TSR

Development Kit

Ontwikkelpakket voor TSR programma's onder MemMan 2

MSX2
80 kB RAM
3,5 inch, 1DD
MSXDOS 1 of 2

MSI

MemMan

Original and scanned by Bifi
Converted to PDF by HansO, 2002

Wijzigingen in MemMan 2.3 ten opzichte van 2.2

- De functie XTsrCall (61) is toegevoegd. Deze functie werkt identiek aan de functie TsrCall (63), het Tsr-ID wordt echter verwacht in register IX in plaats van BC. Hierdoor komt register BC vrij om als invoerparameter gebruikt te worden. Nadeel is dat deze functie niet via de MemMan hook mag worden aangeroepen, maar alleen direct.
- Door middel van de functie Info (50) kan het adres worden opgevraagd waarop XTsrCall rechtstreeks kan worden aangeroepen.
- De funtie status (31) is verbeterd. De totale hoeveelheid bruikbaar werkgeheugen in de computer wordt nu correct gemeld, ook onder MSX-DOS2.
- De Alloc (10) functie herkent nu ook geheugen dat beschikbaar komt wanneer de DOS2 RAMdisk wordt verwijderd of verkleind! Het maakt daarbij niet meer uit of de RAMdisk wordt aangemaakt voor- of nadat MemMan werd geïnstalleerd.
- De interne stack van MemMan die gebruikt worden om functieaanroepen te verwerken is vergroot tot 240 bytes, in plaats van 160. In de praktijk bleek dat de functiestack van MemMan 2.2 te krap was om geneste "tsrCalls" te verwerken.

Naam: **XTsrCall**
Nummer: **61**
Functie: **Roep het driver-entry van een TSR aan**
In: **IX = ID code van de aan te roepen TSR**
AF,HL,DE,BC worden ongewijzigd doorgeven aan de TSR.
Uit: **AF,HL,BC,DE komen ongewijzigd terug van de TSR.**

Deze functie is een verbeterde versie van de functie TsrCall (63). Omdat met deze functie alle main-registers aan de TSR kunnen worden doorgegeven, verdient het aanbeveling om deze functie te gebruiken in plaats van functie 63.
Opm: Deze functie mag niet worden aangeroepen via de EXTBIO hook, omdat bij een aanroep via EXTBIO het IX-register verminkt wordt. Roep deze functie daarom rechtstreeks aan, of gebruik de MemMan functie-afhandelingsroutine. De adressen waarop deze routines aangeroepen kunnen worden, kunnen via de info functie (50) worden opgevraagd.

Naam: **Info**
Nummer: **50**
Functie: **Geeft informatie over o.a. aanroep-adressen van MemMan functies**
In: **B = Informatie nummer (0..8)**
Uit: **HL = Informatie**

Informatie nummer overzicht. Tussen haakjes staan de equivalente MemMan functie codes:
0 Aanroepadres van FastUse0 (functie 0)
1 Aanroepadres van FastUse1 (functie 1)
2 Aanroepadres van FastUse2 (functie 2)
3 Aanroepadres van TsrCall (functie 63)
4 Aanroepadres van BasicCall
5 Aanroepadres van FastCurSeg (functie 32)
6 Aanroepadres van MemMan, de functie-afhandelingsroutine
7 Versienummer van MemMan, format: Versie #H.L
8 Aanroepadres van XTsrCall (functie 61)

De bovengenoemde functie-adressen mogen door een toepassingsprogramma of TSR rechtstreeks aangeroepen worden. Alle entry adressen liggen gegarandeerd in pagina 3.
De functies worden snel uitgevoerd omdat de MemMan CALL naar de EXTBIO hook vervalt en de functie-codes in registers D en E niet uitgeplozen hoeven worden. Een ander voordeel is dat parameters ook via het register DE doorgegeven kunnen worden, dit is vooral van belang bij de TsrCall en BasicCall functies.
Bijvoorbeeld, de initialisatie routine van een TSR kan de benodigde functieadressen via de INFO functie opvragen en deze vervolgens voor later gebruik in de TSR programmacode opslaan, wat de snelheid van het TSR programma zeer ten goede kan komen.

