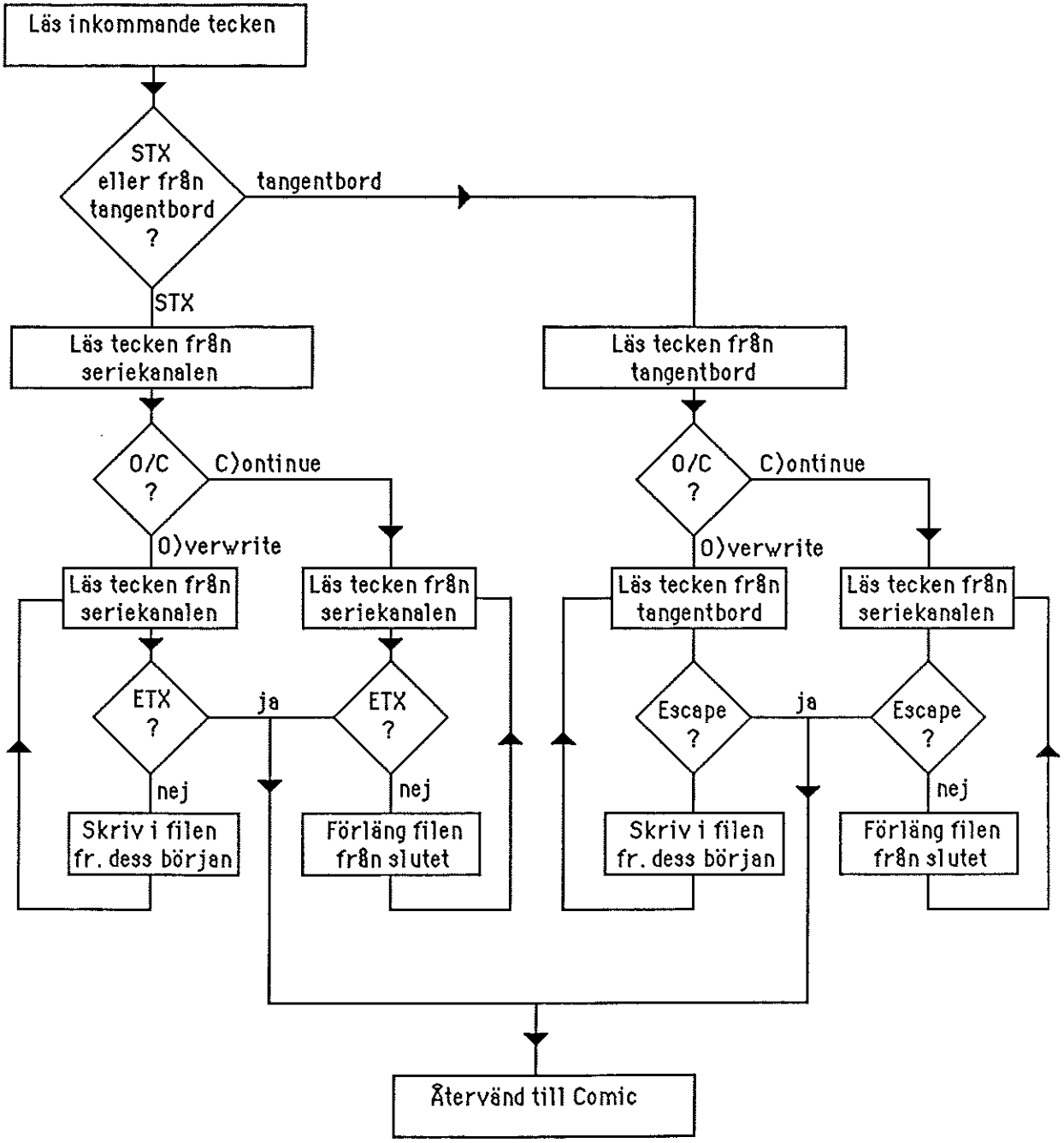


Flödesschema 4 : FSCODE

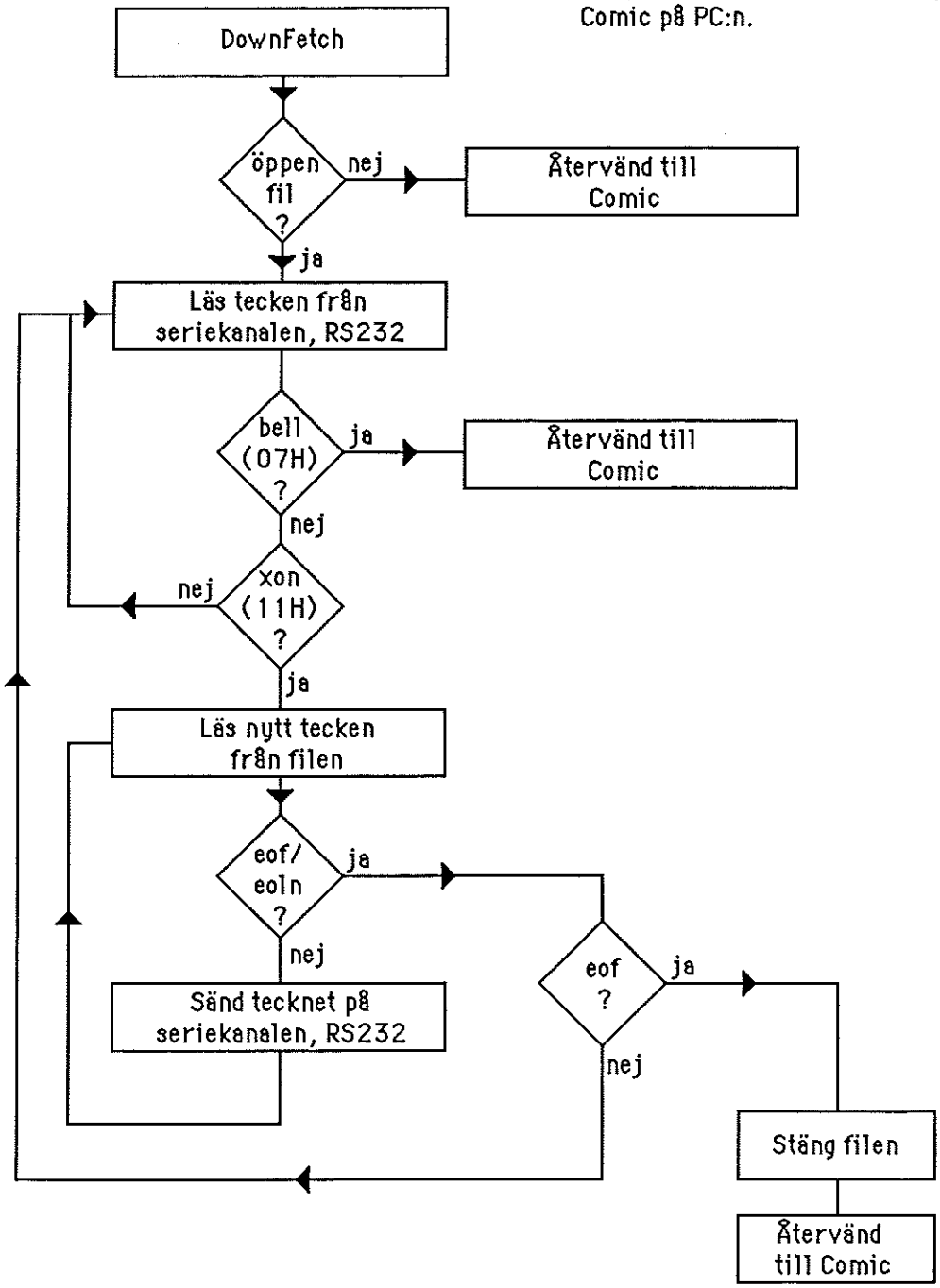
Program i Microflex10. Aktiveras av kommandot SECUREFETCH i MF10 och deaktiveras då slutram mottages.



Program i modul FileHandler på PC. Anropas av Comic på PC:n.

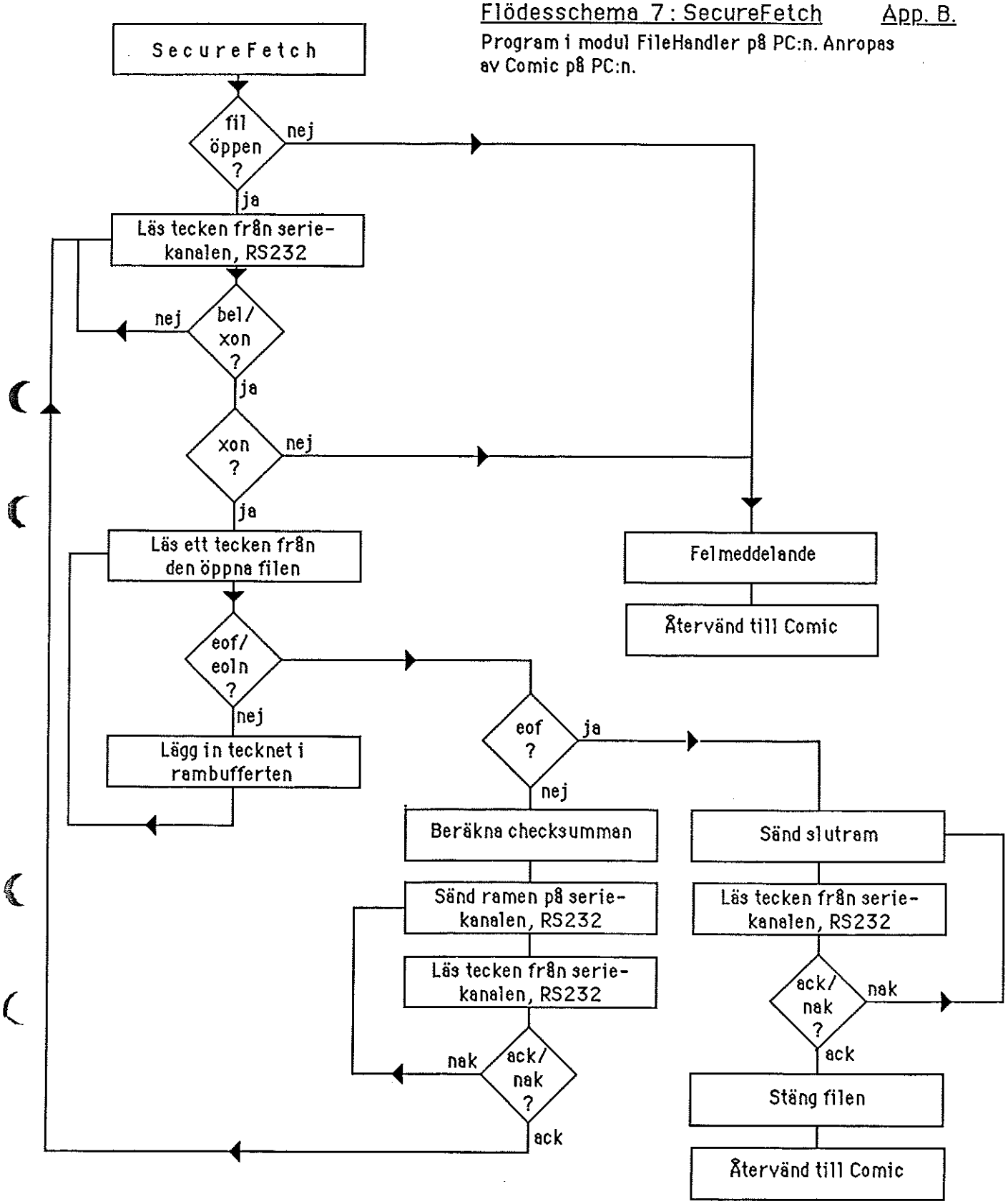


Program i modul FileHandler på PC:n. Anropas från Comic på PC:n.

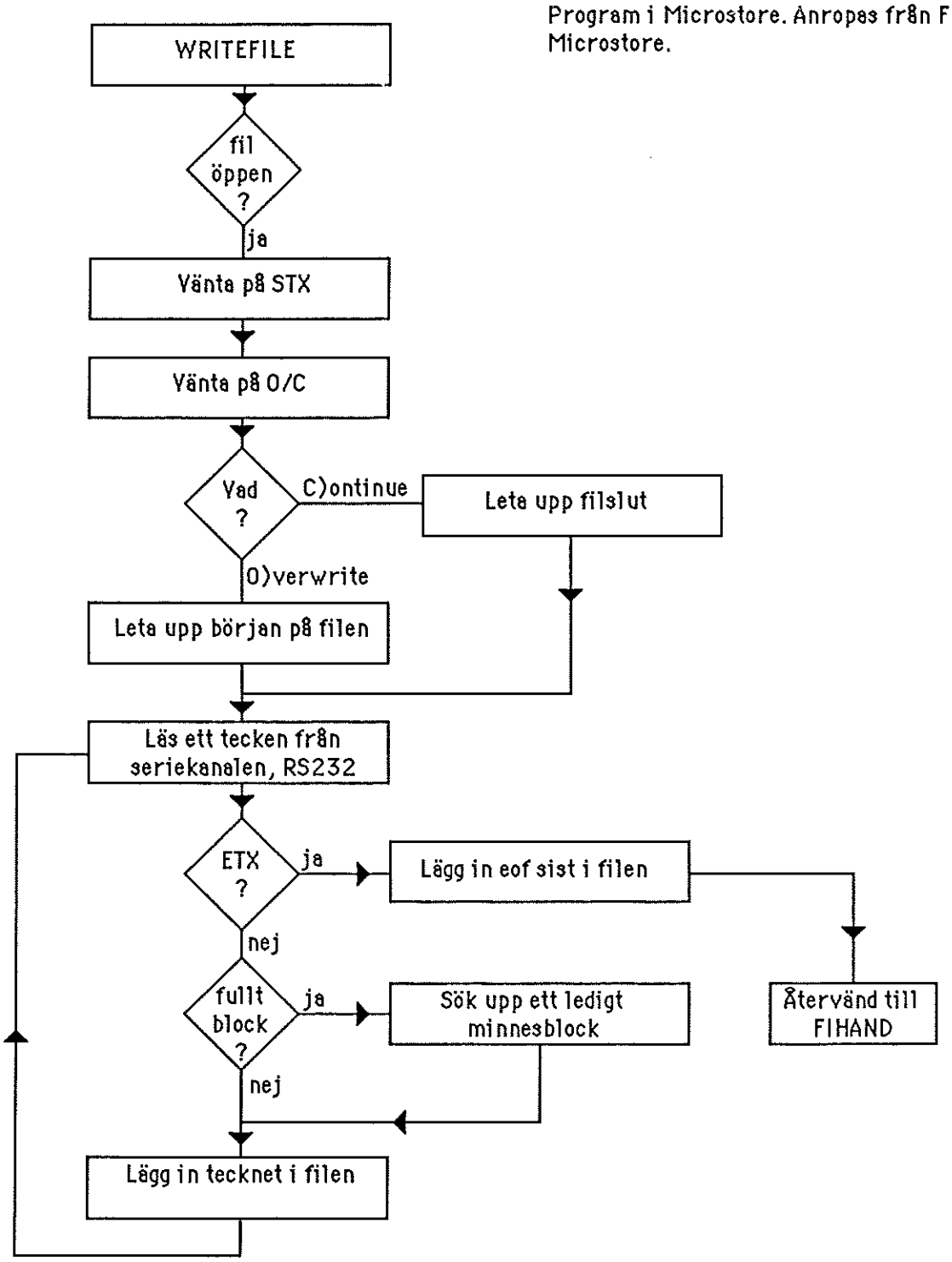


Flödesschema 7 : SecureFetch

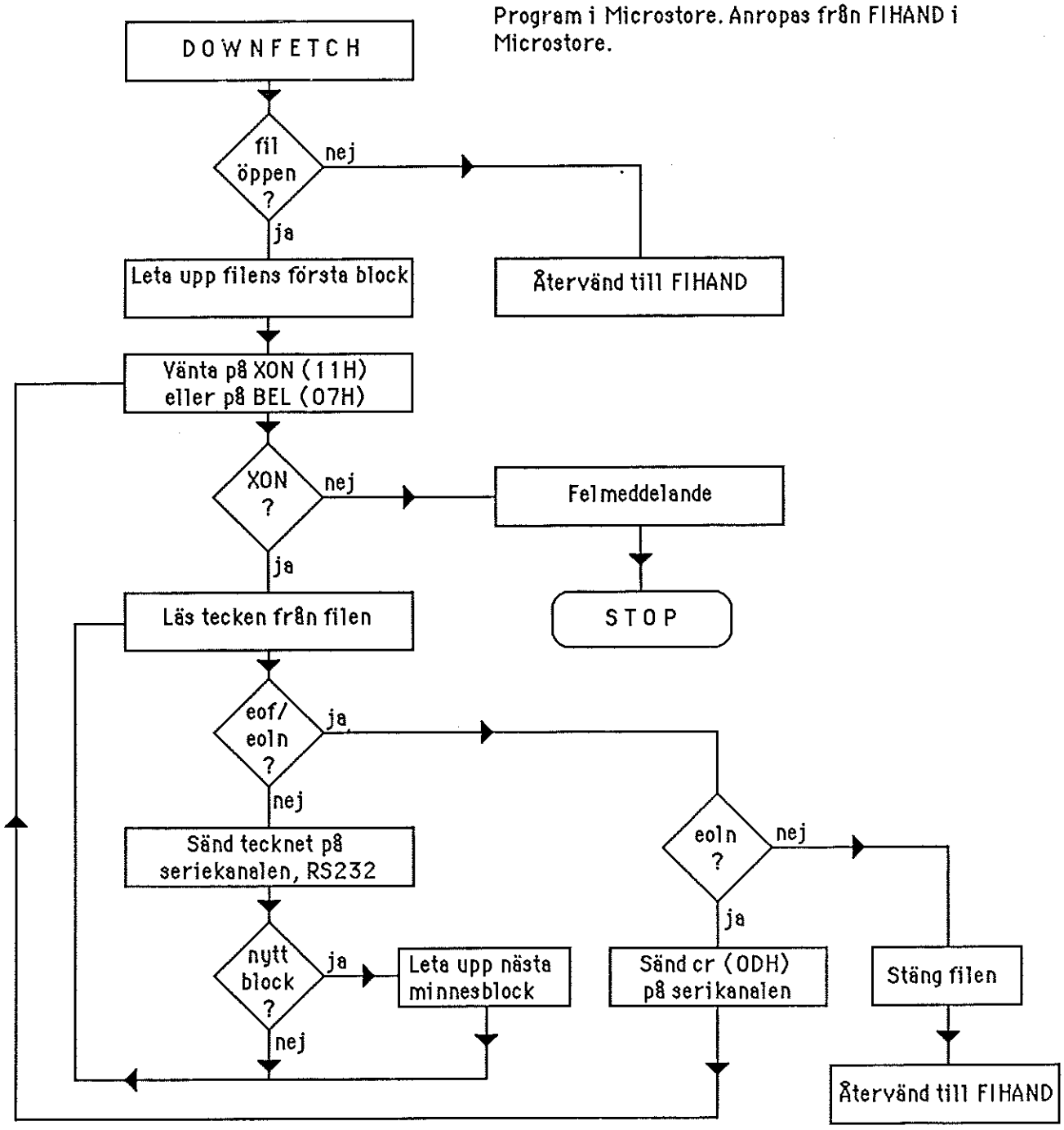
Program i modul FileHandler på PC:n. Anropas av Comic på PC:n.



Program i Microstore. Anropas från FIHAND i Microstore.

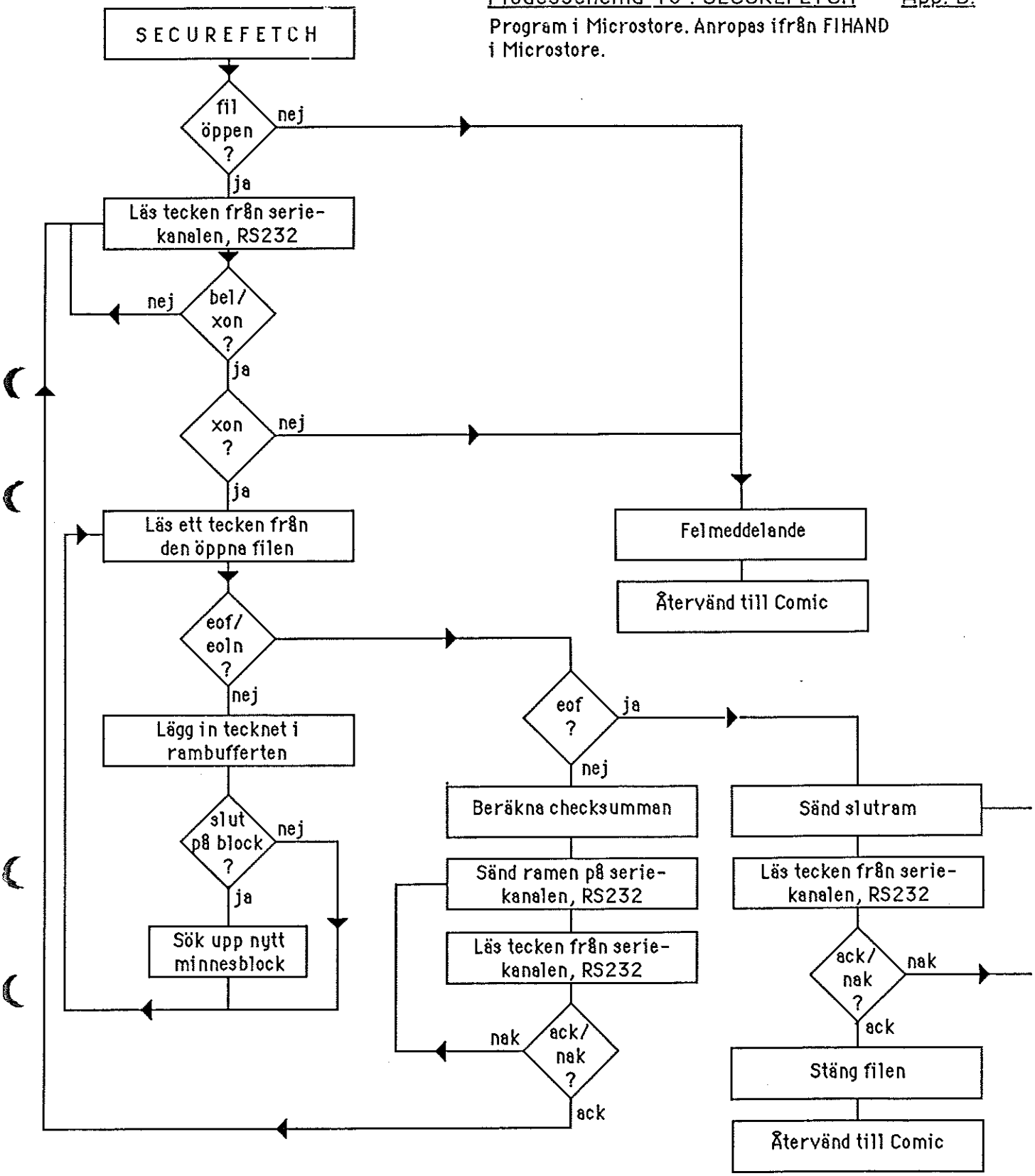


Program i Microstore. Anropas från FIHAND i Microstore.



Flödesschema 10 : SECUREFETCH App. B.

Program i Microstore. Anropas ifrån FIHAND i Microstore.



APPENDIX C

Datablad.

R6501Q AND R6511Q ONE-CHIP MICROPROCESSOR

INTRODUCTION

The Rockwell R6501Q and R6511Q are extended, high performance 8-bit NMOS-3, single chip microprocessors, and are compatible with all members of the R6500 family.

The devices contain an enhanced R6502 CPU, an internal clock oscillator, 192 bytes of Random Access Memory, and versatile interface circuitry. The interface circuitry includes two 16-bit programmable timer/counters, 32 bidirectional input/output lines (including four edge sensitive lines and input latching on one 8-bit port), a full-duplex serial I/O channel, ten interrupts and bus expandability. A full 16-bit address bus and 8-bit data bus provide accessing to 65K bytes of external memory.

The devices come in a 64-pin Quad Inline package (QUIP).

The devices may be used as a CPU-RAM-I/O counter device in multichip systems or as an emulator for the R6500/11 family of microcomputers. They provide all R6500/11 interface lines, plus the address bus, data bus and control lines to interface with external memory.

SYSTEMS DEVELOPMENT

Rockwell supports development of the devices with the Rockwell Design Center System and the R6500/* Personality Set: Complete in-circuit emulation with the Personality Set allows total systems test and evaluation.

This data sheet is for the reader familiar with the R6502 CPU hardware and programming capabilities. For additional information see the R6501Q Product Description, (Document Order Number 2145) or the R6511Q Product Description, (Document Order Number 2133).

ORDERING INFORMATION

Part Number	Package Type	Frequency Option	Temp. Range
R6501Q	Plastic (QUIP)	1 MHz	0°C to 70°C
R6501AQ	Plastic (QUIP)	2 MHz	0°C to 70°C
R6511Q	Plastic (QUIP)	1 MHz	0°C to 70°C
R6511AQ	Plastic (QUIP)	2 MHz	0°C to 70°C

FEATURES

- Enhanced R6502 CPU
  - —Four new bit manipulation instructions
    - Set Memory Bit (SMB)
    - Reset Memory Bit (RMB)
    - Branch on Bit Set (BBS)
    - Branch on Bit Reset (BBR)
  - —Decimal and binary arithmetic modes
  - —13 Addressing modes
  - —True indexing
- 192-byte static RAM
- 32 bidirectional, TTL-compatible I/O lines (four ports)
- One 8-bit port may be tri-stated under software control
- One 8-bit port may have latched inputs under software control
- Two 16-bit programmable counter/timers, with 3 latches
  - —Pulse width measurement
  - —Pulse generation (1 symmetrical, 1 asymmetrical)
  - —Interval timer
  - —Event counter
  - —Retriggerable interval timer
- Serial Port — Full Duplex, Buffered UART
  - —Receiver Wake Up and Transmitter End of Transmission Features
  - —Programmable Standard Asynchronous Baud Rates from 50 to 125K bits/sec at 2 MHz

    Satisfies SMPTE 422 Broadcast Standard (8 Data, Parity, 1 Stop) at 38.4K bits/sec
  - —Programmable 5-8 bit Character Lengths, with or without parity
  - —Receiver Error Detection for Framing, Parity, and Overrun
  - —Synchronous Shift Register alternate mode (250KC at 2 MHz)
- Ten interrupts
  - —Four edge-sensitive lines; two positive, two negative
  - —Two counter underflows
  - —Serial data receiver buffer full
  - —Serial data transmitter buffer empty
  - —Non-maskable
  - —Reset
- Full data and address pins for 65K bytes of external memory
- Flexible clock circuitry
  - —2 MHz or 1 MHz internal operation
  - —Internal clock with external XTAL at four times internal frequency (R6501Q) or two times internal frequency (R6511Q)
  - —External clock input divided by one or four (R6501Q) or one or two (R6511Q)
- 68% of the instructions have execution times less than 2 μs at 2 MHz
- NMOS-3 silicon gate, depletion load technology
- Single +5V power supply
- 12 mW stand-by power for 32 bytes of the 192-byte RAM
- 64-pin QUIP
- R6501Q has pullup resistors on PA, PB, and PC
  R6511Q has no pullup resistors

FUNCTIONAL DESCRIPTION

CENTRAL PROCESSING UNIT (CPU)

The internal CPU of the device is a standard R6502 configuration with the standard R6502 instructions plus 4 new bit manipulation instructions. These new bit manipulator instructions form an enhanced R6502 instruction set and improve memory utilization efficiency and performance.

Set Memory Bit (SMB #,ADDR.)

This instruction sets to "1" one bit of the 8-bit data field specified by the zero page address (memory or I/O port). The first byte of the instruction specifies the SMB operation and which one of 8 bits to be set. The second byte of the instruction designates the address (0-225) of the byte or I/O port to be operated upon.

Reset Memory Bit (RMB #,ADDR.)

This instruction is the same operation and format as the SMB instruction except a reset to "0" of the bit results.

Branch on Bit Set Relative (BBS #,ADDR.,DEST)

This instruction tests one of 8 bits designated by a 3-bit immediate field within the first byte of the instruction. The second byte is used to designate the location of the byte or I/O port to be tested within the zero page address range. The third byte of the instruction is used to specify the 8-bit relative address to which the instruction branches if the bit tested is a "1". If the bit tested is not set, the next sequential instruction is executed.

Branch on Bit Reset Relative (BBR #,ADDR.,DEST)

This instruction is the same operation and format as the BBS instruction except that a branch takes place if the bit tested is a "0".

Random Access Memory (RAM)

The RAM consists of 192 by 8 bits of read/write memory with an assigned page zero address of 0040 through 00FF. The devices provide a separate power pin (V_{RR}) which may be used for standby power. In the event of the loss of V_{CC} power, the lowest 32 bytes of RAM data will be retained if standby power is supplied to the V_{RR} pin.

Clock Oscillator

The clock oscillator provides the basic timing signals. A reference frequency can be generated with the on board oscillator (with external crystal) or an external reference source can be driven into the XTLI pin. If the XTLO pin is left floating, the reference frequency is internally divided by four (R6501Q) or two (R6511Q) to obtain the internal clock. The internal clock is then available as an output at the Ø2 pin. The XTLI pin may be used as an undivided clock input by connecting XTLO to V_{SS}, in which case the internal division circuitry is bypassed and the device operates at the reference frequency.

Parallel Input/Output Ports

The devices have 32 I/O lines grouped into four 8-bit ports (PA, PB, PC, PD). Ports A through C may be used either for input or output individually, or in groups of any combination. The R6501Q has pullup resistors on PA, PB and PC. The R6511Q has no pullup resistors. Port D may be used as all inputs or all outputs. It has active pull-ups.

Port A (PA) can be programmed as a standard parallel 8-bit I/O port or under software control as serial I/O lines, counter I/O lines, positive (2) and negative (2) edge detects, or an input data strobe for the Port B (PB) input latch.

Port B (PB) can be programmed as an I/O port with latched input enabled or disabled.

Port C (PC) can be programmed as an I/O port, as an abbreviated bus, as a multiplexed bus, or as part of the full address mode. In the full address mode pins PC6 and PC7 serve as addresses A13 and A14, respectively; PC0–PC5 are I/O pins.

Port D (PD) functions as an I/O port, an 8-bit tri-state data bus, or as a multiplexed address/data bus.

Serial Input/Output Channel — UART

The devices provide a full duplex serial I/O channel with programmable bit rates covering all standard baud rates from 50 to 125K bits/sec including the SMPTE 422 standard at 38.4K bits/sec. Character lengths of 5 to 8 bits, with or without parity are programmable. A full complement of flags provides for Receiver Wake Up; Receiver Buffer Full; Receiver Error Conditions detecting Framing, Parity, and Overrun errors; Transmitter End of Transmission and Transmitter Buffer Empty. In addition, a synchronous shift register mode to 250 KC at 2 MHz is available.

Wake-Up Feature

In a multi-distributed microcomputer application, a destination address is usually included at the beginning of the message. The Wake-Up Feature allows non-selected CPUs to ignore the remainder of the message until the beginning of the next message by setting the Wake-Up bit.

Counter/Latch Logic

The devices contain two 16-bit counters (Counter A and Counter B) and three 16-bit latches associated with the counters. Counter A has one 16-bit latch and Counter B has two 16-bit latches. Each counter can be independently programmed to operate in one of four modes:

Counter A	Counter B
• Pulse width measurement	• Retriggerable Interval Counter
• Pulse Generation	• Assymmetrical Pulse Generation
• Interval Timer	• Interval Timer
• Event Counter	• Event Counter

Mode Control Register (MCR)

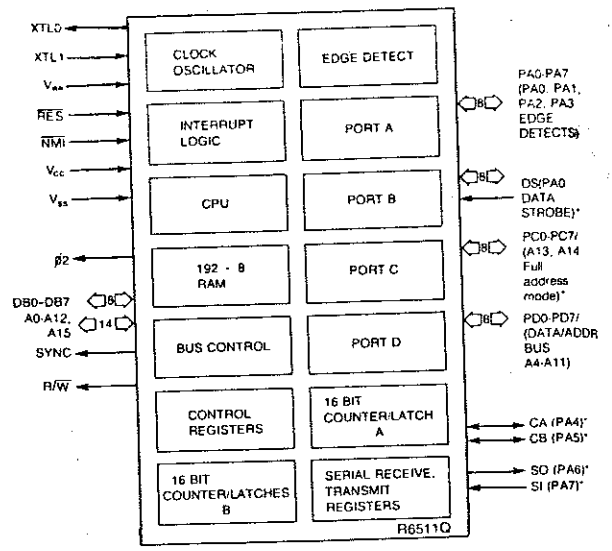
The Mode Control Register contains control bits for the multifunction I/O ports and mode select bits for Counter A and Counter B.

Ports C and D Operation Modes

There are four operating modes available in ports C and D, software programmable via the Mode Control Register. The full address mode allows access to a full 65K bytes of external storage. In this mode PC6 and PC7 are automaticallyu used for A13 and A14. In the Input/Output mode the four ports are all used for I/O. In the abbreviated and multiplexed modes some port pins set up for addressing 64 or 16,384 bytes of external memory.

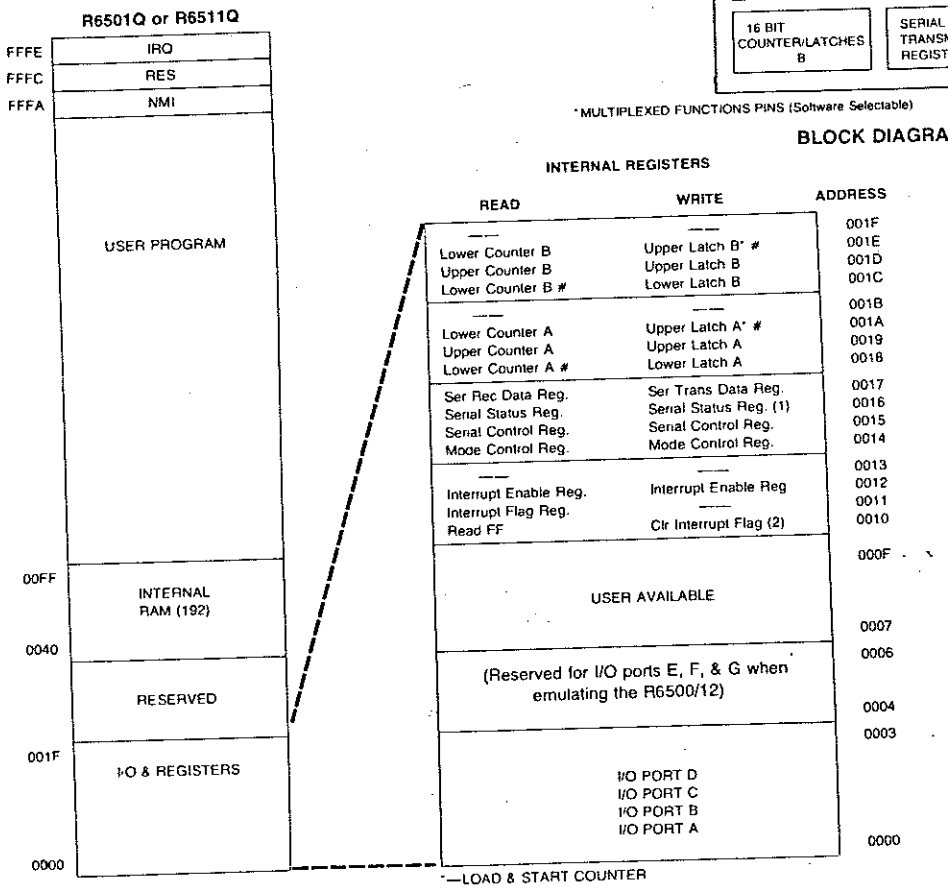
Interrupt Flag Register (IFR) and Interrupt Enable Register (IER)

The devices include an Interrupt Flat Register and an Interrupt Enable Register which flags and controls I/O and counter status.



*MULTIPLEXED FUNCTIONS PINS (Software Selectable)

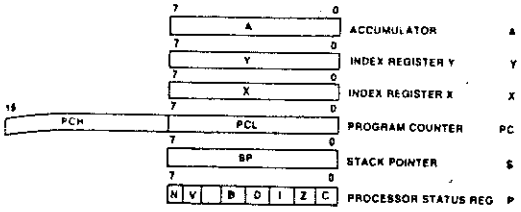
BLOCK DIAGRAM



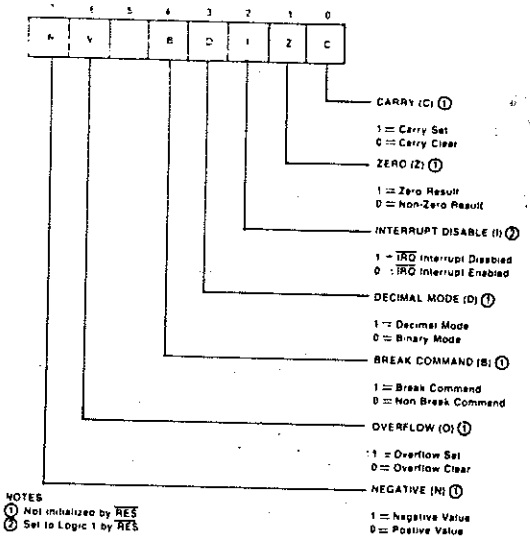
*—LOAD & START COUNTER
CLEAR FLAG
(1) BITS 4 & 5 ONLY
(2) BITS 0-3 ONLY

MEMORY MAP

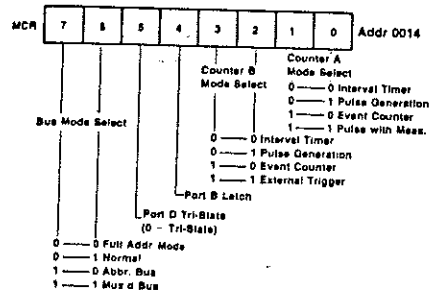
KEY REGISTER SUMMARY



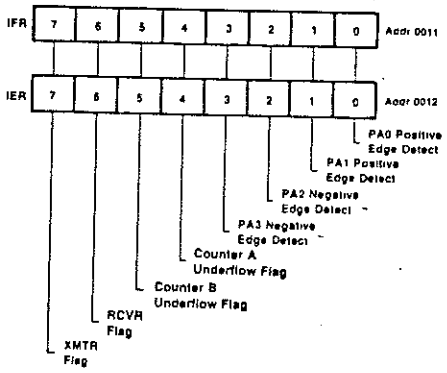
CPU Registers



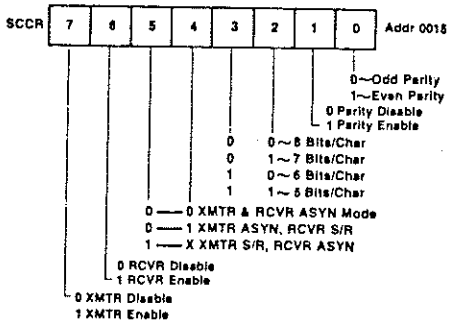
Processor Status Register



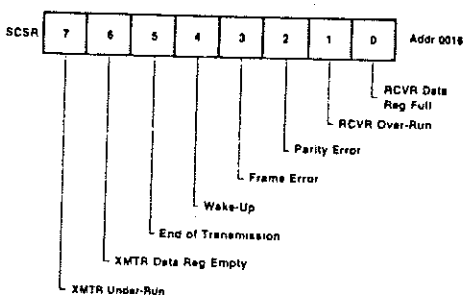
Mode Control Register



Interrupt Enable and Flag Registers



Serial Communications Control Register



Serial Communications Status Register

ABSOLUTE MAXIMUM RATINGS*

Parameter	Symbol	Value	Unit
Supply Voltage	V_{CC} & V_{RR}	−0.3 to +7.0	Vdc
Input Voltage	V_{IN}	−0.3 to +7.0	Vdc
Operating Temperature Commercial	T	0 to +70	°C
Storage Temperature Range	T_{STG}	−55 to +150	°C

*NOTE: Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in other sections of this document is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

(V_{CC} = 5V ±5% V_{SS} = 0)

Parameter	Symbol	Min	Typ	Max	Unit
Power Dissipation (Outputs High) Commercial at 25°C	P_D	—	—	1200	mW
RAM Standby Voltage (Retention Mode)	V_{RR}	3.0	—	V_{CC}	Vdc
RAM Standby Current (Retention Mode) Commercial at 25°C	I_{RR}	—	4	—	mAdc
Input High Voltage Except XTLI	V_{IH}	+2.0	—	V_{CC}	Vdc
Input High Voltage (XTLI)	V_{IH}	+4.0	—	V_{CC}	Vdc
Input Low Voltage	V_{IL}	−0.3	—	+0.8	Vdc
Input Leakage Current ($\overline{RES}$, $\overline{NMI}$) V_{IN} = 0 to 5.0 Vdc	I_{IN}	—	—	±10	μAdc
Input Low Current (V_{IL} = 0.4 Vdc)	I_{IL}	—	−1.0	−1.6	mAdc
Output High Voltage Except XTLO (I_{LOAD} = −100 μAdc)	V_{OH}	+2.4	—	V_{CC}	Vdc
Output Low Voltage (I_{LOAD} = 1.5 mAdc)	V_{OL}	—	—	+0.4	Vdc
Input Capacitance (V_{IN} = 0, T_A = 25°C, f = 1.0 MHz) XTL1, XTLO All Others	C_{IN}	— —	— —	50 10	pF
I/O Port Pull-Up Resistance PA0–PA7, PB0–PB7, PC0–PC7 R6501Q only	R_L	3.0	6.0	11.5	KΩ

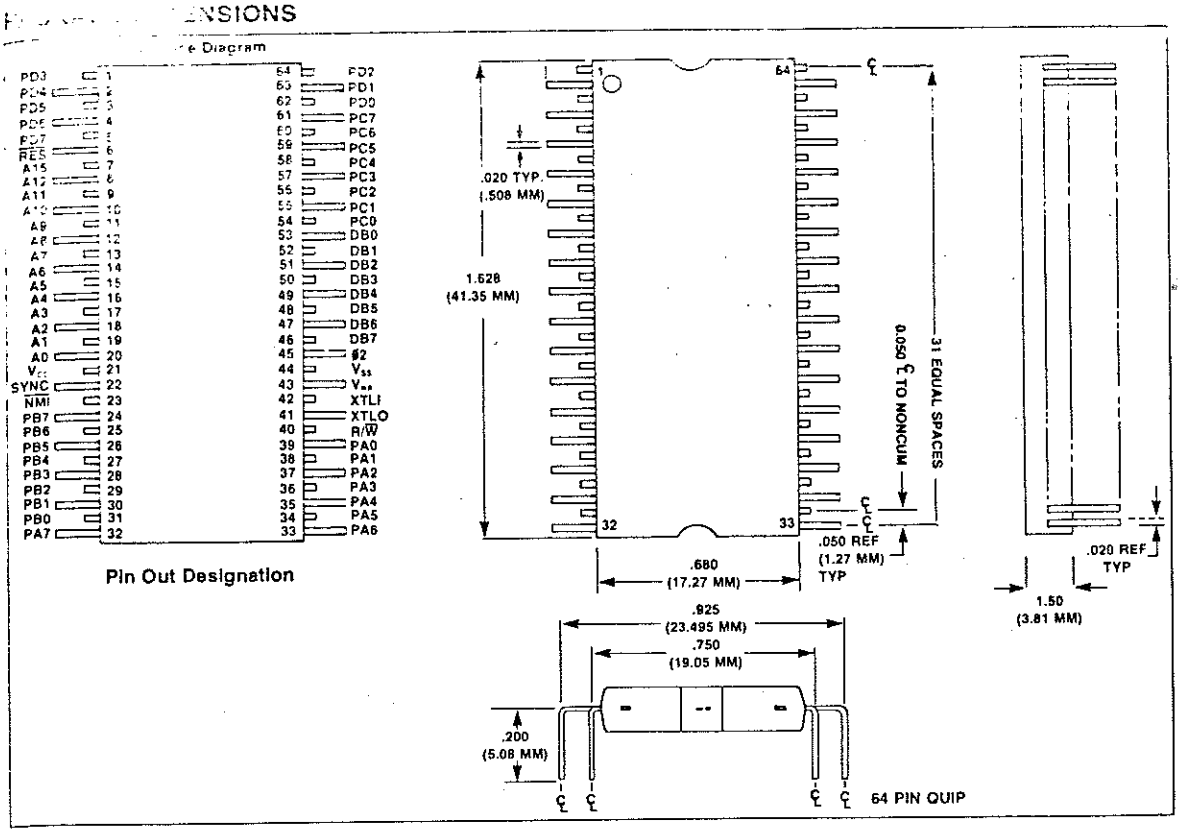
Note: Negative sign indicates outward current flow, positive indicates inward flow.