Een exacte beschrijving van de bovenstaande functies kan gevonden worden bij de MemMan functie waarvan het nummer tussen haakjes is aangegeven.
Houd echter onder de aandacht dat de 'snelle' functies op de volgende punten van de gewone MemMan functies verschillen:
fastUse0-2: Schakelt een segment in in een bepaalde geheugen pagina. Zie de omschrijving bij de memMan 'Use' functies.
tsrCall: Register [DE] wordt ongewijzigd aan de TSR doorgegeven. Dit in tegenstelling tot functie 63 (TsrCall), register DE is dan al bezet om het MemMan functienummer in op te slaan.
xTsrCall: Alle mainregisters (AF,HL,BC,DE) worden ongewijzigd aan de TSR doorgegeven. De TSR-ID code dient in register IX te worden geplaatst.

basicCall: **Heeft geen MemMan functie nummer. (adres opvraagbaar via info)**
Functie: **Aanroepen van een routine in de BASIC ROM.**
In: **IX = Call address in pagina 0 of 1**
AF,HL,BC,DE = dataregisters voor de BASIC-ROM
Uit: **AF,HL,BC,E = dataregisters van de BASIC-ROM**
Interrupts disabled
Via deze functie kunnen TSR's een routine aanroepen die zich in pagina 0 en/of pagina 1 van het BASIC-ROM bevindt. De bios moet al in pagina 0 aangeschakeld zijn. In pagina 1 wordt de BASIC ROM door MemMan aangeschakeld.
Dit is bijvoorbeeld noodzakelijk om de math-pack routines aan te kunnen roepen die in pagina 0 van de BASIC ROM zitten, maar tussendoor ook een aantal routines in pagina 1 aanroepen.
De H.STKE (stack error) hook wordt afgebogen, zodat na een eventueel op getreden BASIC error de interne stacks van MemMan gereset kunnen worden.

fastCurSeg: In register [A] komt geen zinnige waarde terug. De MemMan CurSeg functie (32) geeft aan of het een FSEG/PSEG betreft.

memMan: **Heeft geen MemMan functienummer (adres opvraagbaar via info)**
Functie: **Rechtstreeks aanroepen van een MemMan functie.**
In: **E=MemMan functienummer**
AF,HL,BC = Dataregisters afhankelijk van de aan te roepen functie.
Uit: **AF,HL,BC,DE = dataregisters afhankelijk van de aangeroepen functie.**
Een aanroep van deze routine heeft hetzelfde effect als het aanroepen van een MemMan functie via de EXTBIO hook. Doordat echter de aanroep naar de EXTBIO hook vervalt, worden de overige uitbreidingen die aan deze hook gekoppeld zijn niet aangeroepen. Hierdoor blijft het stack gebruik beperkt en wordt de verwerkingssnelheid verhoogd.

MemMan 2.3 BASIC-statements

Vanaf versie 2.3 bevat MemMan enkele statements en functies die vanuit MSX-BASIC toegepast kunnen worden. Hiertoe is MemMan van een systeem TSR voorzien, met de ID-naam "MST TsrUtils". Deze TSR heeft volgnummer 0 en kan niet door TSR-Kill verwijderd worden. Hieronder volgt een beschrijving van de beschikbare BASIC-instructies. De betekenis van de gebruikte symbolen is als volgt:

[] Wat tussen vierkante haken staat mag worden weggelaten.
<> Omschrijvingen van parameters staan tussen gehoekte haken.
() Ronde haken moeten worden ingetikt, evenals leestekens en comma's.

Commando: **TSR-Load**
Syntax: **CMD TL("<filename>",[T],[A$],[<F>])**
Soort: **Statement**
Voorbeeld: **CMD TL("PB")**
CMD TL("ALARM",T,,A)
Functie: **Laadt een TSR-bestand in het geheugen.**
<filename> = Naam van het TSR-bestand. Onder MSX-DOS2 mag een subdirectory worden opgegeven.
T = Toon de intro-tekst van de TSR.
A$ = Naam van een string-variabele waarin de TSR ID-naam zal worden opgeslagen.
<F> = Variabele waarin een foutcode wordt opgeslagen. Indien deze variabele wordt weggelaten, zal in geval van laadfouten een standaard BASIC-foutmelding worden gegeven. De foutcodes zijn als volgt:
0: TSR met succes ingeladen
1: Installatie door de TSR afgebroken
2: Structuurfout in TSR bestand
3: TSR-tabel vol
4: Hook-tabel vol
5: Geen vrij MemMan segment
6: Te weinig vrij BASIC werkgeheugen