AC CHARACTERISTICS

(V_{CC} = 5V ±6% V_{SS} = 0

Parameter	Symbol	1 MHz		2 MHz		Unit
		Min	Max	Min	Max	
XTLI Input Clock Cycle Time	T_{CYC}	1.0	10.0	0.500	10.0	μsec
Internal Write to Peripheral Data Valid (TTL)	T_{POW}	1.0		0.5		μsec
Peripheral Data Setup Time	T_{PDSU}	500		500		nsec
Count and Edge Detect Pulse Width	T_{PW}	1.0		0.5		μsec

...ENSIONS


e Diagram
PD3 1
PD4 2
PD5 3
PD6 4
PD7 5
RES 6
A15 7
A12 8
A11 9
A10 10
A9 11
A8 12
A7 13
A6 14
A5 15
A4 16
A3 17
A2 18
A1 19
A0 20
V_{cc} 21
SYNC 22
NMI 23
PB7 24
PB6 25
PB5 26
PB4 27
PB3 28
PB2 29
PB1 30
PB0 31
PA7 32
64 PD2
63 PD1
62 PD0
61 PC7
60 PC6
59 PC5
58 PC4
57 PC3
56 PC2
55 PC1
54 PC0
53 DB0
52 DB1
51 DB2
50 DB3
49 DB4
48 DB5
47 DB6
46 DB7
45 Ø2
44 V_{ss}
43 V_{ss}
42 XTLI
41 XTLO
40 R/W
39 PA0
38 PA1
37 PA2
36 PA3
35 PA4
34 PA5
33 PA6
Pin Out Designation
.020 TYP.
(.508 MM)
1.628
(41.35 MM)
31 EQUAL SPACES
0.050 ℄ TO NONCUM
.050 REF
(1.27 MM)
TYP
.680
(17.27 MM)
.925
(23.495 MM)
.750
(19.05 MM)
.200
(5.08 MM)
.020 REF
TYP
1.50
(3.81 MM)
64 PIN QUIP

2

APPENDIX D

Programlistor.

D1. FILECOMM.4TH
SCODE.4TH
FSCODE.4TH

D2. COMIC2.MOD
FILEHANDLER.MOD
FILEHANDLER.DEF

D3. READFILE.MOD
DUMPFILE.MOD
DELFILE.MOD
SHOWDIR.MOD
CLEARDIR.MOD

D4. SYMB.PAS

D5. FIHAND1.ASM
FIHAND2.ASM

APPENDIX D1.

D1. FILECOMM.4TH
SCODE.4TH
FSCODE.4TH

```
: CREATEFILE CR 04 EMIT 31 EMIT ;
: OPENFILE CR 04 EMIT 32 EMIT ;
: CLOSEFILE CR 04 EMIT 33 EMIT ;
: READFILE CR 04 EMIT 34 EMIT ;
: WRITEFILE CR 04 EMIT 35 EMIT ;
: DOWNFETCH CR 04 EMIT 36 EMIT SCODE ;
: SECUREFETCH CR 04 EMIT 37 EMIT FSCODE ;
: DELFILE CR 04 EMIT 38 EMIT ;
: SHOWDIR CR 04 EMIT 39 EMIT ;

: STX 2 EMIT ;
: ETX 3 EMIT ;
```

```
VARIABLE UT

: SCODE
  \ A modified version of the original SOURCECODE.
  [ 20 C, ' [ , ] UT OFF
  BEGIN
    >IN OFF #TIB OFF TIB 50 BOUNDS XON# EMIT
    ?DO KEY DUP 0D = IF DROP LEAVE THEN DUP 04 = IF UT ON LEAVE THEN
    I C! 1 #TIB +! LOOP UT @ IF DROP EXIT THEN
    XOF# EMIT 0 TIB #TIB @ + C!
    INTERPRET
  AGAIN ;
```

```
VARIABLE NUM
VARIABLE LOP
VARIABLE CSUMI
VARIABLE CSUME

: ACK 06 EMIT ;
: NAK 15 EMIT ;

: EXPCT    \ EXPECT utan eko.
  SPAN OFF 0
  ?DO KEY OVER SPAN @ + C! 1 SPAN +! LOOP
  0 SWAP SPAN @ + C! ;

: CONV TIB SPAN VAL 2DROP ; \ Konverterar ASCII-tecken i TIB till en byte.

: SUMTIB  \ Summerar innehaallet i TIB och laegger resultatet i CSUMI.
  LOP OFF
  NUM @ 2 / 0 ?DO TIB LOP @ + @ SPLIT SWAP + CSUMI @ +
  FF MOD CSUMI ! 2 LOP +! LOOP
  NUM @ 2 MOD DUP 1 =
  IF DROP TIB LOP @ + @ SPLIT SWAP CSUMI +! DROP CSUMI @ FF MOD CSUMI !
  ELSE DROP
  THEN ;


: CODE
  \ A modified version of the original SOURCECODE.
  [ 20 C, ' [ , ] UT OFF
  BEGIN
    >IN OFF #TIB OFF CSUMI OFF 2 NUM ! XON# EMIT
    BEGIN KEY 01 = UNTIL
    TIB 2 EXPCT SUMTIB CONV DUP NUM !
    0 <> IF
      TIB NUM @ EXPCT TEMP 2 EXPCT TEMP SPAN VAL 2DROP CSUME !
      SUMTIB CSUMI @ CSUME @
      = IF
        ACK XOF# EMIT NUM @ #TIB ! INTERPRET
      ELSE
        NAK
      THEN
    ELSE
      TEMP 2 EXPCT ACK XOF# EMIT EXIT
    THEN
  AGAIN ;
```

APPENDIX D2.

D2. COMIC2.MOD
FILEHANDLER.MOD
FILEHANDLER.DEF

```
MODULE Comic ;

  FROM SYSTEM IMPORT OUTBYTE ;
  FROM FileHandler IMPORT CreateFile, ReadFileName, CloseFile, OpenFile,
                          ReadFile, WriteFile, DownFetch, SecureFetch,
                          DelFile, ShowDir, Execute, ForgetWord,
                          FileError, AFileIsOpen, com1char ;

  IMPORT RS232Int, InOut, Keyboard, Display ;

  CONST  esc     = 33C ;      (* ASCII characters in octal base *)
         eoln    = 36C ;
         cr      = 15C ;
         ctrlD   = 04C ;
         Create  = 61C ;
         Open    = 62C ;
         Close   = 63C ;
         Read    = 64C ;
         Write   = 65C ;
         Down    = 66C ;
         Secure  = 67C ;
         Delete  = 70C ;
         Show    = 71C ;
         colon   = 72C ;
         semicolon = 73C ;
         Exec      = 101C ;
         ForgW     = 102C ;


  VAR  keychar      : CHAR ;
       ok,more      : BOOLEAN ;
       i            : INTEGER ;
       KeyPr        : BOOLEAN ;



PROCEDURE ReadCommand ;
BEGIN
  REPEAT
    RS232Int.BusyRead( com1char, ok ) ;
    KeyPr := Keyboard.KeyPressed() ;
  UNTIL (( ok ) OR ( KeyPr )) ;
  IF ( KeyPr ) THEN
    Keyboard.Read( keychar ) ;
    RS232Int.Write( keychar ) ;
    RS232Int.Read( com1char ) ;
    KeyPr := FALSE ;
  END ;
END ReadCommand ;



BEGIN  (* Module Comic *)
  keychar := CHR( 0 ) ;
  RS232Int.Init( 4800, 2, FALSE, FALSE, 7, ok ) ;
  IF ( ok ) THEN
    InOut.WriteString( 'Initialization of COM1 completed.' ) ;
    InOut.WriteLn ;
    RS232Int.StartReading ;
    RS232Int.Write( cr ) ;
    AFileIsOpen := FALSE ;
    REPEAT
      i := 0 ;
      RS232Int.BusyRead( com1char, ok ) ;
      WHILE ( ok AND ( i < 80 ) AND ( com1char <> ctrlD )) DO
```

```
      Display.Write( comichar ) ;
      RS232Int.BusyRead( comichar, ok ) ;
      INC( i );
    END ;
    IF ( comichar = ctrlD ) THEN
      ReadCommand ;
      CASE comichar OF
          Create : IF AFileIsOpen THEN
                     FileError( 10 ) ;
                   ELSE
                     ReadFileName ;
                     CreateFile ;
                     AFileIsOpen := TRUE ;
                   END
        | Open   : IF AFileIsOpen THEN
                     FileError( 10 ) ;
                   ELSE
                     ReadFileName ;
                     AFileIsOpen := TRUE ;
                     OpenFile ;
                   END
        | Close  : CloseFile ;
                   AFileIsOpen := FALSE
        | Read   : IF AFileIsOpen THEN
                     ReadFile ;
                   ELSE
                     FileError( 11 ) ;
                   END
        | Write  : IF AFileIsOpen THEN
                     WriteFile ;
                   ELSE
                     FileError( 11 ) ;
                   END
        | Down   : DownFetch ;
                   AFileIsOpen := FALSE
        | Secure : SecureFetch ;
                   AFileIsOpen := FALSE
        | Delete : IF AFileIsOpen THEN
                     FileError( 8 ) ;
                   ELSE
                     ReadFileName ;
                     DelFile ;
                   END
        | Show   : ShowDir
        | Exec   : Execute
        | ForgW  : ForgetWord
      ELSE
        InOut.WriteString( ' Not a proper command. ' ) ;
      END ;
    ELSIF ( ok AND ( i = 80 ) AND ( comichar <> ctrlD )) THEN
      OUTBYTE( 1020, 10 ) ;
      Display.Write( comichar ) ;
    ELSE
      OUTBYTE( 1020, 11 ) ;
    END ;
    KeyPr := Keyboard.KeyPressed() ;
    IF ( KeyPr ) THEN ;
      Keyboard.Read( keychar ) ;
      IF ( keychar <> esc ) THEN
        IF ( keychar = eoln ) THEN
          keychar := cr ;
        END ;
        RS232Int.Write( keychar ) ;
      END ;
    END ;
  UNTIL ( keychar = esc ) ;
```

```
    RS232Int.StopReading ;
  ELSE
    InOut.WriteString( 'Initialization of COM1 failed.' ) ;
    InOut.WriteLn ;
  END ;
END Comic.
```

```
IMPLEMENTATION MODULE FileHandler ;

IMPORT InOut, RS232Int, Display, Delay, Keyboard ;
FROM FileSystem IMPORT Create, Rename, Close, File, Response, Flag, FlagSet,
                       Lookup, SetOpen, ReadChar, SetRead, SetWrite,
                       WriteChar, Reset, Length, SetPos, Delete ;
FROM Exec IMPORT DosCommand ;

CONST  cr   = 15C ;
       MaxNoOfChar = 31 ;

TYPE   FileNameString = ARRAY [0..MaxNoOfChar] OF CHAR ;

VAR    FileName      : FileNameString ;
       Fi            : File ;
       TerminalInput : BOOLEAN ;



PROCEDURE FileError( KindOf : CARDINAL ) ;
BEGIN
  InOut.WriteString( '===> Error in fileoperation. ' ) ;
  InOut.WriteLn ;
  CASE KindOf OF
      1 : InOut.WriteString( '     No file created. ' ) ;
    | 2 : InOut.WriteString( '     The file exist but could not be opened. ' )
    | 3 : InOut.WriteString( '     No file to close. ' ) ;
    | 4 : InOut.WriteString( '     Error in reading file. ' ) ;
    | 5 : InOut.WriteString( '     Error in set read. ' ) ;
    | 6 : InOut.WriteString( '     Error in writing file. ' ) ;
    | 7 : InOut.WriteString( '     Error in resetting the file. ' ) ;
    | 8 : InOut.WriteString( '     To delete a file all files must be closed. '
    | 9 : InOut.WriteString( '     No such file to delete. ' ) ;
    | 10 : InOut.WriteString( '     Only one file could be open at the time. '
    | 11 : InOut.WriteString( '     No File is open. ' ) ;
  ELSE
  END ;
END FileError ;



PROCEDURE WriteFileName ;
CONST blank = 40C ;
VAR   j     : INTEGER ;
      ch    : CHAR ;
BEGIN
  j := 0 ;
  REPEAT
    ch := FileName[ j ] ;
    InOut.Write( ch ) ;
    INC( j ) ;
  UNTIL (( ch = blank ) OR ( j = 31 )) ;
END WriteFileName ;



PROCEDURE ReadFileName ;
(*
 "ReadFileName" reads the name of the file to operate on, either from
  the keyboard or from the "MF10".
*)
CONST blank = 40C ;
      eoln  = 36C ;
      nul   = 0C ;
      stx   = 2C ;
      etx   = 3C ;
```

```
      BackSpace = 10C ;
      Delete    = 177C ;

VAR   k, j      : CARDINAL ;
      Prompt,
      KeyPr, ok : BOOLEAN ;
      keychar   : CHAR ;

BEGIN
  InOut.WriteString( 'Filename : ' ) ;
  REPEAT
    RS232Int.BusyRead( comichar, ok ) ;
    KeyPr := Keyboard.KeyPressed() ;
  UNTIL (( comichar = stx ) OR ( KeyPr )) ;
  k := 0 ;
  IF ( comichar = stx ) THEN
    TerminalInput := FALSE ;
    RS232Int.Read( comichar ) ;
    WHILE (( comichar <> etx ) AND ( k < MaxNoOfChar )) DO
      FileName[ k ] := comichar ;
      Display.Write( comichar ) ;
      INC( k ) ;
      RS232Int.Read( comichar ) ;
    END ;
  ELSE
    TerminalInput := TRUE ;
    InOut.Read( keychar ) ;
    WHILE (( keychar <> eoln ) AND ( k < MaxNoOfChar )) DO
      FileName[ k ] := keychar ;
      Display.Write( keychar ) ;
      IF ((( keychar = BackSpace ) OR ( keychar = Delete )) AND ( k > 0 )) THEN
        DEC( k ) ;
      ELSE
        INC( k ) ;
      END ;
      InOut.Read( keychar ) ;
    END ;
  END ;
  InOut.WriteLn ;
  IF ( k = MaxNoOfChar ) THEN
    InOut.WriteString( ' Error in filename, to many letters.' ) ;
    InOut.WriteLn ;
  ELSE
    FOR j := k TO MaxNoOfChar DO
      FileName[ j ] := blank ;
    END ;
  END ;
END ReadFileName ;




PROCEDURE CreateFile ;
(*
  Search for the file "FileName". If the file exists it will not
  be created once again. If it doesn't exist this procedure will
  try to create it.
*)

BEGIN
  Lookup( Fi, FileName, FALSE ) ;
  IF ( Fi.res = done ) THEN
    WriteFileName ;
    InOut.WriteString( '  does already exist. ' ) ;
    InOut.WriteLn ;
```

```
  ELSE
    Lookup( Fi, FileName, TRUE ) ;
    IF ( Fi.res = done ) THEN
      InOut.WriteString( 'Created file : ' ) ;
      WriteFileName ;
      InOut.WriteLn ;
    ELSE
      FileError( 1 ) ;
    END ;
  END ;
  IF ( TerminalInput ) THEN
    RS232Int.Write( cr ) ;
  END ;
END CreateFile ;



PROCEDURE OpenFile ;
(*
  This procedure looks for a file, specified with "FileName", and
  if it succeds it will open the file. If not succesful in finding
  the searched file it asks wether to create a new file or not. If
  the answer is Y (Yes), the procedure will create a new file and
  leaves it open. If any fileoperation failes no file is opened.
*)
CONST  Y = 131C ;
       y = 171C ;

VAR    Choise : CHAR ;
       Exec   : BOOLEAN ;

BEGIN
  Lookup( Fi, FileName, FALSE ) ;
  IF ( Fi.res = notdone ) THEN
    WriteFileName ;
    InOut.WriteString( '  does not exist.' ) ;
    InOut.WriteLn ;
    InOut.WriteString( 'Do you want to create it, ( Y/N ) ? ' ) ;
    InOut.Read( Choise ) ;
    InOut.Write( Choise ) ;
    InOut.WriteLn ;
    IF (( Choise = y ) OR ( Choise = Y )) THEN
      Lookup( Fi, FileName, TRUE ) ;
      IF ( Fi.res = done ) THEN
        Exec := TRUE ;
        AFileIsOpen := TRUE ;
        InOut.WriteString( 'Created file : ' ) ;
        WriteFileName ;
        InOut.WriteLn ;
      ELSE
        Exec := FALSE ;
        AFileIsOpen := FALSE ;
        FileError( 1 ) ;
      END ;
    ELSE
      AFileIsOpen := FALSE ;
    END ;
    Exec := FALSE ;
  ELSIF ( Fi.res = done ) THEN
    Exec := TRUE ;
  END ;
  IF Exec THEN
    SetOpen( Fi ) ;
    IF ( Fi.res = done ) THEN
      WriteFileName ;
      InOut.WriteString( '  is open.' ) ;
```

```
    ELSE
      FileError( 2 ) ;
    END ;
  END ;
  InOut.WriteLn ;
  IF ( TerminalInput ) THEN
    RS232Int.Write( cr ) ;
  END ;
END OpenFile ;



PROCEDURE CloseFile ;
(*
  Only one file is allowed to be open at the same time. "CloseFile"
  must close that file before a new file could be opened.
*)
BEGIN
  Close( Fi ) ;
  IF ( Fi.res = done ) THEN
    WriteFileName ;
    InOut.WriteString( '  is closed.' ) ;
    InOut.WriteLn ;
  ELSE
    FileError( 3 ) ;
  END ;
END CloseFile ;



PROCEDURE ReadFile ;
(*
 "ReadFile" reads from a file on the diskstation and sends the content
  down to the MF10 via RS232-serialinterface. The file to be read from
  must be an open file before entering "ReadFile".
*)
VAR ch : CHAR ;
BEGIN
  Reset( Fi ) ;
  SetRead( Fi ) ;
  IF ( Fi.res = done ) THEN
    WriteFileName ;
    InOut.WriteLn ;
    InOut.WriteString( '-------------------------------------------------------------
    InOut.WriteLn ;
    WHILE (( NOT Fi.eof ) AND ( Fi.res = done )) DO
      ReadChar( Fi, ch ) ;
      InOut.Write( ch ) ;
    END ;
    IF ( Fi.res = done ) THEN
      InOut.WriteLn ;
      InOut.WriteString( '---------------------------------- E N D ----------------
      InOut.WriteLn ;
    ELSE
      FileError( 4 ) ;
    END ;
  ELSE
    FileError( 5 ) ;
  END ;
END ReadFile ;



PROCEDURE WriteFile ;
(*
```

```
  "WriteFile" reads a character from the RS232-serialinterface,
   connected to MF10, and writes the character to an open file
   wich is set in write-mode.
*)
CONST esc   = 33C ;
      nul   = 0C ;
      eoln = 36C ;
      stx   = 2C ;
      etx   = 3C ;
      NoInRow = 80 ;
      BackSpace = 10C ;
      Delete    = 177C ;

VAR   cur, keychar     : CHAR ;
      OverWrite,
      KeyPr, Stop, ok : BOOLEAN ;
      k, i             : CARDINAL ;
      OneRow           : ARRAY [0..NoInRow] OF CHAR ;

BEGIN
  nOut.WriteLn ;
  nOut.WriteString( 'Overwrite or Continue the file ? ( O/C ) : ' ) ;
  REPEAT
    RS232Int.BusyRead( comichar, ok ) ;
    KeyPr := Keyboard.KeyPressed() ;
  NTIL (( comichar = stx ) OR ( KeyPr )) ;
  F ( KeyPr ) THEN
    InOut.Read( keychar ) ;
    InOut.Write( keychar ) ;
    InOut.Read( cur ) ;
    InOut.WriteLn ;
    WHILE (( keychar <> 'O' ) AND ( keychar <> 'C' )) DO
      InOut.WriteString( 'Must type in "O" or "C" in uppercase, try again : ' )
      InOut.Read( keychar ) ;
      InOut.Write( keychar ) ;
      InOut.Read( cur ) ;
      InOut.WriteLn ;
    END ;
    OverWrite := keychar = 'O' ;
    InOut.WriteLn ;
    WriteFileName ;
    InOut.WriteString( '              Press escape when finished.' ) ;
  ( InOut.WriteLn ;
    InOut.WriteString( '-------------------------------------------------------------
    InOut.WriteLn ;
    IF OverWrite THEN
      Reset( Fi ) ;
  ( ELSE
      Reset( Fi ) ;
      SetRead( Fi ) ;
      ReadChar( Fi, comichar ) ;
      WHILE (( NOT Fi.eof ) AND ( Fi.res = done )) DO
        InOut.Write( comichar ) ;
        ReadChar( Fi, comichar ) ;
      END ;
    END ;
    SetWrite( Fi ) ;
    keychar := nul ;
    REPEAT
      k := 0 ;
      InOut.Read( keychar ) ;
      WHILE (keychar <> cr) AND (keychar <> esc) AND (k < NoInRow) DO
        OneRow[ k ] := keychar ;
        InOut.Write( keychar ) ;
        IF (((keychar = BackSpace) OR (keychar = Delete)) AND (k > 0)) THEN
          DEC( k ) ;
```

```
        ELSE
          INC( k ) ;
        END ;
        InOut.Read( keychar ) ;
      END ;
      FOR i := 0 TO (k - 1) DO
        WriteChar( Fi, OneRow[ i ]) ;
      END ;
      WriteChar( Fi, eoln ) ;
    UNTIL (( Fi.res <> done ) OR ( keychar = esc )) ;
    IF ( Fi.res <> done ) THEN
      FileError( 6 ) ;
    ELSE
      InOut.WriteLn ;
      InOut.WriteString( '---------------------------------- E N D ----------------
    END ;
    InOut.WriteLn ;
    RS232Int.Write( cr ) ;
  ELSE   (* Caracters are coming from MF10. *)
    SetWrite( Fi ) ;
    RS232Int.Read( com1char ) ;
    Display.Write( com1char ) ;
    Stop := FALSE ;
    IF ( com1char = 117C ) THEN          (* 117C = 'O' *)
      Reset( Fi ) ;
    ELSIF ( com1char = 103C ) THEN       (* 103C = 'C' *)
      Reset( Fi ) ;
      SetRead( Fi ) ;
      WHILE (( NOT Fi.eof ) AND ( Fi.res = done )) DO
        ReadChar( Fi, com1char ) ;
      END ;
      SetWrite( Fi ) ;
    ELSE
      FileError( 0 ) ;
      Stop := TRUE ;
    END ;
    IF ( NOT Stop ) THEN
      RS232Int.Read( com1char ) ;
      WHILE ( com1char <> etx ) DO
        WriteChar( Fi, com1char ) ;
        RS232Int.Read( com1char ) ;
      END ;
      WriteChar( Fi, eoln ) ;
    ELSE
      InOut.WriteString( '    Error in writing-choise. ' ) ;
    END ;
  END ;
E WriteFile ;



PROCEDURE DownFetch ;
(*
 "DownFetch" sends down a file to the "MF10". The file to send down must
  be an open file otherwise FileError will occur. To use "DownFetch", the
  file must contain either compileable deifinitions or already existing
  words to match a definition in "WORDS" or numbers to push on the stack,
  any other contents in the file will cause an error in the "MF10". After
  a successful transmission the file will be closed.
*)
CONST  xon  = 21C ;
       bell = 7C ;
       eoln = 36C ;
       eof  = 0C ;
       eot  = 04C ;
```

```
VAR     filechar        : CHAR ;
        index, rownum   : INTEGER ;
        end, newln      : BOOLEAN ;
        text            : ARRAY [0..9] OF CHAR ;
        fail, ok        : BOOLEAN ;

BEGIN
  Reset( Fi ) ;
  IF ( Fi.res = done ) THEN
    rownum := 1 ;
    end := FALSE ;
    fail := FALSE ;
    REPEAT
      newln := FALSE ;
      REPEAT
        RS232Int.Read( com1char ) ;
        IF ( com1char = bell ) THEN
          fail := TRUE ;
        ELSE
          newln := ( com1char = xon ) ;
        END ;
        Display.Write( com1char ) ;
      UNTIL ( fail OR newln ) ;
      IF ( NOT fail ) THEN
        ReadChar( Fi, filechar ) ;
        WHILE (( filechar <> eoln ) AND ( filechar <> eof )) DO
          RS232Int.Write( filechar ) ;
          ReadChar( Fi, filechar ) ;
        END ;
        RS232Int.Write( cr ) ;
        InOut.WriteInt( rownum, 1 ) ;
        INC( rownum ) ;
        end := ( filechar = eof ) ;
      END ;
    UNTIL ( end OR fail ) ;
    Close( Fi ) ;
    RS232Int.Write( eot ) ;
  ELSE
    FileError( 7 ) ;
    InOut.WriteLn ;
  END ;
END DownFetch ;



PROCEDURE SecureFetch ;
(*
  Same as "DownFetch". The differens is the way of transfering data.
  "DownFetch" transfers single characters one by one contrary to
  "SecureFetch" wich transfers data row by row in frames. If there's
  noise on the line during the transfer and the frame is damaged in some
  way the sender will recive a NAK(Negative AcKowledge) and the frame will
  be retransmitted until an ACK (positiv ACKnowledge) is recived.
*)
CONST eof   = 0C ;
      soh   = 01C ;
      eot   = 04C ;
      ack   = 06C ;
      bell  = 07C ;
      xon   = 21C ;
      nak   = 25C ;
      eoln  = 36C ;
      NoInRow = 80 ;

 TYPE  FrameBufferType = ARRAY [0..NoInRow] OF CHAR ;
```

```
VAR   FileChar, comich                : CHAR ;
      index, CheckSum, RowNum, i      : CARDINAL ;
      Check1, Check2, NoCh1, NoCh2    : CHAR ;
      End, Fail, NewLine              : BOOLEAN ;
      NoOfNAK                         : INTEGER ;
      FrameBuffer                     : FrameBufferType ;

PROCEDURE ASCIITransform(K:CARDINAL; VAR A,B:CHAR) ;
(* Transforns one byte K into two ASCII-characters A and B. *)
VAR   Temp1, Temp2 : CARDINAL ;
BEGIN
  Temp1 := (K DIV 16) ;
  CASE Temp1 OF
    0..9 :    Temp1 := Temp1 + 48 ;   |
    10..15 : Temp1 := Temp1 + 55 ;
  ELSE
    InOut.WriteString('Rotten transforming, eih !') ;
    InOut.WriteLn ;
  END ;
  := CHR( Temp1 ) ;
emp2 := (K MOD 16) ;
  CASE Temp2 OF
    0..9 :    Temp2 := Temp2 + 48 ;   |
    10..15 : Temp2 := Temp2 + 55 ;
LSE
    InOut.WriteString('Oh no, wrong again !') ;
    InOut.WriteLn ;
  END ;
  B := CHR( Temp2 ) ;
END ASCIITransform ;

BEGIN (* SecureFetch *)
  Reset( Fi ) ;
  IF (Fi.res = done) THEN
    RowNum := 1 ;
    End := FALSE ;
    Fail := FALSE ;
    REPEAT
      NewLine := FALSE ;
      REPEAT
        RS232Int.Read(comichar) ;
        IF (comichar = bell) THEN
          Fail := TRUE ;
        ELSE
          NewLine := comichar = xon ;
        END ;
        InOut.Write(comichar) ;
      UNTIL (Fail OR NewLine) ;
      IF (NOT Fail) THEN
        index := 0 ;
        CheckSum := 0 ;
        REPEAT
          ReadChar(Fi, FileChar) ;
          WHILE ((FileChar <> eoln) AND (FileChar <> eof)) DO
            FrameBuffer[index] := FileChar ;
            INC( index ) ;
            CheckSum := (CheckSum + ORD(FileChar)) MOD 255 ;
            ReadChar(Fi, FileChar) ;
          END ;
          End := FileChar = eof ;
          INC( RowNum ) ;
        UNTIL ((index > 0) OR End) ;
        InOut.WriteCard( RowNum, 1 ) ;

        ASCIITransform(index, NoCh1, NoCh2) ;
```

```
        CheckSum := (CheckSum + ORD(NoCh1)) MOD 255 ;
        CheckSum := (CheckSum + ORD(NoCh2)) MOD 255 ;
        ASCIITransform(CheckSum, Check1, Check2) ;
        NoOfNAK := 0 ;
        REPEAT
          IF (NOT End) THEN                (* Send dataframe. *)
            RS232Int.Write(soh) ;          (* soh = Start OF Header. *)
            RS232Int.Write(NoCh1) ;
            RS232Int.Write(NoCh2) ;
            FOR i := 0 TO (index - 1) DO
              RS232Int.Write(FrameBuffer[i]) ;
            END ;
            RS232Int.Write(Check1) ;
            RS232Int.Write(Check2) ;
          ELSE                                (* Send endframe. *)
            RS232Int.Write(soh) ;
            FOR i := 0 TO 3 DO
              RS232Int.Write('0') ;
            END ;
          END ;
          RS232Int.Read(comichar) ;
          IF (comichar = nak) THEN
            RS232Int.Read(comichar) ;
            InOut.Write(comichar) ;
            InOut.WriteString('NAK recived') ;
            INC( NoOfNAK ) ;
            Fail := NoOfNAK = 10 ;
          END ;
        UNTIL ((comichar = ack) OR Fail) ;
        RS232Int.Read(comichar) ;
        InOut.Write(comichar) ;
      END ;
    UNTIL (End OR Fail) ;
    Close( Fi ) ;
    InOut.WriteLn ;
  ELSE
    FileError( 7 ) ;
    InOut.WriteLn ;
  END ;
END SecureFetch ;


PROCEDURE DelFile ;
(*
  "DelFile" deletes a closed file specified by "FileName".
*)
BEGIN
  Delete( FileName, Fi ) ;
  IF ( Fi.res = done ) THEN
    InOut.WriteString( 'Deleted file : ' ) ;
    WriteFileName ;
  ELSE
    FileError( 9 ) ;
  END ;
  IF ( TerminalInput ) THEN
    RS232Int.Write( cr ) ;
  END ;
  InOut.WriteLn ;
END DelFile ;



PROCEDURE ShowDir ;
(*
  Displays the current directory on the screen. Options is regular MS-DOS
```

```
 options e.g /P, /W, wildcharacters etc.
)
ONST eoln = 36C ;
     BackSpace = 10C ;
     Delete    = 177C ;
     blank     = 40C ;

AR   Ok : BOOLEAN ;
     CommandParameters : ARRAY [0..MaxNoOfChar] OF CHAR ;
     KeyChar : CHAR ;
     k, j : CARDINAL ;

EGIN
  InOut.WriteString( 'Options : ' ) ;
  k := 0 ;
  InOut.Read( KeyChar ) ;
  WHILE (( KeyChar <> eoln ) AND ( k < MaxNoOfChar )) DO
    CommandParameters[ k ] := KeyChar ;
    Display.Write( KeyChar ) ;
    IF ((( KeyChar = BackSpace ) OR ( KeyChar = Delete )) AND ( k > 0 )) THEN
      DEC( k ) ;
    ELSE
      INC( k ) ;
    END ;
    InOut.Read( KeyChar ) ;
  ND ;
  nOut.WriteLn ;
  IF ( k = MaxNoOfChar ) THEN
    InOut.WriteString( ' Options, to many letters. ' ) ;
    InOut.WriteLn ;
  ELSIF ( k = 0 ) THEN
    CommandParameters := '*.*' ;
  ELSE
    FOR j := k TO MaxNoOfChar DO
      CommandParameters[ j ] := blank ;
    END ;
  END ;
  DosCommand( 'DIR', CommandParameters, Ok ) ;
END ShowDir ;



P  EDURE Execute ;
(*
  Recive a command from MF10 (or any other sourcecomputer adapted to
  these routines) and sends it back to be executed in MF10.
*)
C  ST stx = 02C ;
     etx = 03C ;
VAR  ExecBuff : ARRAY [0..MaxNoOfChar] OF CHAR ;
     ok       : BOOLEAN ;
     k, i     : INTEGER ;

BEGIN
  REPEAT
    RS232Int.BusyRead( com1char, ok ) ;
  UNTIL (com1char = stx) ;
  RS232Int.Read( com1char ) ;
  k := 0 ;
  WHILE (com1char <> etx) AND (k < MaxNoOfChar) DO
    ExecBuff[ k ] := com1char ;
    INC( k ) ;
    RS232Int.Read( com1char ) ;
  END ;
  FOR i := 0 TO (k - 1) DO
    RS232Int.Write( ExecBuff[ i ] ) ;
```

```
  END ;
  RS232Int.Write( cr ) ;
END Execute ;



PROCEDURE ForgetWord ;
(*
  Recive a word from MF10 (or any other sourcecomputer adapted to
  these routines) and sends it back to be removed from the wordlist
  (WORDS) of MF10.
*)
CONST stx = 02C ;
      etx = 03C ;
      blank = 40C ;
VAR   ExecBuff : ARRAY [0..MaxNoOfChar] OF CHAR ;
      FBuff    : ARRAY [0..5] OF CHAR ;
      ok       : BOOLEAN ;
      k, i     : INTEGER ;

BEGIN
  REPEAT
    RS232Int.BusyRead( comichar, ok ) ;
  UNTIL (comichar = stx) ;
  RS232Int.Read( comichar ) ;
  k := 0 ;
  WHILE (comichar <> etx) AND (k < MaxNoOfChar) DO
    ExecBuff[ k ] := comichar ;
    INC( k ) ;
    RS232Int.Read( comichar ) ;
  END ;
  FBuff := 'FORGET' ;
  FOR i := 0 TO 5 DO
    RS232Int.Write( FBuff[i] ) ;
  END ;
  RS232Int.Write( blank ) ;
  FOR i := 0 TO (k - 1) DO
    RS232Int.Write( ExecBuff[i] ) ;
  END ;
  RS232Int.Write( cr ) ;
END ForgetWord ;



END FileHandler .
```

```
EFINITION MODULE FileHandler ;

XPORT QUALIFIED ReadFileName, CreateFile, OpenFile, CloseFile, ReadFile,
                WriteFile, DownFetch, SecureFetch, DelFile, ShowDir,
                Execute, ForgetWord, FileError, AFileIsOpen, comlchar ;

AR  comlchar     : CHAR ;
    AFileIsOpen : BOOLEAN ;

ROCEDURE ReadFileName ;
ROCEDURE CreateFile ;
ROCEDURE OpenFile ;
ROCEDURE CloseFile ;
ROCEDURE ReadFile ;
ROCEDURE WriteFile ;
ROCEDURE DownFetch ;
ROCEDURE SecureFetch ;
ROCEDURE DelFile ;
ROCEDURE ShowDir ;
ROCEDURE Execute ;
ROCEDURE ForgetWord ;
ROCEDURE FileError( KindOf : CARDINAL ) ;

ND FileHandler.
```

APPENDIX D3.

D3. READFILE.MOD
DUMPFILE.MOD
DELFILE.MOD
SHOWDIR.MOD
CLEARDIR.MOD

```
MPORT RS232Int, InOut, Keyboard ;

ONST stx    = 02C ;
     etx    = 03C ;
     ctrlD  = 04C ;
     ack    = 06C ;
     nak    = 25C ;
     eoln   = 36C ;
     blank  = 40C ;
     Delete     = 177C ;
     BackSpace  = 10C ;
     NameLimit  = 32 ;

YPE  NameType = ARRAY [0..NameLimit] OF CHAR ;

AR   ok : BOOLEAN ;
     FileName : NameType ;
     ch       : CHAR ;
     k, j     : CARDINAL ;

EGIN
 RS232Int.Init( 9600, 2, FALSE, FALSE, 7, ok ) ;
 IF ( ok ) THEN
   RS232Int.StartReading ;
   InOut.WriteLn ;
   InOut.WriteString( 'Enter filename : ' ) ;
   k := 0 ;
   Keyboard.Read( ch ) ;
   WHILE ((ch <> eoln) AND (k <= NameLimit)) DO
     FileName[ k ] := ch ;
     InOut.Write( ch ) ;
     IF (((ch = BackSpace) OR (ch = Delete)) AND (k > 0)) THEN
       DEC( k ) ;
     ELSE
       INC( k ) ;
     END ;
     Keyboard.Read( ch ) ;
   END ;
   FOR j := k TO NameLimit DO
     FileName[ j ] := blank ;
   END ;
   InOut.WriteLn ;
   RS232Int.Write( ctrlD ) ;
   RS232Int.Write( 62C ) ;          (* 62C = OpenFile *)
   RS232Int.Write( stx ) ;
   k := 0 ;
   ch := FileName[ k ] ;
   WHILE ((ch <> blank) AND (k <= NameLimit)) DO
     RS232Int.Write( ch ) ;
     INC( k ) ;
     ch := FileName[ k ] ;
   END ;
   RS232Int.Write( etx ) ;
   RS232Int.Write( ctrlD ) ;
   RS232Int.Write( 64C ) ;          (* 64C = ReadFile *)
   RS232Int.Read( ch ) ;
   IF (ch = ack) THEN
     REPEAT
       k := 0 ;
       RS232Int.BusyRead( ch, ok ) ;
       WHILE (( ok ) AND (ch <> ctrlD) AND (k < 80)) DO
         InOut.Write( ch ) ;
         RS232Int.BusyRead( ch, ok ) ;
```

```
        END ;
        IF ((k = 80) AND ( ok )) THEN
          RS232Int.Write( 23C ) ;       (* 23C = XOFF *)
          InOut.Write( ch ) ;
        ELSE
          RS232Int.Write( 21C ) ;       (* 21C = XON  *)
        END ;
      UNTIL (ch = ctrlD) OR (NOT ok) ;
      IF ( NOT ok ) THEN
        InOut.WriteString( '--- Transmission error. ' ) ;
      END ;
    ELSIF (ch = nak) THEN
      InOut.WriteLn ;
      InOut.WriteString( '--- Unable to open : ' ) ;
      InOut.WriteString( FileName ) ;
    ELSE
      InOut.WriteString( ' ?????????? ' ) ;
    END ;
    RS232Int.Write( ctrlD ) ;
    RS232Int.Write( 63C ) ;         (* 63C = CloseFile *)
    RS232Int.StopReading ;
    InOut.WriteLn ;
    InOut.WriteLn ;
  ELSE
    InOut.WriteLn ;
    InOut.WriteString( 'Initialization failed. ') ;
  END ;
END ReadFile.
```

```
MPORT RS232Int, FileSystem, InOut, Keyboard ;

ONST ctrlD = 04C ;          (*:ASCII characters in octal base, *)
     create = 61C ;
     close  = 63C ;
     write  = 65C ;
     blank  = 40C ;
     stx    = 02C ;
     etx    = 03C ;
     eoln   = 36C ;
     BackSpace = 10C ;
     Delete = 177C ;
     MAX    = 32 ;
     XOFF   = 23C ;
     XON    = 21C ;

YPE  NameType = ARRAY [0..MAX] OF CHAR ;

AR   ok           : BOOLEAN ;
     FileName     : NameType ;
     FileVar      : FileSystem.File ;
     ch, Choice   : CHAR ;
     k, j         : CARDINAL ;


ROCEDURE SendFile( Fi : FileSystem.File ) ;
ONST eof = 0C ;

AR   FileChar, comch : CHAR ;

EGIN
  FileSystem.Reset( Fi ) ;
  IF ( Fi.res = FileSystem.done ) THEN
    REPEAT
      FileSystem.ReadChar( Fi, FileChar ) ;
      RS232Int.Write( FileChar ) ;
      RS232Int.BusyRead( comch, ok ) ;
      IF ( ok ) THEN
        ch := comch ;
        WHILE ( ch = XOFF ) DO
          RS232Int.BusyRead( comch, ok ) ;
          IF ( ok ) THEN
            ch := comch ;
          END ;
        END ;
      END ;
    UNTIL ( FileChar = eof ) ;
  ELSE
    InOut.WriteLn ;
    InOut.WriteString( 'Error in fileoperation. ' ) ;
  END ;
ND SendFile ;



EGIN
  RS232Int.Init( 9600, 2, FALSE, FALSE, 7, ok ) ;
  IF ( ok ) THEN
    RS232Int.StartReading ;
    InOut.WriteLn ;
    InOut.WriteString( 'Enter filename : ' ) ;
```

```
    Keyboard.Read( ch ) ;
    WHILE (( ch <> eoln ) AND ( k < MAX )) DO
      FileName[ k ] := ch ;
      InOut.Write( ch ) ;
      IF ((( ch = BackSpace ) OR ( ch = Delete )) AND ( k > 0 )) THEN
        DEC( k ) ;
      ELSE
        INC( k ) ;
      END ;
      Keyboard.Read( ch ) ;
    END ;
    FOR j := k TO MAX DO
      FileName[ j ] := blank ;
    END ;
    InOut.WriteLn ;
    FileSystem.Lookup( FileVar, FileName, FALSE ) ;
    IF ( FileVar.res = VAL( FileSystem.Response, 1 ) ) THEN
      InOut.WriteString( '--- Unable to open : ' ) ;
      InOut.WriteString( FileName ) ;
      InOut.WriteLn ;
    ELSE
      RS232Int.Write( ctrlD ) ;
      RS232Int.Write( create ) ;
      RS232Int.Write( stx ) ;
      k := 0 ;
      ch := FileName[ k ] ;
      WHILE (( k <= 32 ) AND ( ch <> blank )) DO
        RS232Int.Write( ch ) ;
        INC( k ) ;
        ch := FileName[ k ] ;
      END ;
      RS232Int.Write( etx ) ;
      RS232Int.Write( ctrlD ) ;
      RS232Int.Write( write ) ;
      InOut.WriteLn ;
      InOut.WriteString( 'Continue or Overwrite the file, (C/O) ? : ' ) ;
      REPEAT
        Keyboard.Read( Choice ) ;
      UNTIL (Choice = CHR(67)) OR (Choice = CHR(79)) ;
      InOut.Write( Choice ) ;
      InOut.WriteLn ;
      InOut.WriteString( 'Data transfering......' ) ;
      RS232Int.Write( stx ) ;
      RS232Int.Write( Choice ) ;
      SendFile( FileVar ) ;
      RS232Int.Write( etx ) ;
      RS232Int.Write( ctrlD ) ;
      RS232Int.Write( close ) ;
      InOut.WriteLn ;
      InOut.WriteLn ;
      InOut.WriteString( 'Ready. ' ) ;
      InOut.WriteLn ;
      RS232Int.StopReading ;
    END ;
  ELSE
    InOut.WriteLn ;
    InOut.WriteString( 'Initialization failed. ' ) ;
  END ;
END DumpFile.
```

```
MPORT RS232Int, InOut, Keyboard ;

ONST ctrlD = 04C ;
     stx   = 02C ;
     etx   = 03C ;
     eoln  = 36C ;
     blank = 40C ;
     Delete    = 177C ;
     BackSpace = 10C ;
     NameLimit = 32 ;

YPE  NameType = ARRAY [0..NameLimit] OF CHAR ;

AR   ok       : BOOLEAN ;
     FileName : NameType ;
     ch       : CHAR ;
     k, j     : CARDINAL ;


EGIN
 RS232Int.Init( 9600, 2, FALSE, FALSE, 7, ok ) ;
 IF ( ok ) THEN
   InOut.WriteLn ;
   InOut.WriteString( 'Enter filename : ' ) ;
   k := 0 ;
   Keyboard.Read( ch ) ;
   WHILE ((ch <> eoln) AND (k <= NameLimit)) DO
     FileName[ k ] := ch ;
     InOut.Write( ch ) ;
     IF (((ch = BackSpace) OR (ch = Delete)) AND (k > 0)) THEN
       DEC( k ) ;
     ELSE
       INC( k ) ;
     END ;
     Keyboard.Read( ch ) ;
   END ;
   FOR j := k TO NameLimit DO
     FileName[ k ] := blank ;
   END ;
   InOut.WriteLn ;
   RS232Int.Write( ctrlD ) ;
   RS232Int.Write( 70C ) ;
   RS232Int.Write( stx ) ;
   k := 0 ;
   ch := FileName[ k ] ;
   WHILE ((ch <> blank) AND (k <= NameLimit)) DO
     RS232Int.Write( ch ) ;
     INC( k ) ;
     ch := FileName[ k ] ;
   END ;
   RS232Int.Write( etx ) ;
   InOut.WriteLn ;
 ELSE
   InOut.WriteLn ;
   InOut.WriteString( 'Initialization failed. ' ) ;
 END ;
END DelFile.
```

```
IMPORT RS232Int, InOut ;

CONST ctrlD = 04C ;          (* ASCII characters in octal base *)
      eoln = 36C ;
      EndOfDirectory = 03C ;
      blank = 40C ;
      NameLimit = 20 ;
      BlockSize = 512 ;

TYPE  NameType = ARRAY [0..NameLimit] OF CHAR ;

VAR   ok              : BOOLEAN ;
      NoOfFiles,
      k, j, Hex,
      Occupied,
      TotalOccupied : CARDINAL ;
      ch              : CHAR ;
      Size            : CARDINAL ;
      FileName        : NameType ;


PROCEDURE CONVHEX(HexChar : CHAR ; VAR HexValue : CARDINAL) ;
VAR Value : CARDINAL ;

BEGIN
  Value := ORD(HexChar) ;
  IF (Value <= 39H) THEN
    Value := Value - 30H ;
  ELSIF (Value >= 41H) THEN
    Value := Value - 37H ;
  END ;
  HexValue := Value ;
END CONVHEX ;



BEGIN
  RS232Int.Init( 9600, 2, FALSE, FALSE, 7, ok ) ;
  IF ( ok ) THEN
    InOut.WriteString( ' PROM - Files   (Segment size = 512 bytes) ' ) ;
    InOut.WriteLn ;
    InOut.WriteLn ;
    InOut.WriteString( '    FILENAME          SIZE       ' ) ;
    InOut.WriteLn ;
    InOut.WriteLn ;
    RS232Int.StartReading ;
    RS232Int.Write( ctrlD ) ;
    RS232Int.Write( 71C ) ;
    NoOfFiles := 0 ;
    TotalOccupied := 0 ;
    RS232Int.Read( ch ) ;
    WHILE ( ch <> EndOfDirectory ) DO
      IF ( ch <> eoln ) THEN
        k := 0 ;
        REPEAT
          FileName[ k ] := ch ;
          INC( k ) ;
          RS232Int.Read( ch ) ;
        UNTIL ( ch = eoln ) ;
        FOR j := k TO NameLimit DO
          FileName[ j ] := blank ;
        END ;
        k := 0 ;
```

```
      CONVHEX(ch, Hex) ;
      Size := Hex * 16 ;
      RS232Int.Read( ch ) ;
      CONVHEX(ch, Hex) ;
      Size := Size + Hex ;
      Occupied := Size * BlockSize ;
      TotalOccupied := TotalOccupied + Occupied ;
      InOut.WriteString( FileName ) ;
      InOut.WriteCard( Occupied, 2 ) ;
      InOut.WriteString( ' bytes. ' ) ;
      InOut.WriteLn ;
      INC( NoOfFiles ) ;
      RS232Int.Read( ch ) ;
    END ; (* IF *)
  END ; (* WHILE *)
  RS232Int.StopReading ;
  InOut.WriteLn ;
  InOut.WriteString( '    ' ) ;
  InOut.WriteCard( NoOfFiles, 1 ) ;
  InOut.WriteString( ' File(s)          ' ) ;
  InOut.WriteCard( TotalOccupied, 2 ) ;
  InOut.WriteString( ' bytes occupied. ' ) ;
  InOut.WriteLn ;
ELSE
  InOut.WriteString( ' Initialization failed. ' ) ;
  InOut.WriteLn ;
END ;
END ShowDir.
```

```
IMPORT RS232Int, InOut ;

CONST   ctrlD = 04C ;
        Clear = 102C ;

VAR     ok : BOOLEAN ;

BEGIN
  RS232Int.Init( 9600, 2, FALSE, FALSE, 7, ok ) ;
  IF ( ok ) THEN
    RS232Int.Write( ctrlD ) ;
    RS232Int.Write( Clear ) ;
  ELSE
    InOut.WriteLn ;
    InOut.WriteString( 'Initialization failed. ' ) ;
  END ;
END ClearDir.
```

APPENDIX D4.

D4. SYMB.PAS

```
* Exekveras efter assemblering. Genererar symb.sym textfil  *)
* som innehaller symboler och adresser enligt ebug:s format.*)
* Sista symbolen i *.asm filen maste ha namnet SLUT !!!!!   *)

onst base=$2A67;
     offset=$53A8;

ype hexcontyp=string[2];

ar fil                      : text;
  n,val,mul,add,x,y,max : integer;
  ch                        : char;
  slut                      : array[1..4] of char;
  ok                        : boolean;
  basen                     : integer;



unction Hexcon(hex:integer): hexcontyp;
* omvandlar hex (heltal 0-255) till motsv. hextal i teckenform. *)
ar n                        : integer;
   ch                       : array[1..2] of char;

egin
 := hex div 16;
 case n of
    0..9 : ch[1]:=chr(n+48);
    10..15 : ch[1]:=chr(n+55);
    else ch[1]:='ö';
 end;
 n:= hex mod 16;
 case n of
    0..9 : ch[2]:=chr(n+48);
    10..15 : ch[2]:=chr(n+55);
    else ch[2]:='ö';
 end;
 hexcon:=ch;
nd;(*Hexcon*)


egin (*Huvudprogram*)
 sign(fil,'symb.sym');
 rewrite(fil);
 writeln(fil,'        MODULE   ');
 writeln(fil);
 ax:=400;                  (* max antal symboler som far anvandas  *)
 =offset;                  (* anger startadress i RAM ,kan variera *)
 basen:=Cseg;
 writeln('Cbasen = ',basen);
 basen:=Dseg;
 writeln('Dbasen = ',basen);
 while (n<(offset+max*21))  do
 begin
   mul:=1;
   x:=mem[base:n];
   while x<>0 do
   begin
     write(fil,chr(x));
     if mul<5 then
       slut[mul]:=chr(x);
     n:=n+1;
     mul:=mul+1;
     x:=mem[base:n];
   end;
```

```
    begin
      write(fil,' ');
      mul:=mul+1;
      n:=n+1;
    end;
    x:=mem[base:n];
    n:=n+1;
    y:=mem[base:n];
    write(fil,Hexcon(y),Hexcon(x));
    write(fil,'    ');
    n:=n+2;
    while mem[base:n]=0 do
      n:=n+1;
    writeln(fil);
    if slut='SLUT' then
      n:=offset+max*21+1;
  end;
  writeln('              ** SYMB.SYM genererad ! **');
  if (slut<>'SLUT') then
  begin
    writeln('** Label SLUT fanns inte i assemblerfilen !! **');
    writeln('** SYMB.SYM kan vara felaktig.               **');
  end;
  close(fil);
end.
```

APPENDIX D5.

D5. FIHAND1.ASM
FIHAND2.ASM

```
;*************************************************************************
; FileHandleSystem
; Last change, time : 09:45
;              date : 880601
;*************************************************************************


            ORG #8800H      ; Program startaddress
            JMP INIT



;*************************************************************************
; Variables and Constants
;*************************************************************************

;-------------------------------------------------------------------------
; Zero-page parameters
;-------------------------------------------------------------------------
; * * * variables * * *
PORTA:      EQU 00H         ; Port in processor
PORTC:      EQU 02H         ; Dito
IFR:        EQU 11H         ; Interrupt Flag Register
IER:        EQU 12H         ; Interrupt Enable Register
MCR:        EQU 14H         ; Mode Control Register
SCCR:       EQU 15H         ; Serial Communication Control Register
SCSR:       EQU 16H         ; Serial Communication Status Register
RTREG:      EQU 17H         ; Recive Transmit REGister
COUNTAL:    EQU 18H         ; Low bits of Counter A
COUNTAH:    EQU 19H         ; High bits
ActHeadP:   EQU 0F0H        ; Pointer to current filehead
NameP:      EQU 0F2H        ; Pointer to FileName in current filehead
SizeP:      EQU 0F4H        ; Pointer to Size in current filehead
IndexBlockP:EQU 0F6H        ; Pointer to IndexBlock in current filehead
OpenFileP:  EQU 0F8H        ; Pointer to open file, = nil if none open
ActBlockP:  EQU 0FAH        ; Pointer to actual fileblock
TempP:      EQU 0FCH        ; Temporary pointer


;-------------------------------------------------------------------------
; FileHandlerSystem-Parameters
;-------------------------------------------------------------------------
; * * * variables * * *
InitFlag:   EQU 0100H       ; Tells if PROM is initialized or not
NoOfFiles:  EQU 0103H       ; Actual number of files
FullDir:    EQU 0104H       ; Boolean, TRUE if all files occupied
Available:  EQU 0105H       ; Map over currently existing fileheads
NoOfBlocks: EQU 0108H       ; Number of occupied blocks
FreeBlock:  EQU 010AH       ; Boolean, FALSE if all blocks occupied, else TRUE

; * * * constants * * *
MaxNoFiles: EQU 0102H       ; The number of files is limited
MaxNoBlocks:EQU 0106H       ; Number of present blocks
HeadPoint:  EQU 010BH       ; Pointer to the first filehead in headarea
BlockPoint: EQU 010DH       ; Pointer to the first block in blockarea
HeadSize:   EQU 010FH       ; The size of one filehead (96H = 150)
BlockSize:  EQU 0111H       ; The size of one block in bytes (200H = 512)
```

```
;
; Program parameters
;
; * * * variables * * *
INB:          EQU 7F80H      ; Pointer to last inserted char
UTB:          EQU 7F81H      ; Pointer to next char to pick out
TRANSERR:     EQU 7F82H
CHAR:         EQU 7F83H
INDEX:        EQU 7F84H
EMPTYBUFF:    EQU 7F85H      ; EmptyBuff : Boolean ;
EQUALNAME:    EQU 7F86H      ; EqualName : Boolean ;
No:           EQU 7F87H      ; No : Integer ;
WAITL:        EQU 7F88H
WAITH:        EQU 7F89H
RUN:          EQU 7F8AH
XOFF:         EQU 7F8BH      ; XOFF : Boolean ;
CheckSum:     EQU 7F8CH
Check1:       EQU 7F8DH
Check2:       EQU 7F8EH
FBIndex:      EQU 7F8FH
NoCh1:        EQU 7F90H
NoCh2:        EQU 7F91H
NoOfNAK:      EQU 7F92H
COMCH:        EQU 7F93H
BlockChange:  EQU 7F94H
FileNAME:     EQU 7F95H      ; FileName = array [1..20] of char; (* 890A-891E *)
FrameBuff:    EQU 7F80H      ; FrameBuff = array [1..80] of char;

; * * * constants * * *
BUFF:         EQU 7F00H      ; Pointer to ringbuffer (first address in EBUG-RAM)
STX:          EQU 02H        ; Start of TeXt = ASCII 02
ETX:          EQU 03H        ; End of TeXt = ASCII 03
TRUE:         EQU 01H
FALSE:        EQU 00H
NIL:          EQU 00H




; ***************************************************************************
; Initializing routines
; ***************************************************************************

; ---------------------------------------------------------------------------
; NAME      : INIT
; FUNCTION  : MainInitialization.
;             Initialize stackpointer, Serial Communication Control Register
;             - SCCR, counter A to a communication speed at 9600 baud,
;             Interrupt Enable Register - IER.
;             Also resets ringbuffer pointers INB, UTB and error transfer
;             variable - TRANSERR. Check wether the target memory is
;             initialized or not by reading InitFlag.
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A, X
; ---------------------------------------------------------------------------
INIT:                           ; procedure Init ;
              SEI               ;
              LDX #0EDH         ; begin
              TXS               ;   SP := 0EDH ;  (* Set StackPointer *)
              CLD               ;
              CLC               ;
              LDA #0C4H         ;
              STA SCCR          ;   SCCR := 0C4H ;
```

```
            LDA #00H            ;   (* Serial Communication Control Register *)
            STA INB             ;   INB := 0 ;
            STA UTB             ;   UTB := 0 ;
            STA TRANSERR        ;   TRANSERR := 0 ;
            STA MCR             ;   MCR := 0 ;
            STA OpenFileP       ;   OpenFileP := nil ;
            STA OpenFileP+1     ;
            STA COUNTAH         ;   CounterA := 0009H  ==>  9600 Baud
            STA WAITL           ;
            STA WAITH           ;   Wait := 0 ;
            LDA #FALSE          ;
            STA XOFF            ;   XOFF := FALSE ;
            LDA #09H            ;
            STA COUNTAL         ;
            LDA #40H            ;
            STA IER             ;   IER := 40H ; (* Interrupt Enable Register *)
            LDA InitFlag        ;   if (not InitFlag) then
            EOR #0AAH           ;
            BEQ OK1             ;
            JSR PROMINIT        ;     PromInit ;
            JMP OK2             ;
OK1:                            ;
            LDA InitFlag+1      ;
            EOR #0AAH           ;
            BEQ OK2             ;
            JSR PROMINIT        ;
OK2:                            ;
            CLI                 ;
            JMP WAITCOMMAND     ; end ;


; ------------------------------------------------------------------------
; NAME      : PROMINIT
; FUNCTION  : Initialize the controlarea in PROM
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A, X, Y
; ------------------------------------------------------------------------
PROMINIT:                       ; procedure PromInit ;
            LDA #0AAH           ;
            STA InitFlag        ; begin
            STA InitFlag+1      ;   Initialized := TRUE
            LDA #08H            ;
            STA MaxNoFiles      ;   MaxNoOfFiles := 8 ;
            LDA #3AH            ;
            STA MaxNoBlocks     ;   MaxNoOfBlocks := 58 ;
            LDA #00H            ;
            STA MaxNoBlocks+1   ;
            STA NoOfFiles       ;   NoOfFiles := 0 ;
            STA NoOfBlocks      ;   NoOfBlocks := 0 ;
            STA NoOfBlocks+1    ;
            STA FullDir         ;   FullDir := FALSE ;
            STA Available       ;   Available := 0 ;
            LDA #TRUE           ;
            STA FreeBlock       ;   FreeBlock := TRUE ;
            LDA #00H            ;
            STA BlockPoint      ;   BlockPoint := 0800H ;
            LDA #08H            ;
            STA BlockPoint+1    ;   (* 0800, start address to the first block *)
            LDA #40H            ;
            STA HeadPoint       ;   HeadPoint := 0140H ;
            LDA #01H            ;
            STA HeadPoint+1     ;   (* 0140, start address to the fileheadarea *
            LDA #96H            ;
            STA HeadSize        ;   HeadSize := 0096H ;
            LDA #00H            ;
```

```
            STA HeadSize+1      ;   (* 96H (= 150 bytes), size of a filehead *)
            LDA #00H            ;
            STA BlockSize       ;   BlockSize := 0200H ;
            LDA #02H            ;
            STA BlockSize+1     ;   (* 0200H (= 512 bytes), size of a block *)
            LDA BlockPoint      ;
            STA ActBlockP       ;   ActBlockP := BlockPoint ;
            LDA BlockPoint+1    ;
            STA ActBlockP+1     ;
            LDA #00H            ;
            TAY                 ;
            TAX                 ;
PR_FOR:     CPX MaxNoBlocks     ;   for x := 1 to MaxNoOfBlocks do
            BNE PR1             ;   begin
            JMP PR_FOR_END      ;
PR1:        STA [ActBlockP],Y   ;     ActBlockP^.Ocupied := FALSE ;
            CLC                 ;
            LDA ActBlockP       ;
            ADC BlockSize       ;
            STA ActBlockP       ;     ActBlockP := ActBlockP + BlockSize ;
            LDA ActBlockP+1     ;
            ADC BlockSize+1     ;
            STA ActBlockP+1     ;
            INX                 ;
            LDA #FALSE          ;
            JMP PR_FOR          ;   end ;
PR_FOR_END: RTS                 ; end ;



; ***************************************************************************
; Routines for detecting control characters
; ***************************************************************************

; ---------------------------------------------------------------------------
; NAME      : WAITCOMMAND
; FUNCTION : Waits for the filecontrol-character (ASCII = 04H) to arrive.
;            Then branch to READCOMMCAND.
; INPUT     : None
; OUTPUT    : None
; REGISTER : A
; ---------------------------------------------------------------------------
WAITCOMMAND:                    ; procedure WaitCommand ;
            LDA INB             ; begin
            CMP UTB             ;   repeat
            BEQ WAITCOMMAND     ;     repeat
            JSR GETCHAR         ;       OK := INB <> UTB ;
            LDA CHAR            ;     until OK ;
            EOR #04H            ;     GetChar(Char, OK) ;
            BEQ READCOMMAND     ;   until Char = 04H ;
            JMP WAITCOMMAND     ;   ReadCommand ;
                                ; end ;



; ---------------------------------------------------------------------------
; NAME      : READCOMMAND
; FUNCTION : Waits for the specific filecommand-character to arrive. Then
;            branch to corresponding filehandle-routine.
; INPUT     : None
; OUTPUT    : None
; REGISTER : A
```

```
EADCOMMAND:                     ; procedure ReadCommand ;
            LDA INB             ; begin
            CMP UTB             ;   repeat
            BEQ READCOMMAND     ;     OK := INB <> UTB ;
            JSR GETCHAR         ;   until OK ;
            LDA CHAR            ;   GetChar(Char, OK) ;
            EOR #31H            ;   case Char of
            BNE NEXT1           ;
            JMP CREATEFILE      ;     1 : CreateFile ;
EXT1:       LDA CHAR            ;
            EOR #32H            ;
            BNE NEXT2           ;
            JMP OPENFILE        ;     2 : OpenFile ;
EXT2:       LDA CHAR            ;
            EOR #33H            ;
            BNE NEXT3           ;
            JMP CLOSEFILE       ;     3 : CloseFile ;
EXT3:       LDA CHAR            ;
            EOR #34H            ;
            BNE NEXT4           ;
            JMP READFILE        ;     4 : ReadFile ;
EXT4:       LDA CHAR            ;
            EOR #35H            ;
            BNE NEXT5           ;
            JMP WRITEFILE       ;     5 : WriteFile ;
EXT5:       LDA CHAR            ;
            EOR #36H            ;
            BNE NEXT6           ;
            JMP DOWNFETCH       ;     6 : DownFetch ;
EXT6:       LDA CHAR            ;
            EOR #37H            ;
            BNE NEXT7           ;
            JMP SECUREFETCH     ;     7 : SecureFetch ;
EXT7:       LDA CHAR            ;
            EOR #38H            ;
            BNE NEXT8           ;
            JMP DISPOSEFILE     ;     8 : DisposeFile ;
EXT8:       LDA CHAR            ;
            EOR #39H            ;
            BNE NEXT9           ;
            JMP SHOWDIR         ;     9 : ShowDir ;
EXT9:       LDA CHAR            ;
            EOR #41H            ;
            BNE NEXTA           ;
            JMP EXECUTE         ;     A : Execute ;
EXTA:       LDA CHAR            ;
            EOR #42H            ;
            BNE NEXTB           ;
            JMP FORGETWORD      ;     B : ForgetWord ;
EXTB:       LDA CHAR            ;
            EOR #43H            ;
            BNE NEXTC           ;
            JMP CLEARDIR        ;     C : ClearDir ;
EXTC:       JMP READCOMMAND     ;   end ;
                                ; end ;
```

```
; ************************************************************************
; Subroutines
; ************************************************************************

; ------------------------------------------------------------------------
```

```
; NAME      : GETCHAR
; FUNCTION  : Reads one character from RingBuffer and puts it in variable
;             CHAR. An variable EMPTY will be set if BUFF is empty, i.e
;             when INB = UTB.
; INPUT     : BUFF = Pointer
; OUTPUT    : EMPTYBUFF = Boolean
; REGISTER  : A, X
; ------------------------------------------------------------------------
GETCHAR:                              ; procedure GetChar(Buff : Pointer ;
            SEI                       ;                   var EmptyBuff : Boolean) ;
            TXA                       ;
            PHA                       ; begin
G_IF3:      LDA INB                   ;   if (INB <> UTB) then
            CMP UTB                   ;
            BEQ G_ELSE3               ;   begin
            LDA #00H                  ;
            STA WAITL                 ;
            STA WAITH                 ;     Wait := 0 ;
            LDA #FALSE                ;
            STA EMPTYBUFF             ;     EmptyBuff := FALSE ;
            LDX UTB                   ;
            LDA BUFF,X                ;
            STA CHAR                  ;     Char := Buff[UTB] ;
            INX                       ;
            TXA                       ;
            AND #7FH                  ;     UTB := (UTB + 1) mod 127 ;
G_IF1:      STA UTB                   ;     if (INB > UTB) then
            LDA INB                   ;
            CMP UTB                   ;     begin
            BCC G_ELSE1               ;
            LDA #7FH                  ;       Free := (127 - INB) + UTB ;
            SEC                       ;
            SBC INB                   ;
            CLC                       ;     end
            ADC UTB                   ;
            JMP G_ENDIF1              ;     else
G_ELSE1:                              ;
            LDA UTB                   ;     begin
            SEC                       ;
            SBC INB                   ;       Free := UTB - INB ;
G_ENDIF1:                             ;     end ;
G_IF2:      CMP #19H                  ;     if (Free > 25) then
            BCC G_ENDIF2              ;
            LDA XOFF                  ;
            CMP #FALSE                ;
            BEQ G_ENDIF2              ;
            LDA #11H                  ;       SendA( XON ) ;
            JSR SENDA                 ;
;            LDA PORTA                ;       RTS := 0            (* turn on *)
;            AND #0DFH                ;     else
;            STA PORTA                ;       RTS := 1 ;          (* turn off *)
G_ENDIF2:   JMP G_ENDIF3              ;   end
G_ELSE3:    LDA #TRUE                 ;   else
            STA EMPTYBUFF             ;   begin
            LDA WAITL                 ;     EmptyBuff := TRUE ;
            CLC                       ;
            ADC #01H                  ;     Wait := Wait + 1 ;
            STA WAITL                 ;
            LDA #00H                  ;
            ADC WAITH                 ;     if (Wait = FFH) then
            STA WAITH                 ;
            CMP #0FFH                 ;       Error ;
            BNE G_ENDIF3              ;
            JMP Error                 ;   end ;
G_ENDIF3:   PLA                       ;
            TAX                       ;
```

```
            CLI                     ;
            RTS                     ; end ;



;-------------------------------------------------------------------------------
; NAME      : SENDCHAR
; FUNCTION  : Sends character in memorycell CHAR.
; INPUT     : CHAR : char
; OUTPUT    : None
; REGISTER  : A
;-------------------------------------------------------------------------------
SENDCHAR:                           ; procedure SendChar(Char : char) ;
            PHA                     ;
            LDA CHAR                ; begin
            STA RTREG               ;
SE1:        LDA SCSR                ;   RS232.Write(Char) ;
            AND #40H                ;
            BEQ SE1                 ;   (* Wait for Tx-buffer empty. *)
            PLA                     ;
            RTS                     ; end ;


;-------------------------------------------------------------------------------
; NAME      : SENDA
; FUNCTION  : Sends character in register A.
; INPUT     : A : Register
; OUTPUT    : None
; REGISTER  : A
;-------------------------------------------------------------------------------
SENDA:                              ; procedure SendA(A : Register) ;
            STA RTREG               ; begin
SA1:        LDA SCSR                ;   RS232.Write( A ) ;
            AND #40H                ;
            BEQ SA1                 ;   (* Wait for Tx-buffer empty. *)
            RTS                     ; end ;


;-------------------------------------------------------------------------------
; NAME      : MSDELAY
; FUNCTION  : Creates a one milliseconds delay.
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A
;-------------------------------------------------------------------------------
MSDELAY:                            ; procdure MSDelay ;
            PHA                     ;
            LDA WAITL               ;
            PHA                     ;
            LDA #0AAH               ; begin
            STA WAITL               ;
MSLOOP:     DEC WAITL               ;   Delay( 1 ) ;
            BNE MSLOOP              ;
            PLA                     ;
            STA WAITL               ;
            PLA                     ;
            RTS                     ; end ;


;-------------------------------------------------------------------------------
```

```
; NAME      : ERROR
; FUNCTION  : Different kind of errors will be treated here.
; INPUT     : None
; OUTPUT    : None
; REGISTER  :  -
; ---------------------------------------------------------------------
Error:
            LDA PORTC           ; procedure Error ;
            EOR #0FH            ;
            STA PORTC           ; begin
            LDA #00H            ;
            STA WAITL           ;   Wait := 0 ;
            STA WAITH           ;
            JMP INIT            ; end ;




; ---------------------------------------------------------------------
; NAME      : READFILENAME
; FUNCTION  : Reads charcters from the RingBuffer and puts them in string-
;             variable FileName. The first character must be the control-
;             character STX and subdirectories are ignored. The filename
;             must be ended by ETX.
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A, X
; ---------------------------------------------------------------------
READFILENAME:                   ; procedure ReadFileName ;
            JSR GETCHAR         ; repeat
            LDA CHAR            ;
            EOR #STX            ;   GetChar(Char) ;
            BNE READFILENAME    ; until (Char = STX) ;
            LDA #00H            ;
            STA INDEX           ; Index := 0 ;
R_WHILE:    JSR GETCHAR         ; while ((Char <> ETX ) and
            LDA CHAR            ;         (Index <= 20)) do
            EOR #ETX            ;
            BEQ FILL            ; begin
            LDA INDEX           ;
            EOR #14H            ;   (* 14H = 20 *)
            BEQ FILL            ;
            LDA EMPTYBUFF       ;   if (not EmptyBuff) then
            CMP #TRUE           ;   begin
            BEQ R_ENDIF         ;     read(Char) ;
            LDA CHAR            ;     if (Char = "\") then
            EOR #5CH            ;        (* 5CH = "\" *)
            BNE R_ELSE          ;     begin
            LDA #00H            ;       Index := 0 ;
            STA INDEX           ;     end
            JMP R_ENDIF         ;     else
R_ELSE:     LDX INDEX           ;     begin
            LDA CHAR            ;
            STA FILENAME,X      ;       FileName[Index] := Char ;
            INX                 ;       Index := Index + 1 ;    end ;
            STX INDEX           ;   end ;
R_ENDIF:    JMP R_WHILE         ; end;
FILL:       TXA INDEX           ; while (Index <= 20) do
            EOR #14H            ; begin
            BEQ FULL            ;
            LDA #20H            ;
            STA FILENAME,X      ;   FileName[Index] := "   " ;
            INX                 ;   Index := Index + 1 ;
            JMP FILL            ; end;
FULL:       RTS                 ;
```

```
;-------------------------------------------------------------------------
; NAME      : COMPARE
; FUNCTION  : Compares the recived filename (stored in FILENAME) with one
;             filename, pointed out by NameP, in the directory.
; INPUT     : EQUALNAME = Boolean, NameP = Pointer
; OUTPUT    : EQUALNAME = Boolean
; REGISTER  : A, Y
;-------------------------------------------------------------------------
COMPARE:                          ; procedure Compare(var EqualName : Boolean ;
            TYA                   ;                       NameP : Pointer) ;
            PHA                   ; (* save Y-reg *)
            LDA #TRUE             ; begin
            STA EQUALNAME         ;   EqualName := TRUE ;
            LDY #00H              ;   Index := 0 ;
C_WHILE:    LDA EQUALNAME         ;   while ((EqualName) and
            EOR #TRUE             ;           (Index <= 20)) do
            BNE C_W_END           ;
            TYA                   ;
            EOR #14H              ;
            BEQ C_W_END           ;   begin
            LDA [NameP],Y         ;     if (FILENAME[Index] =
            STA CHAR              ;             NameP[Index]) then
            LDA FILENAME,Y        ;
            CMP CHAR              ;
            BNE C_ELSE            ;
            LDA #TRUE             ;       EqualName := TRUE
            STA EQUALNAME         ;
            JMP C_ENDIF           ;     else
C_ELSE:     LDA #FALSE            ;       EqualName := FALSE ;
            STA EQUALNAME         ;
C_ENDIF:    INY                   ;     Index := Index + 1 ;
            JMP C_WHILE           ;   end ; (* while  *)
C_W_END:    PLA                   ;
            TAY                   ;
            RTS                   ; end ;

;-------------------------------------------------------------------------
; NAME      : COMPARENAME
; FUNCTION  : Looks trough the dictionary comparing the recived filename
;             (stored in FILENAME) with filenames in the dictionary. If the
;             filename is found, Equal will be set TRUE, otherwise FALSE,
;             and a pointer, ActHeadP, will be set.
; INPUT     : HeadPoint = Pointer
; OUTPUT    : EqualName = Boolean, ActHeadP = Pointer, No
; REGISTER  : A, X
;-------------------------------------------------------------------------
COMPARENAME:                      ; procedure CompareFileName(
            TXA                   ;                           EqualName : Boolean ;
            PHA                   ;                           var ActHeadP : Pointer ;
CO_IF:      LDA NoOfFiles         ;                           var No : Integer ) ;
            EOR #00H              ; begin
            BEQ CO_ELSE           ;   if (NoOfFiles <> 0) then
            LDA HeadPoint         ;
```

```
                STA ActHeadP          ;   begin
                STA NameP             ;
                LDA HeadPoint+1       ;
                STA ActHeadP+1        ;     ActHeadP := HeadPoint ;
                STA NameP+1           ;     NameP := ActHeadP ;
                LDA #FALSE            ;
                STA EQUALNAME         ;     EqualName := FALSE ;
                LDA #01H              ;
                STA No                ;     No := 1 ;
                TAX                   ;     X := (0000 0001)B ;
                AND Available         ;     if (Available AND X <> 0) then
                BEQ CO_WHILE          ;
                JSR COMPARE           ;       Compare(EqualName, NameP) ;
CO_WHILE:       LDA EQUALNAME         ;     while ((not EqualName) and
                EOR #TRUE             ;             (No < MaxNoFiles)) do
                BEQ CO_W_END          ;
                LDA No                ;     begin
                EOR MaxNoFiles        ;
                BEQ CO_W_END          ;
                LDA ActHeadP          ;       ActHeadP := ActHeadP + HeadSize ;
                CLC                   ;
                ADC HeadSize          ;
                STA ActHeadP          ;
                STA NameP             ;       NameP := ActHeadP ;
                LDA ActHeadP+1        ;
                ADC HeadSize+1        ;
                STA ActHeadP+1        ;
                STA NameP+1           ;
                TXA                   ;
                ASL A                 ;       RotateLeft(X) ;
                TAX                   ;
                AND Available         ;       if (Available AND X <> 0) then
                BEQ CO1               ;
                JSR COMPARE           ;         Compare(EqualName, NameP) ;
CO1:            INC No                ;       No := No + 1 ;
                JMP CO_WHILE          ;     end ;   (* while *)
CO_W_END:       JMP CO_ENDIF          ;   end
CO_ELSE:        LDA #FALSE            ;   else
                STA EQUALNAME         ;     EqualName := FALSE ;
CO_ENDIF:       PLA                   ;
                TAX                   ;
                RTS                   ; end ;
```

```
; ---------------------------------------------------------------------
; NAME      : LASTBLOCK
; FUNCTION  : A pointer, ActBlockP, will be set to the last block belonging
;             to the actual file. If the file is empty, i.e no blocks is
;             occupied, the pointer will be set to an erroneous value. This
;             condition must be checked outside this subroutine.
; INPUT     : None
; OUTPUT    : ActBlockP : Pointer ;
; REGISTER  : A, Y
; ---------------------------------------------------------------------
LASTBLOCK:                            ; procedure LastBlock(var ActBlockP : Pointer)
                LDA No                ;
                PHA                   ;
                LDY #00H              ; begin
                LDA [SizeP],Y         ;
                STA No                ;
                DEC No                ;
                CLC                   ;
```

```
            LDA No                  ;   No := 2 * (SizeP^.Size - 1) ;
            ADC No                  ;
            TAY                     ;
            LDA [IndexBlockP],Y     ;
            STA ActBlockP           ;   ActBlockP := IndexBlockP^.IndexBlock[No] ;
            INY                     ;
            LDA [IndexBlockP],Y     ;
            STA ActBlockP+1         ;
            PLA                     ;
            STA No                  ;
            RTS                     ; end ;



;-------------------------------------------------------------------------
; NAME     : NEWBLOCK
; FUNCTION : Looks for a free block to expand the file. A pointer, ActBlockP,
;            will be set to the first free block found and this block will
;            be marked occupied. If all blocks are occupied the pointer will
;            be set to nil.
; INPUT    : None
; OUTPUT   : ActBlockP : Pointer ;
; REGISTER : A, X, Y
;-------------------------------------------------------------------------
NEWBLOCK:                           ; procedure NewBlock(var ActBlockP : Pointer) ;
            TXA                     ;
            PHA                     ;
            TYA                     ;
            PHA                     ; begin
            LDA BlockPoint          ;
            STA ActBlockP           ;   ActBlockP := BlockPoint ;
            LDA BlockPoint+1        ;
            STA ActBlockP+1         ;
            LDX #00H                ;   x := 0 ;
            LDY #00H                ;
NE_WHILE:   LDA [ActBlockP],Y       ;   while (ActBlockP^.Ockupied) and
            EOR #FALSE              ;             (x < MaxNoBlocks) do
            BEQ NE_IF               ;   begin
            CPX MaxNoBlocks         ;
            BCS NE_W_END            ;
            CLC                     ;
            LDA ActBlockP           ;
            ADC BlockSize           ;
            STA ActBlockP           ;     ActBlockP := ActBlockP + BlockSize ;
            LDA ActBlockP+1         ;
            ADC BlockSize+1         ;
            STA ActBlockP+1         ;
            INX                     ;     x := x + 1 ;
            JMP NE_WHILE            ;   end ;
NE_IF:      LDA #TRUE               ;   if (not ActBlockP^.Ockupied) then
            STA [ActBlockP],Y       ;   begin
            INC NoOfBlocks          ;     ActBlockP^.Ockupied := TRUE ;
            LDA MaxNoBlocks         ;     NoOfBlocks := NoOfBlocks + 1 ;
            CMP NoOfBlocks          ;
            BNE NE1                 ;
            LDA #FALSE              ;     FreeBlock := MaxNoBlocks = NoOfBlocks ;
            STA FreeBlock           ;   end
NE1:        JMP NE_END              ;   else
NE_W_END:   LDA #NIL                ;
            STA ActBlockP           ;     ActBlockP := nil ;
            STA ActBlockP+1         ;
NE_END:     PLA                     ;
            TAY                     ;
            PLA                     ;
            TAX                     ;
```

```
                RTS                     ; end ;


; ---------------------------------------------------------------------
; NAME      : CLEARBLOCK
; FUNCTION  : All the blocks corresponding to the open file returns to the
;             system by setting Ockupied = FALSE. Size becomes zero.
; INPUT     : IndexBlockP : Pointer
; OUTPUT    : Size : integer
; REGISTER  : A, X, Y
; ---------------------------------------------------------------------
CLEARBLOCK:                             ; procedure ClearBlock(IndexBlockP : Pointer ;
                LDA No                  ;                         ( var Size : integer ) ;
                PHA                     ;
                TXA                     ;
                PHA                     ; begin
                TYA                     ;
                PHA                     ;
                LDY #00H                ;   y := 0 ;
                LDX #00H                ;   x := 0 ;
                LDA [SizeP],Y           ;
                STA No                  ;   No := SizeP^.Size ;
CB_WHILE:       CPX No                  ;   while (x < No) do
                BEQ CB_W_END            ;   begin
                LDA [IndexBlockP],Y     ;     ActBlockP :=
                STA ActBlockP           ;             IndexBlockP^.IndexBlock[ y ] ;
                INY                     ;     y := y + 1 ;
                LDA [IndexBlockP],Y     ;     ActBlockP+1 :=
                STA ActBlockP+1         ;             IndexBlockP^.IndexBlock[ y ] ;
                INY                     ;
                TYA                     ;
                PHA                     ;
                LDA #FALSE              ;
                LDY #00H                ;
                STA [ActBlockP],Y       ;     ActBlockP^.Ockupied := FALSE ;
                PLA                     ;
                TAY                     ;
                INX                     ;     x := x + 1 ;
                DEC NoOfBlocks          ;     NoOfBlocks := NoOfBlocks - 1 ;
                LDA #TRUE               ;
                STA FreeBlock           ;     FreeBlock := TRUE ;
                JMP CB_WHILE            ;
CB_W_END:       LDA #00H                ;   end ;
                LDY #00H                ;
                STA [SizeP],Y           ;   SizeP^.Size := 0 ;
                PLA                     ;
                TAY                     ;
                PLA                     ;
                TAX                     ;
                PLA                     ;
                STA No                  ;
                RTS                     ; end ;


; ---------------------------------------------------------------------
; NAME      : CONHEX
; FUNCTION  : Converts A=Byte into A=ASCII(low) and X=ASCII(high).
; INPUT     : CHAR : char ;
; OUTPUT    : A, X : register ;
; REGISTER  : A, X
; ---------------------------------------------------------------------
CONHEX:                                 ; procedure ConHex(C:Char; A,X:integer) ;
                LDA CHAR                ; begin
```

```
            LSR A                   ;
            LSR A                   ;   A := C ;
            LSR A                   ;
            LSR A                   ;   RotateRight(A, 4) ;
            CMP #0AH                ;
            BCS CON1                ;   if (A <= 09) then
            CLC                     ;
            ADC #30H                ;     A := A + 48
            JMP CON2                ;   else
CON1:       CLC                     ;     A := A + 55 ;
            ADC #37H                ;
CON2:       PHA                     ;   Push(A) ;
            LDA CHAR                ;   A := Char ;
            AND #0FH                ;   A := (A AND 00001111) ;
            CMP #0AH                ;   if (A <= 09) then
            BCS CON3                ;
            CLC                     ;     A := A + 48
            ADC #30H                ;   else
            JMP CON4                ;     A := A + 55 ;
CON3:       CLC                     ;
            ADC #37H                ;   X := A ;
CON4:       TAX                     ;   Pop(A) ;
            PLA                     ;
            RTS                     ; end ;


;-------------------------------------------------------------------------
; NAME      : SENDFRAME
; FUNCTION  : Sends one frame including header (ctrlA), number of bytes in
;             data, data and checksum. If ComCh = eof sends an endframe
;             consisting of header and four zeroes.
; INPUT     : ComCh : Char ;
; OUTPUT    : None
; REGISTER  : A, X
;-------------------------------------------------------------------------
SENDFRAME:                          ; procedure SendFrame(ComCh:Char) ;
SE_IF8:     LDA COMCH               ; if (ComCh = eoln) then
            CMP #1AH                ; begin
            BEQ SE_ELSE8            ;
            LDA #01H                ;
            JSR SENDA               ;   SendA(soh);
            LDA NoCh1               ;
            JSR SENDA               ;   SendA(NoCh1);
            LDA NoCh2               ;
            JSR SENDA               ;   SendA(NoCh2);
            LDX #00H                ;   for i := 1 to FBIndex do
SE_FOR:     LDA FrameBuff,X         ;   begin
            JSR SENDA               ;     A := FrameBuff[i] ;
            INX                     ;     SendA(A) ;
            CPX FBIndex             ;
            BNE SE_FOR              ;   end ;
            LDA Check1              ;   SendA(Check1) ;
            JSR SENDA               ;
            LDA Check2              ;   SendA(Check2) ;
            JSR SENDA               ;
            JMP SE_ENDIF8           ;   end
SE_ELSE8:   LDA #01H                ;   else
            JSR SENDA               ;   begin
            LDA #30H                ;     SendA(soh) ;
            JSR SENDA               ;     SendA('0') ;
            LDA #30H                ;
            JSR SENDA               ;     SendA('0') ;
            LDA #30H                ;
            JSR SENDA               ;     SendA('0') ;
            LDA #30H                ;
```

```
                JSR SENDA               ;       SendA('O') ;
SE_ENDIFB:      RTS                     ;     end ;



;-------------------------------------------------------------------------
; NAME      : EXFO_READ
; FUNCTION  : Subroutine for EXecute and FOrgetword.
; INPUT     : None
; OUTPUT    : NoOfBuffChar : integer ; (= reg. X)
; REGISTER  : A, X
;-------------------------------------------------------------------------
EXFO_READ:                              ; procedure EXFO_Read(var NoOfBuffChar) ;
EXFO_REP:       JSR GETCHAR             ; begin
                LDA #STX                ;   repeat
                CMP CHAR                ;     GetChar(Char) ;
                BNE EXFO_REP            ;   until (Char = stx) ;
                LDX #00H                ;   NoOfBuffChar := 0 ;
EXFO_WHILE:     JSR GETCHAR             ;   GetChar(Char) ;
                LDA EMPTYBUFF           ;   while (Char <> stx) do
                CMP #TRUE               ;   begin
                BEQ EXFO_WHILE          ;     FrameBuff[NoOfBuffChar] := Char ;
                LDA CHAR                ;
                CMP #ETX                ;     NoOfBuffChar := NoOfBuffChar + 1 ;
                BEQ EXFO_END            ;
                STA FrameBuff,X         ;     GetChar(Char) ;
                INX                     ;
                JMP EXFO_WHILE          ;   end ;
EXFO_END:       RTS                     ; end ;



;-------------------------------------------------------------------------
; NAME      : EXFO_WRITE
; FUNCTION  : Subroutine to EXecute and FOrgetword.
; INPUT     : NoOfBuffChar : integer ; (= reg. X)
; OUTPUT    : None
; REGISTER  : A, X, Y
;-------------------------------------------------------------------------
EXFO_WRITE:                             ; procedure EXFO_Write(NoOfBuffChar) ;
                LDA No                  ; begin
                PHA                     ;
                STX No                  ;   for x := 1 to NoOfBuffChar do
                LDX #00H                ;   begin
EXFO_FOR:       LDA FrameBuff,X         ;     A := FrameBuff[ x ] ;
                JSR SENDA               ;
                INX                     ;     Send( A ) ;
                CPX No                  ;
                BNE EXFO_FOR            ;   end ;
                LDA #0DH                ;
                JSR SENDA               ;   Send( cr ) ;
EXFOEND:        PLA                     ;
                STA No                  ;
                RTS                     ; end ;



;-------------------------------------------------------------------------
; NAME      : CREATEFILE
; FUNCTION  : Search for the file "FileName". If the file exist nothing
;             happens, if it doesn't exist it will be created.
; INPUT     : None
; OUTPUT    : OpenFileP = Pointer
; REGISTER  : A, Y
;-------------------------------------------------------------------------
```

```
CREATEFILE:                    ; procedure CreateFile(var OpenFileP : Pointer)
            TYA                ;
            PHA                ; begin
            TXA                ;
            PHA                ;
            JSR READFILENAME   ;   ReadFileName ;
            JSR COMPARENAME    ;   CompareName(EqualName, ActHeadP, No) ;
CR_IF:      LDA EQUALNAME      ;   if ((not EqualName) and
            EOR #TRUE          ;       (not FullDir) and
            BNE CR1            ;       (FreeBlock)) then
            JMP CR_ELSIF_EQ    ;
CR1:        LDA FullDir        ;
            EOR #TRUE          ;
            BEQ CR2            ;
            LDA FreeBlock      ;   begin
            CMP #TRUE          ;
            BEQ CR3            ;
CR2:        JMP CR_ELSIF_FU    ;
CR3:        LDA #00H           ;
            STA ActHeadP       ;
            STA ActHeadP+1     ;
            LDA #00H           ;
            STA No             ;     No := 0 ;
            LDX #01H           ;     X := (0000 0001)B ;
            TXA                ;
CR_WHILE2:  AND Available      ;     while (Available AND X <> 0) do
            BEQ CR_END_W2      ;     begin
            INC No             ;       No := No + 1 ;
            TXA                ;
            ASL A              ;       RotateLeft(X) ;
            TAX                ;
            JMP CR_WHILE2      ;     end;
CR_END_W2:  TXA                ;
            EOR Available      ;     Available := Available EXOR X ;
            STA Available      ;
            LDY No             ;
CR_WHILE:                      ;
            CPY #00H           ;
            BEQ CR_OUTMUL      ;
            CLC                ;
            LDA HeadSize       ;
            ADC ActHeadP       ;     ActHeadP := No * HeadSize +
            STA ActHeadP       ;                 HeadPoint ;
            LDA HeadSize+1     ;
            ADC ActHeadP+1     ;
            STA ActHeadP+1     ;
            DEY                ;
            JMP CR_WHILE       ;
CR_OUTMUL:  LDA ActHeadP       ;
            CLC                ;
            ADC HeadPoint      ;
            STA ActHeadP       ;
            STA NameP          ;     NameP := ActHeadP ;
            STA OpenFileP      ;
            LDA ActHeadP+1     ;
            ADC HeadPoint+1    ;
            STA ActHeadP+1     ;
            STA NameP+1        ;
            STA OpenFileP+1    ;     OpenFileP := ActHeadP ;
            CLC                ;
            LDA ActHeadP       ;
            ADC #14H           ;
            STA SizeP          ;     SizeP := ActHeadP + 20 ;
            LDA ActHeadP+1     ;
            ADC #00H           ;
            STA SizeP+1        ;
```

```
                CLC                     ;
                LDA ActHeadP            ;
                ADC #16H                ;
                STA IndexBlockP         ;     IndexBlockP := ActHeadP + 22 ;
                LDA ActHeadP+1          ;
                ADC #00H                ;
                STA IndexBlockP+1       ;
                LDY #00H                ;
CR_FOR1:        LDA FILENAME,Y          ;     for y := 1 to 20 do
                STA [NameP],Y           ;
                INY                     ;
                CPY #14H                ;       NameP[y] := FileName[y] ;
                BEQ CR_FOR1_END         ;
                JMP CR_FOR1             ;
CR_FOR1_END:
                LDY #00H                ;
                LDA #00H                ;
                STA [SizeP],Y           ;     SizeP := 0 ;
                INY                     ;
                STA [SizeP],Y           ;
                LDY #00H                ;
                LDA #00H                ;
CR_FOR2:        STA [IndexBlockP],Y     ;    for y := 1 to 128 do
                INY                     ;
                CPY #80H                ;        IndexBlockP[y] := nil ;
                BEQ CR_FOR2_END         ;
                JMP CR_FOR2             ;
CR_FOR2_END:                            ;
                INC NoOfFiles           ;     NoOfFiles := NoOfFiles + 1 ;
                LDA NoOfFiles           ;
                CMP MaxNoFiles          ;
                BNE NOTFULL             ;
                LDA #TRUE               ;     FullDir := NoOfFiles = MaxNoFiles ;
                STA FullDir             ;
                JMP CR_ENDIF            ;
NOTFULL:        LDA #FALSE              ;
                STA FullDir             ;
                JMP CR_ENDIF            ;   end
CR_ELSIF_EQ:                            ;   else if (EqualName) then
                LDA ActHeadP            ;   begin
                STA OpenFileP           ;     OpenFileP := ActHeadP ;
                STA NameP               ;     NameP := ActHeadP ;
                LDA ActHeadP+1          ;
                STA OpenFileP+1         ;
                STA NameP+1             ;
                CLC                     ;
                LDA ActHeadP            ;
                ADC #14H                ;
                STA SizeP               ;     SizeP := ActHeadP + 20 ;
                LDA ActHeadP+1          ;
                ADC #00H                ;
                STA SizeP+1             ;
                CLC                     ;
                LDA ActHeadP            ;
                ADC #16H                ;
                STA IndexBlockP         ;     IndexBlockP := ActHeadP + 22 ;
                LDA ActHeadP+1          ;
                ADC #00H                ;
                STA IndexBlockP+1       ;
                JMP CR_ENDIF            ;
CR_ELSIF_FU:                            ;   else if (FullDir) or (not FreeBlock) then
                JMP Error               ;     Error ;
CR_ENDIF:                               ;
                PLA                     ;
                TAX                     ;
                PLA                     ;
```

```
        TAY                     ;
        JMP WAITCOMMAND         ; end ;
```

```
;-------------------------------------------------------------------------------
; NAME      : OPENFILE
; FUNCTION  : Looks for a file specified by "FileName" and if it's found,
;             a pointer, OpenFileP, to the file will be set, otherwise nil.
; INPUT     : None
; OUTPUT    : OpenFileP = Pointer
; REGISTER  : A
;-------------------------------------------------------------------------------
OPENFILE:                           ; procedure OpenFile(var OpenFileP : Pointer) ;
          NOP                       ; begin
          JSR READFILENAME          ;   ReadFileName ;
          JSR COMPARENAME           ;   CompareName(EqualName,ActHeadP, No) ;
OP_IF:    LDA EQUALNAME             ;   if (EqualName) then
          EOR #TRUE                 ;
          BNE OP_ELSE               ;
          LDA ActHeadP              ;     OpenFileP := ActHeadP ;
          STA OpenFileP             ;
          STA NameP                 ;     NameP := ActHeadP ;
          LDA ActHeadP+1            ;
          STA OpenFileP+1           ;
          STA NameP+1               ;
          CLC                       ;
          LDA ActHeadP              ;
          ADC #14H                  ;
          STA SizeP                 ;     SizeP := ActHeadP + 20 ;
          LDA ActHeadP+1            ;
          ADC #00H                  ;
          STA SizeP+1               ;
          CLC                       ;
          LDA ActHeadP              ;
          ADC #16H                  ;
          STA IndexBlockP           ;     IndexBlockP := ActHeadP + 22 ;
          LDA ActHeadP+1            ;
          ADC #00H                  ;
          STA IndexBlockP+1         ;
          JMP OP_ENDIF              ;
OP_ELSE:  LDA #NIL                  ;   else
          STA OpenFileP             ;
          STA OpenFileP+1           ;     OpenFileP := nil ;
OP_ENDIF:                           ;
          JMP WAITCOMMAND           ; end ;


;-------------------------------------------------------------------------------
; NAME      : CLOSEFILE
; FUNCTION  : The pointer corresponding to the open file will be set to nil
;             (the file is closed).
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A
;-------------------------------------------------------------------------------
CLOSEFILE:                          ; procedure CloseFile ;
          LDA #NIL                  ; begin
          STA OpenFileP             ;   OpenFileP := nil ;
          STA OpenFileP+1           ;
          JMP WAITCOMMAND           ; end ;


;-------------------------------------------------------------------------------
; NAME      : READFILE
; FUNCTION  : Sends characters to be displayed on a screen.
```

```
REGISTER : A, Y
-------------------------------------------------------------------------------
READFILE:                       ; procedure ReadFile ;
          LDA No                ; begin
          PHA                   ;
          TYA                   ;
          PHA                   ;
RE_IF1:   LDA OpenFileP         ;   if (OpenFile) then
          CMP #NIL              ;
          BNE RE1               ;
          LDA OpenFileP+1       ;
          CMP #NIL              ;   begin
          BNE RE1               ;
          JMP RE_ELSIF1         ;
RE1:      LDA #06H              ;
          STA CHAR              ;
          JSR SENDCHAR          ;     SendChar( ack ) ;
          LDY #00H              ;
          LDA [SizeP],Y         ;
          CLC                   ;
          ADC [SizeP],Y         ;
          STA Index             ;
RE_FOR:   CPY Index             ;     for y := 0 to (Size-1) do
          BEQ RE_FOR_END        ;     begin
          LDA [IndexBlockP],Y   ;
          STA ActBlockP         ;       ActBlockP := IndexBlockP^.IndexBlock[y];
          STA TempP             ;
          INY                   ;
          LDA [IndexBlockP],Y   ;
          STA ActBlockP+1       ;
          STA TempP+1           ;       TempP := ActBlockP ;
          LDA #00H              ;
          STA No                ;       No := 0 ;
          TYA                   ;
          PHA                   ;
          LDY #01H              ;       y := 1 ;
          LDA [TempP],Y         ;
          STA CHAR              ;       Char := TempP^.Data[ y ] ;
RE_WHILE: CMP #1AH              ;       while (Char <> eof) and
          BEQ RE_W_END          ;             (y,No < BlockSize) do
          CPY BlockSize         ;
          BNE RE2               ;
          LDA No                ;       begin
          CMP BlockSize+1       ;
          BEQ RE_W_END          ;
RE2:      JSR SENDCHAR          ;         SendChar(Char) ;
          INY                   ;         y := y + 1 ;
RE_IF2:   CPY #00H              ;         if (y = 00H) then
          BNE RE_ENDIF2         ;         begin
          LDA ActBlockP+1       ;
          CLC                   ;           TempP := ActBlockP + 0100H ;
          ADC #01H              ;
          STA TempP+1           ;
          INC No                ;           No := No + 1 ;
RE_ENDIF2: LDA INB              ;         end ;
          CMP UTB               ;         if (INB <> UTB) then
          BEQ RE_ENDIF3         ;         begin
          JSR GETCHAR           ;           GetChar( Char ) ;
RE_WHILE2: LDA CHAR             ;           while (Char = XOF) do
          CMP #13H              ;           begin
          BNE RE_W2_END         ;
          JSR MSDELAY           ;             Delay( 1 ) ;
          JSR GETCHAR           ;             GetChar( Char ) ;
          JMP RE_WHILE2         ;
```

```
            STA CHAR            ;          Char := TempP^.Data[ y ] ;
            JMP RE_WHILE        ;        end ;
E_W_END:    PLA                 ;      end ;
            TAY                 ;
            INY                 ;
            JMP RE_FOR          ;
E_FOR_END:  LDA #NIL            ;
            STA OpenFileP       ;      OpenFileP := nil ;
            STA OpenFileP+1     ;
            JMP RE_ENDIF1       ;    end
E_ELSIF1:   LDA #15H            ;    else
            STA CHAR            ;
            JSR SENDCHAR        ;      SendChar( nak ) ;
E_ENDIF1:   LDA #04H            ;    end ;
            STA CHAR            ;
            JSR SENDCHAR        ;    SendChar( eot ) ;
            PLA                 ;
            TAY                 ;
            PLA                 ; end ;
            STA No              ;
            JMP WAITCOMMAND     ;
```

```
NAME      : WRITEFILE
FUNCTION  : "WriteFile" reads a character from "console" (the RS232 -
             serialinterface) on MF10 and writes it to an openfile.
INPUT     :
OUTPUT    :
REGISTER  : A, Y
```

```
WRITEFILE:                      ; procedure WriteFile ;
            LDA No              ; begin
            PHA                 ;
            TYA                 ;
            PHA                 ;
E_IFWR:     LDA FreeBlock       ;   if (OpenFileP <> nil) and
            CMP #TRUE           ;      ( FreeBlock ) then
            BNE WR14            ;
            LDA OpenFileP       ;
            CMP #NIL            ;
            BNE WR_REPEAT1      ;
            LDA OpenFileP+1     ;
            CMP #NIL            ;   begin
            BNE WR_REPEAT1      ;
E_4:        JMP WR_ENDIFWR      ;
E_REPEAT1:  JSR GETCHAR         ;     repeat
            LDA #00H            ;
            STA WAITL           ;       Wait := 0 ;
            LDA CHAR            ;       GetChar(Char) ;
            EOR #STX            ;
E_UNTIL1:   BNE WR_REPEAT1      ;     until (Char = STX) ;
E_REPEAT2:  JSR GETCHAR         ;     repeat
            LDA CHAR            ;       GetChar(Char) ;
            EOR #STX            ;
E_UNTIL2:   BEQ WR_REPEAT2      ;     until (Char <> STX) ;
            LDA #00H            ;
            STA No              ;     No := 0 ;
            LDA CHAR            ;
E_IF1:      EOR #43H            ;     if (Char = 'C') then    (* C = Continue *)
            BNE WR_ELSE1        ;     begin
            LDY #00H            ;
            LDA [SizeP],Y       ;
```

```
                JSR NEWBLOCK          ;          NewBlock(ActBlockP) ;
WR_IF10:        LDA ActBlockP         ;          if (ActBlockP <> nil) then
                CMP #NIL              ;
                BNE WR12              ;          begin
                LDA ActBlockP+1       ;
                CMP #NIL              ;
                BNE WR12              ;
                JMP WR_ELSIF10        ;
WR12:           LDA ActBlockP         ;
                STA TempP             ;            TempP := ActBlockP ;
                LDA ActBlockP+1       ;
                STA TempP+1           ;
                LDY #00H              ;
                LDA #01H              ;
                STA [SizeP],Y         ;            SizeP^.Size := 1 ;
                LDA ActBlockP         ;
                STA [IndexBlockP],Y   ;            IndexBlockP^.IndexBlock[y] :=
                INY                   ;                                  ActBlockP ;
                LDA ActBlockP+1       ;          end
                STA [IndexBlockP],Y   ;          else
                JMP WR_ENDIF10        ;            Error ;
WR_ELSIF10:     JMP Error             ;        end ;
WR_ENDIF10:                           ;        else
                JMP WR_ENDIF2         ;        begin
WR_ELSE2:       JSR LASTBLOCK         ;          LastBlock(ActBlockP) ;
                LDA ActBlockP         ;
                STA TempP             ;          TempP := ActBlockP ;
                LDA ActBlockP+1       ;
                STA TempP+1           ;
                LDY #01H              ;          y := 1 ;
                LDA [TempP],Y         ;          Char := TempP^.Data[ y ] ;
WR_WHILE1:      EOR #1AH              ;          while (Char <> eof) do  (* eof = 1AH *
                BEQ WR_W1_END         ;          begin
                INY                   ;            y := y + 1 ;
WR_IF3:         CPY #00H              ;            if (y = 00H) then   (* wrap around *
                BNE WR_ENDIF3         ;            begin
                CLC                   ;
                LDA ActBlockP+1       ;              TempP := ActBlockP + #0100H ;
                ADC #01H              ;
                STA TempP+1           ;              No := No + 1 ;
                INC No                ;            end ;
WR_ENDIF3:      LDA [TempP],Y         ;            Char := TempP^.Data[ y ] ;
                JMP WR_WHILE1         ;
WR_W1_END:                            ;          end ;
WR_ENDIF2:      JMP WR_ENDIF1         ;        end ;
WR_ELSE1:       LDA CHAR              ;      end
                EOR #4FH              ;      else if (Char = 'O') then
                BNE WR_ERROR          ;      begin
                JSR CLEARBLOCK        ;        ClearBlock(IndexBlockP, Size) ;
                LDY #00H              ;        y := 0 ;
WR_IF9:         LDA [IndexBlockP],Y   ;        if (IndexBlockP[ y ] = nil) then
                CMP #NIL              ;
                BEQ WR10              ;        begin
                JMP WR_ELSE9          ;
WR10:           INY                   ;
                LDA [IndexBlockP],Y   ;
                CMP #NIL              ;
                BEQ WR11              ;
                JMP WR_ELSE9          ;
WR11:           JSR NEWBLOCK          ;          NewBlock(ActBlockP) ;
WR_IF11:        LDA ActBlockP         ;          if (ActBlockP <> nil) then
                CMP #NIL              ;
                BNE WR13              ;          begin
                LDA ActBlockP+1       ;
```

```
        JMP WR_ELSIF11          ;
        LDY #00H                ;           y := 0 ;
        LDA ActBlockP           ;
        STA TempP               ;           TempP := ActBlockP ;
        STA [IndexBlockP],Y     ;           IndexBlockP^.IndexBlockP[ y ] :=
        INY                     ;                       ActBlockP ;
        LDA ActBlockP+1         ;
        STA TempP+1             ;
        STA [IndexBlockP],Y     ;         end
        JMP WR_ENDIF11          ;         else
11:     JMP Error               ;           Error
11:                             ;
        JMP WR_ENDIF9           ;       else
        LDY #00H                ;       begin
        LDA [IndexBlockP],Y     ;         y := 0 ;
        STA ActBlockP           ;         ActBlockP := IndexBlockP^.IndexBlockP
        STA TempP               ;         TempP := ActBlockP ;
        INY                     ;         y := y + 1 ;
        LDA [IndexBlockP],Y     ;         ActBlockP+1 :=IndexBlockP^.IndexBlock
        STA ActBlockP+1         ;
        STA TempP+1             ;         TempP+1 := ActBlockP+1 ;
9:      LDA #01H                ;       end ;
        LDY #00H                ;
        STA [SizeP],Y           ;       SizeP^.Size := 1 ;
        LDA #TRUE               ;
        STA [ActBlockP],Y       ;       ActBlockP^.Ockupied := TRUE ;
        LDY #01H                ;       y := 1 ;
        JMP WR_ENDIF1           ;     end
        JMP Error               ;     else
                                ;       Error ;
1:                              ;     repeat
T3:     JSR GETCHAR             ;       GetChar(Char) ;
        LDA EMPTYBUFF           ;       if (not EmptyBuff) and
        CMP #TRUE               ;          (Char <> ETX) then
        BEQ WR1                 ;
        LDA CHAR                ;
        CMP #ETX                ;       begin
        BNE WR2                 ;
        JMP WR_ENDIF4           ;
        LDA CHAR                ;
        STA [TempP],Y           ;         TempP^.Data[ y ] := Char ;
        INY                     ;         y := y + 1 ;
        CPY #00H                ;         if (y = 00H) then
        BNE WR_ENDIF5           ;         begin
        CLC                     ;
        LDA ActBlockP+1         ;
        ADC #01H                ;           TempP := ActBlockP + 0100H ;
        STA TempP+1             ;           No := No + 1 ;
        INC No                  ;         end ;
                                ;
5:      CPY BlockSize           ;         if (y = BlockSize) and
        BEQ WR5                 ;            (No = BlockSize+1) then
        JMP WR_ENDIF7           ;
        LDA No                  ;         begin
        CMP BlockSize+1         ;
        BEQ WR6                 ;
        JMP WR_ENDIF7           ;
        LDA FreeBlock           ;           if ( FreeBlock ) then
        CMP #TRUE               ;           begin
        BEQ WR16                ;
        JMP WR_ELSE_NF          ;
        JSR NEWBLOCK            ;             NewBlock(ActBlockP) ;
        LDA #00H                ;
        STA No                  ;             No := 0 ;
```

```
                BNE WR7                 ;
                LDA ActBlockP+1         ;
                CMP #NIL                ;
                BEQ WR_ENDIF8           ;                 begin
WR7:            LDA ActBlockP           ;
                STA TempP               ;
                LDA ActBlockP+1         ;                   TempP := ActBlockP ;
                STA TempP+1             ;
                LDA No                  ;
                PHA                     ;
                LDY #00H                ;
                LDA [SizeP],Y           ;
                STA No                  ;
                CLC                     ;
                LDA No                  ;
                ADC No                  ;
                TAY                     ;
                LDA ActBlockP           ;
                STA [IndexBlockP],Y     ;                   IndexBlockP^.IndexBlock[Size] :=
                INY                     ;                                 ActBlockP ;
                LDA ActBlockP+1         ;
                STA [IndexBlockP],Y     ;
                PLA                     ;
                STA No                  ;
                LDY #00H                ;
                LDA [SizeP],Y           ;
                CLC                     ;
                ADC #01H                ;                   SizeP^.Size := SizeP^.Size + 1 ;
                STA [SizeP],Y           ;
                LDY #01H                ;                   y := 1 ;
WR_ENDIF8:      JMP WR_UNTIL3           ;                 end ;
WR_ELSE_NF:                             ;               end
                DEY                     ;               else  begin
                LDA #ETX                ;                 y := y - 1 ;
                STA CHAR                ;                 Char := ETX ;  end ;
WR_ENDIF7:                              ;             end ;
WR_ENDIF4:                              ;           end ;
WR_UNTIL3:      LDA CHAR                ;         until (Char = ETX) or (ActBlockP = nil) ;
                CMP #ETX                ;
                BEQ WR9                 ;
                LDA ActBlockP           ;
                CMP #NIL                ;
                BNE WR8                 ;
                LDA ACtBlockP+1         ;
                CMP #NIL                ;
                BEQ WR9                 ;
WR8:            JMP WR_REPEAT3          ;
WR9:                                    ;
WR_IF6:         LDA CHAR                ;       if (Char = ETX) then
                CMP #ETX                ;
                BNE WR_ENDIFWR          ;
                LDA #1AH                ;         TempP^.Data[ y ] := eof ;
                STA [TempP],Y           ;
WR_ENDIFWR:     LDA FreeBlock           ;     end ;
                CMP #FALSE              ;     if (not FreeBlock) then
                BNE WR15                ;
                JMP Error               ;       Error ;
WR15:           PLA                     ;
                TAY                     ;
                PLA                     ;   end ;
                STA No                  ;
                JMP WAITCOMMAND         ;
```

```
; NAME      : DOWNFETCH
; FUNCTION : "DownFetch" takes the content of a specified (an open) file and
;            sends it down to the target, usually a Microflex.
; INPUT     : None
; OUTPUT    : None
; REGISTER : A, Y
; ---------------------------------------------------------------------------
DOWNFETCH:                          ; procedure DownFetch ;
            LDA No                  ; begin
            PHA                     ;
            TYA                     ;
            PHA                     ;
DO_IF2:     LDA OpenFileP           ;   if (OpenFile) then
            CMP #NIL                ;
            BNE DO2                 ;
            LDA OpenFileP+1         ;
            CMP #NIL                ;   begin
            BNE DO2                 ;
            JMP DO_ENDIF2           ;
DO2:        LDA #1EH                ;
            STA CHAR                ;     Char := eoln ;
            LDY #00H                ;     y := 0 ;
            LDA [SizeP],Y           ;
            CLC                     ;
            ADC [SizeP],Y           ;
            STA Index               ;     Index := SizeP^.Size ;
DO_REPEAT1:                         ;     repeat
            LDA [IndexBlockP],Y     ;
            STA ActBlockP           ;       ActBlockP := IndexBlockP^.Index[y] ;
            STA TempP               ;
            INY                     ;
            LDA [IndexBlockP],Y     ;
            STA ActBlockP+1         ;
            STA TempP+1             ;       TempP := ActBlockP ;
            TYA                     ;
            PHA                     ;
            LDA #00H                ;
            STA No                  ;       No := 0 ;
            LDY #01H                ;       y := 1 ;
DO_REPEAT3:                         ;       repeat
DO_IF5:     LDA CHAR                ;         if (Char = eoln) then
            CMP #1EH                ;
            BNE DO_IF3              ;
DO_REPEAT2: JSR GETCHAR             ;           repeat
            LDA CHAR                ;             GetChar( Char ) ;
            CMP #11H                ;             (* 11H = xon *)
            BEQ DO_IF3              ;
            CMP #07H                ;             (* 07H = bell *)
DO_UNTIL2:  BNE DO_REPEAT2          ;           until (Char = xon) or (Char = bell)
DO_IF3:                             ;         end ;
            CMP #07H                ;         if (Char <> bell) then
            BEQ DO_ELSE3            ;         begin
            LDA [TempP],Y           ;
            STA CHAR                ;           Char := TempP^.Data[ y ] ;
DO_WHILE:   CMP #1EH                ;           while (Char <> eoln) and
            BEQ DO_W_END            ;                 (Char <> eof) and
            CMP #1AH                ;                 (y,No < BlockSize) do
            BEQ DO_W_END            ;
            CPY BlockSize           ;
            BNE DO3                 ;
            LDA No                  ;
            CMP BlockSize+1         ;
            BEQ DO_W_END            ;           begin
DO3:        LDA CHAR                ;             if (Char = 00H) then
```

```
                STA CHAR          ;
DO_SEND:        JSR SENDCHAR      ;              SendChar( Char ) ;
                INY               ;              y := y + 1 ;
DO_IF1:         CPY #00H          ;              if (y = 00H) then
                BNE DO_ENDIF1     ;              begin
                LDA ActBlockP+1   ;
                CLC               ;
                ADC #01H          ;                TempP := ActBlockP + 0100H ;
                STA TempP+1       ;
                INC No            ;                No := No + 1 ;
DO_ENDIF1:      LDA [TempP],Y     ;              end ;
                STA CHAR          ;              Char := TempP^.Data[ y ] ;
                JMP DO_WHILE      ;            end ; (* while *)
DO_W_END:       LDA CHAR          ;            if (Char = eoln) or
                CMP #1EH          ;               (Char = eof) then
                BEQ DO6           ;
                CMP #1AH          ;
                BEQ DO6           ;
                JMP DO_ENDIF3     ;
DO6:            LDA #0DH          ;
                JSR SENDA         ;              SendA( cr ) ;
                INY               ;              y := y + 1 ;
DO_IF4:         CPY #00H          ;              if (y = 00H) then
                BNE DO_ENDIF4     ;              begin
                LDA ActBlockP+1   ;
                CLC               ;
                ADC #01H          ;                TempP := ActBlockP + 0100H ;
                STA TempP+1       ;
                INC No            ;                No := No + 1 ;
DO_ENDIF4:      JMP DO_ENDIF3     ;            end
DO_ELSE3:       JMP Error         ;            else
DO_ENDIF3:                        ;              Error ;
DO_UNTIL3:      LDA CHAR          ;          until (Char = eof) or
                CMP #1AH          ;                (y,No = BlockSize) ;
                BEQ DO5           ;
                CPY BlockSize     ;
                BNE DO7           ;
                LDA No            ;
                CMP BlockSize+1   ;
                BEQ DO5           ;
DO7:            JMP DO_REPEAT3    ;
DO5:            PLA               ;
                TAY               ;
                INY               ;          y := y + 1 ;
DO_UNTIL1:      CMP #1AH          ;        until (Char = eof) or (y =Index) ;
                BEQ DO4           ;
                CPY Index         ;
                BEQ DO4           ;
                JMP DO_REPEAT1    ;
DO4:            LDA #04H          ;
                JSR SENDA         ;    SendChar( eot ) ;
                LDA #20H          ;
                JSR SENDA         ;    SendChar(blank) ;
                LDA #0DH          ;
                JSR SENDA         ;    SendChar( cr ) ;
                LDA #NIL          ;
                STA OpenFileP     ;    OpenFileP := nil ;
                STA OpenFileP+1   ;
DO_ENDIF2:      PLA               ; end ;
                TAY               ;
                PLA               ;
                STA No            ;
                JMP WAITCOMMAND   ;
```

```
; NAME      : SECUREFETCH
; FUNCTION  : Same as DOWNFETCH but here the content of the transmitted file
;             is transmitted row by row. Each row is transfered in a frame
;             ended by a chechsum wich tells the reciver if the frame is
;             damaged on its way from the sender. If that's the case the
;             tells the sender that by sending a NAK (Negative AcKnowledgement)
;             and the sender will then retransmitt the actual frame.
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A, X, Y
; ---------------------------------------------------------------------------
SECUREFETCH:                        ; procedure SecureFetch ;
             LDA No                 ; begin
             PHA                    ;
             TYA                    ;
             PHA                    ;
             TXA                    ;
             PHA                    ;
SE_IF2:      LDA OpenFileP          ;   if OpenFile then
             CMP #NIL               ;   begin
             BNE SE2                ;
             LDA OpenFileP+1        ;
             CMP #NIL               ;
             BNE SE2                ;
             JMP SE_ENDIF2          ;
SE2:         LDA #1EH               ;
             STA CHAR               ;     CHAR := eoln ;
             LDA #FALSE             ;
             STA BlockChange        ;
             LDY #00H               ;     y1 := 0 ;
             LDA [SizeP],Y          ;     Index := Size ;
             CLC                    ;
             ADC [SizeP],Y          ;
             STA Index              ;
SE_REPEAT1:  LDA [IndexBlockP],Y    ;     repeat
             STA ActBlockP          ;       ActBlockP := IndexBlockP[y] ;
             STA TempP              ;       TempP := ActBlockP ;
             INY                    ;
             LDA [IndexBlockP],Y    ;
             STA ActBlockP+1        ;
             STA TempP+1            ;
             TYA                    ;
             PHA                    ;
             LDA #00H               ;       No := 0 ;
             STA No                 ;
             LDY #01H               ;       y := 1 ;
             LDA BlockChange        ;
             CMP #TRUE              ;
             BEQ SEC                ;
SE_REPEAT3:                         ;       repeat
SE_IF10:     LDA CHAR               ;         if (Char = eoln) then
             CMP #1EH               ;         begin
             BNE SE_IF3             ;           repeat
SE_REPEAT2:  JSR GETCHAR            ;             GetChar(Char) ;
             LDA CHAR               ;
             CMP #11H               ;             (* 11H = xon *)
             BEQ SE_IF3             ;
             CMP #07H               ;             (* 07H = bell *)
SE_UNTIL2:   BNE SE_REPEAT2         ;           until (Char = xon) or (Char = bell);
SE_IF3:                             ;         end ;
             CMP #07H               ;         if (Char <> bell) then
             BNE SE_REPEAT4         ;
             JMP SE_ELSE3           ;         begin
SE_REPEAT4:                         ;           repeat
```

```
            STA FBIndex         ;              FBIndex := 0 ;
SEC:        LDA [TempP],Y       ;
            STA CHAR            ;              Char := TempP^.Data[y] ;
SE_WHILE:   CMP #1EH            ;              while (Char <> eoln) and
            BEQ SE_W_END        ;                    (Char <> eof) and
            CMP #1AH            ;                    (y,No < BlockSize) do
            BEQ SE_W_END        ;              begin
            CPY BlockSize       ;
            BNE SE3             ;
            LDA No              ;
            CMP BlockSize+1     ;
            BEQ SE_W_END        ;
SE3:        LDA CHAR            ;                if (Char = 00H) then
            BNE SE_FRAME        ;                  CHAR := 20H ;
            LDA #20H            ;                  (* 20H = blank *)
            STA CHAR            ;
SE_FRAME:   LDX FBIndex         ;
            LDA CHAR            ;
            STA FrameBuff,X     ;                FrameBuff[FBIndex] := Char ;
            CLC                 ;
            ADC CheckSum        ;                CheckSum := CheckSum + CHAR ;
            STA CheckSum        ;
            LDA #00H            ;
            ADC CheckSum        ;
            STA CheckSum        ;
            INY                 ;                y := y + 1 ;
SE_IF1:     CPY #00H            ;                if (y = 00H) then
            BNE SE_ENDIF1       ;                begin
            LDA ActBlockP+1     ;
            CLC                 ;
            ADC #01H            ;                  TempP := ActBlockP + 0100H ;
            STA TempP+1         ;                  No := No + 1 ;
            INC No              ;                end;
SE_ENDIF1:  INC FBIndex         ;                FBIndex := FBIndex + 1 ;
            LDA [TempP],Y       ;
            STA CHAR            ;                Char := TempP^.Data[y] ;
            JMP SE_WHILE        ;              end ; (* while *)
SE_W_END:                       ;
SE_IF5:     LDA FBIndex         ;
            CMP #00H            ;
            BNE SE_ENDIF5       ;
            INY                 ;
SE_IF6:     CPY #00H            ;
            BNE SE_ENDIF6       ;
            LDA ActBlockP+1     ;
            CLC                 ;
            ADC #01H            ;
            STA TempP+1         ;
            INC No              ;
SE_ENDIF6:                      ;
SE_UNTIL4:  LDA CHAR            ;            until (FBIndex > 0) or
            CMP #1AH            ;                  (Char = eof) ;
            BEQ SE_IF7          ;
            LDA FBIndex         ;
            CMP #00H            ;
            BNE SE_IF7          ;
            JMP SE_REPEAT4      ;
SE_ENDIF5:                      ;
SE_IF7:     LDA CHAR            ;            if (Char = eoln) or
            CMP #1EH            ;               (Char = eof) then
            BEQ SE6             ;            begin
            CMP #1AH            ;
            BEQ SE6             ;
            JMP SE_ENDIF3       ;
```

```
                LDA #FALSE          ;
                STA BlockChange     ;
                INY                 ;                y := y + 1 ;
SE_IF4:         CPY #00H            ;                if (y = 00H) then
                BNE SE_ENDIF4       ;                begin
                LDA ActBlockP+1     ;
                CLC                 ;                  TempP := ActBlock + 0100H;
                ADC #01H            ;
                STA TempP+1         ;                  No := No + 1 ;
                INC No              ;
SE_ENDIF4:      LDA FBIndex         ;                end ;
                STA CHAR            ;                Char := FBIndex ;
                JSR CONHEX          ;                ConHex(Char, NoCh1, NoCh2);
                STA NoCh1           ;
                STX NoCh2           ;
                CLC                 ;
                ADC CheckSum        ;                CheckSum := CheckSum + NoCh1 ;
                STA CheckSum        ;
                LDA #00H            ;
                ADC CheckSum        ;
                STA CheckSum        ;
                TXA                 ;
                CLC                 ;
                ADC CheckSum        ;                CheckSum := CheckSum + NoCh2 ;
                STA CheckSum        ;
                LDA #00H            ;
                ADC CheckSum        ;
                STA CheckSum        ;
                STA CHAR            ;                Char := CheckSum ;
                JSR CONHEX          ;                ConHex(Char,Check1,Check2) ;
                STA Check1          ;
                STX Check2          ;
                LDA #00H            ;                NoOfNAK := 0 ;
                STA NoOfNAK         ;
SE_REPEAT5:                         ;                repeat
                JSR SENDFRAME       ;                  SendFrame(ComCh) ;
SE_IF9:         JSR GETCHAR         ;                  GetChar(Char) ;
                LDA EMPTYBUFF       ;
                CMP #TRUE           ;
                BEQ SE_IF9          ;
                LDA CHAR            ;
                CMP #15H            ;                  if (Char = nak) then
                BNE SE_ENDIF9       ;                  begin
SEA:            JSR GETCHAR         ;                    GetChar(Char) ;   (* xon *)
                LDA EMPTYBUFF       ;
                CMP #TRUE           ;
                BEQ SEA             ;
                INC NoOfNAK         ;                    NoOfNAK := NoOfNAK + 1 ;
SE_ENDIF9:                          ;                  end ;
SE_UNTIL5:      LDA CHAR            ;                until (Char = ack) or
                CMP #06H            ;                      (NoOfNAK = 10) ;
                BEQ SE8             ;
                LDA NoOfNAK         ;
                CMP #0AH            ;
                BEQ SE_ELSE3        ;                  GetChar(Char) ;
                JMP SE_REPEAT5      ;              end ;
SE8:            JSR GETCHAR         ;            end
                LDA EMPTYBUFF       ;
                CMP #TRUE           ;
                BEQ SE8             ;
                JMP SE_ENDIF3       ;            else
SE_ELSE3:       JMP Error           ;              Error ;
SE_ENDIF3:                          ;
SE_UNTIL3:      LDA COMCH           ;
```

```
                BEQ SE5                 ;                   (y,No = BlockSize) ;
                CPY BlockSize           ;
                BNE SE7                 ;
                LDA No                  ;
                CMP BlockSize+1         ;
                BEQ SE8                 ;
SE7:            JMP SE_REPEAT3          ;
SE8:            LDA #TRUE               ;
                STA BlockChange         ;
SE5:            PLA                     ;
                TAY                     ;
                INY                     ;       yi := yi + 1 ;
SE_UNTIL1:      CMP #1AH                ;
                BEQ SE4                 ;     until (Char = eof) or (yi = Index);
                CPY Index               ;
                BEQ SE4                 ;   end ;
                JMP SE_REPEAT1          ;
SE4:            LDA #NIL                ;
                STA OpenFileP           ;   OpenFile := nil ;
                STA OpenFileP+1         ;
SE_ENDIF2:      PLA                     ; end ;
                TAX                     ;
                PLA                     ;
                TAY                     ;
                PLA                     ;
                STA No                  ;
                JMP WAITCOMMAND         ;



; ---------------------------------------------------------------------------
; NAME      : DISPOSEFILE
; FUNCTION  : Deletes a closed file specified by "FileName".
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A, X
; ---------------------------------------------------------------------------
DISPOSEFILE:                    ; procedure DisposeFile ;
                TXA             ; begin
                PHA             ;
                JSR READFILENAME    ;   ReadFileName ;
                JSR COMPARENAME     ;   CompareName(EqualName, ActHeadP, No) ;
DI_IF:          LDA EQUALNAME       ;   if (EqualName) then
                EOR #TRUE           ;   begin
                BNE DI_ENDIF        ;
                LDA ActHeadP        ;
                STA NameP           ;     NameP := ActHeadP ;
                LDA ActHeadP+1      ;
                STA NameP+1         ;
                CLC             ;
                LDA ActHeadP        ;
                ADC #14H            ;
                STA SizeP           ;     SizeP := ActHeadP + 20 ;
                LDA ActHeadP+1      ;
                ADC #00H            ;
                STA SizeP+1         ;
                CLC             ;
                LDA ActHeadP        ;
                ADC #16H            ;
                STA IndexBlockP     ;     IndexBlockP := ActHeadP + 22 ;
                LDA ActHeadP+1      ;
                ADC #00H            ;
                STA IndexBlockP+1   ;
```

```
                DEX                     ;
DI_FOR:         CPX #0CH                ;     for x := 1 to (No - 1) do
                BEQ DI_FOR_END          ;
                ASL A                   ;       RotateLeft( A ) ;
                DEX                     ;
                JMP DI_FOR              ;
DI_FOR_END:     EOR Available           ;
                STA Available           ;     Avaliable := Avaliable XOR A ;
                DEC NoOfFiles           ;     NoOfFiles := NoOfFiles - 1 ;
                LDA #FALSE              ;
                STA FullDir             ;     FullDir := FALSE ;
                JSR CLEARBLOCK          ;     ClearBlock(IndexBlockP, Size) ;
DI_ENDIF:       PLA                     ;   end ;
                TAX                     ;
                JMP WAITCOMMAND         ; end ;
```

```
;-------------------------------------------------------------------------
; NAME      : SHOWDIR
; FUNCTION  : Shows the current directory.
; INPUT     : None
; OUTPUT    : None
; REGISTER  : A
;-------------------------------------------------------------------------
;
SHOWDIR:                                ; procedure ShowDir ;
                LDA No                  ;
                PHA                     ; begin
                LDA HeadPoint           ;
                STA ActHeadP            ;   ActHeadP := HeadPoint ;
                STA NameP               ;   NameP := ActHeadP ;
                LDA HeadPoint+1         ;
                STA ActHeadP+1          ;
                STA NameP+1             ;
                CLC                     ;
                LDA ActHeadP            ;
                ADC #14H                ;
                STA SizeP               ;   SizeP := ActHeadP + 20 ;
                LDA ActHeadP+1          ;
                ADC #00H                ;
                STA SizeP+1             ;
                LDA #00H                ;
                STA No                  ;   No := 0 ;
                LDX #01H                ;   x := (0000 0001)B ;
SH_WHILE1:      LDA No                  ;   while (No < NoOfFiles) do
                CMP NoOfFiles           ;   begin
                BNE SH_IF               ;
                JMP SH_W1_END           ;
SH_IF:          TXA                     ;     if (Available AND x <> 0) then
                AND Available           ;     begin
                BNE SH1                 ;
                JMP SH_ENDIF            ;
SH1:            LDY #00H                ;       y := 0 ;
                LDA [NameP],Y           ;
                STA CHAR                ;       Char := NameP^.Name[ y ] ;
SH_WHILE2:      CPY #14H                ;       while ((y < 20) and
                BEQ SH_W2_END           ;              (Char <> blank)) do
                CMP #20H                ;       begin
                BEQ SH_W2_END           ;
                JSR SENDCHAR            ;         SendChar(Char) ;
                INY                     ;         y := y + 1 ;
                LDA [NameP],Y           ;
```

```
SH_W2_END:  LDA #1EH            ;        end ; (* while *)
            STA CHAR            ;
            JSR SENDCHAR        ;        SendChar('eoln') ;
            LDY #00H            ;
            LDA [SizeP],Y       ;
            STA CHAR            ;        Char := SizeP^.Size ;
            TXA                 ;
            PHA                 ;
            JSR CONHEX          ;        ConHex(ConCh,ASCIIData) ;
            STA CHAR            ;
            JSR SENDCHAR        ;        SendChar(ASCIICharHigh) ;
            STX CHAR            ;
            JSR SENDCHAR        ;        SendChar(ASCIICharLow) ;
            PLA                 ;
            TAX                 ;
            INC No              ;        No := No + 1 ;
SH_ENDIF:   TXA                 ;      end ;
            ASL A               ;      RotateLeft(x) ;
            TAX                 ;
            LDA ActHeadP        ;
            CLC                 ;
            ADC HeadSize        ;
            STA ActHeadP        ;      ActHeadP := ActHeadP + HeadSize ;
            STA NameP           ;
            LDA ActHeadP+1      ;
            ADC HeadSize+1      ;
            STA ActHeadP+1      ;
            STA NameP+1         ;      NameP := ActHeadP ;
            LDA ActHeadP        ;
            CLC                 ;
            ADC #14H            ;
            STA SizeP           ;      SizeP := ActHeadP + 20 ;
            LDA ActHeadP+1      ;
            ADC #00H            ;
            STA SizeP+1         ;
            JMP SH_WHILE1       ;
SH_W1_END:  LDA #03H            ;    end ; (* while *)
            STA CHAR            ;
            JSR SENDCHAR        ;    SendChar(EndOfDirectory) ;
            PLA                 ;
            STA No              ;
            JMP WAITCOMMAND     ; end ;
```

```
;-------------------------------------------------------------------------
; NAME      : EXECUTE
; FUNCTION : Sends the word recived on the serialchanel back to MF10
;            to be executed there.
;-------------------------------------------------------------------------
EXECUTE:    JSR EXFO_READ
            JSR EXFO_WRITE
            JMP WAITCOMMAND


;-------------------------------------------------------------------------
; NAME      : FORGETWORD
; FUNCTION : Sends the word recived on the serialchanel back to MF10
;            to be deleted there.
;-------------------------------------------------------------------------
FORGETWORD: JSR EXFO_READ
            LDA #46H            ; F
            JSR SENDA
```

```
                LDA #52H                ; R
                JSR SENDA
                LDA #47H                ; G
                JSR SENDA
                LDA #45H                ; E
                JSR SENDA
                LDA #54H                ; T
                JSR SENDA
                LDA #20H
                JSR SENDA
                JSR EXFO_WRITE
                JMP WAITCOMMAND


; ---------------------------------------------------------------------------
; NAME      : CLEARDIR
; FUNCTION : Emergency routin, must only be used if system has crashed.
;            Deletes all files and returns all blocks to system.
; ---------------------------------------------------------------------------
CLEARDIR:                               ;
                LDA InitFlag            ;
                EOR #0FFH               ;
                STA InitFlag            ;
                JMP Init                ;


; ***************************************************************************
; Interrupt handle routines
; ***************************************************************************

; ---------------------------------------------------------------------------
; NAME      : NMIHANDLER
; FUNCTION : Returns to program when NMI occurs.
; ---------------------------------------------------------------------------
NMIHANDLER:
                PHA
                INC RUN
                LDA #10H
                CMP RUN
                BNE NMIEND
                LDA PORTA
                EOR #02H
                ORA #0DDH
                STA PORTA
                LDA #00H
                STA RUN
NMIEND:         PLA
                RTI



; ---------------------------------------------------------------------------
; NAME      : INTHANDLER
; FUNCTION : Interrupt occurs when a character arrives to the RTREG. This
;            routine takes the character and puts it into a ringbuffer.
;            Ringbuffersize = 128 bytes.
; INPUT     : INB, UTB = Pointer
; OUTPUT    : UTB = Pointer
; REGISTER : A, X
; ---------------------------------------------------------------------------
INTHANDLER:
```

```
            PHA
            TYA
            PHA                 ; Save register
            LDA SCSR
            AND #0EH            ; Check RX-flags
            BEQ RINGBUFF
            LDA #01H
            STA TRANSERR
RINGBUFF:
            LDA RTREG           ; Clears RTREG, SCSR(0-3) and IFR
            LDX INB
            STA BUFF,X
            INX
            TXA
            AND #7FH            ; Buffert size = 128 bytes
            STA INB
            CMP UTB
            BCC LABEL1
            LDA #7FH
            SEC
            SBC INB
            CLC
            ADC UTB
            JMP BUFFSPACE
LABEL1:
            LDA UTB
            SEC
            SBC INB
BUFFSPACE:
            CMP #04H            ; Check free space in ringbuffert.
            BCS LABEL2
            LDA #13H
            JSR SENDA           ; Send XOFF
            LDA #TRUE
            STA XOFF
;            LDA PORTA
;            ORA #20H            ; Free space < 4   ==>   RTS = 1   ==>  MLFEX s
;            STA PORTA
LABEL2:
            PLA
            TAY
            PLA
            TAX
            PLA
            RTI

; ---------------------------------------------------------------------------
; The program must be finished with the label SLUT.
; ---------------------------------------------------------------------------

SLUT:
            JMP SLUT
```