Commando: **TSR-Kill**
Syntax: **CMD TK("<TSR ID-naam>")**
Soort: **Statement**
Voorbeeld: **CMD TK("MJV printbuf")**
Functie: **Verwijdert een TSR uit het geheugen.**
<TSR ID-Naam> = Identificatienaam van de te verwijderen TSR. Deze naam kan worden opgevraagd door middel van TSR-View of Find-TSR.

Commando: **TSR-View**
Syntax: **CMD TV**
Soort: **Statement**
Functie: **Toont een overzicht van de ID-namen van alle actieve TSR's.**

Commando: **Find-TSR name**
Syntax: **ATTR$ FT("<TSR ID-naam>")**
Soort: **Functie**
Voorbeeld: **IF ATTR$ FT("CAPS") THEN CMD TK("CAPS")**
Functie: **Levert de waarde -1 indien de opgegeven TSR is geïnstalleerd, anders 0.**
<TSR ID-Naam> = Identificatienaam van de TSR.

Commando: **Find-TSR number**
Syntax: **ATTR$ FT(<TSR volgnummer>)**
Soort: **Functie**
Voorbeeld: **N$ = ATTR$ FT(0)**
Functie: **Levert de ID-naam van de TSR met het opgegeven volgnummer. Indien het volgnummer groter is dan het aantal actieve TSR's, dan wordt een lege string met lengte 0 teruggeven.**

Patches MemMan 2.3 (versie 2.30 naar 2.31)

De eerste fout die gevonden werd betrof TL.COM. Het bleek niet mogelijk TSR's te laden onder MSXDOS 1. De patch werd gepubliceerd in MSX Computer Magazine nummer 49 en MSX Club Magazine nummer 38:

```
10 'Update TL.COM van versie 1.30 naar 1.31                                    0
20 OPEN "TL.COM" AS #1 LEN=1: FIELD #1,1 AS A$: GET #1,1637: IF A$<>"0" THEN PRI
NT "Niet TL.COM versie 1.30": CLOSE: END                                      110
30 READ A: IF A=-2 THEN CLOSE: PRINT "TL.COM versie 1.31 is gemaakt": END ELSE I
F A=-1 THEN READ B,A                                                          228
40 LSET A$=CHR$(A): PUT #1,B: B=B+1: GOTO 30                                  155
50 DATA -1,70,68,10,-1,497,183,32,238,-1,1637,49,-1,2373,33,128,0,84,94,25,35,11
6,195,253,6,-2                                                                212
```

De tweede fout was ernstiger. TSR's die geheugen schakelden in pagina 2 gingen de mist in, gelukkig was dat op dat moment alleen de printerbuffer PB.TSR, en ging het alleen fout als er onder DOS 1 gewerkt werd en CMD TL gebruikt werd om de TSR te laden. De patch werd gepubliceerd in MSX Computer Magazine nummer 52, en waarschijnlijk ook in MSX Club Magazine nummer 40, die op het moment dat dit geschreven wordt nog net niet uit is.

```
10 PRINT "Update MEMMAN.BIN en/of MEMMAN.COM van versie 2.30 naar versie 2.31":
PRINT "1 MEMMAN.BIN patchen": PRINT "2 MEMMAN.COM patchen": PRINT "3 beide patch
en": PRINT "4 stoppen": PRINT "Keuze:";: I=VAL(INPUT$(1)): PRINT I$: PRINT     125
20 IF (IAND1)=1 THEN RESTORE 80: F$="MEMMAN.BIN": C=174: GOSUB 50              241
30 IF (IAND2)=2 THEN RESTORE 90: F$="MEMMAN.COM": C=267: GOSUB 50              225
40 IF I=4 THEN END ELSE PRINT: GOTO 10                                         181
50 OPEN F$ AS#1 LEN=1: FIELD #1,1 AS A$: GET #1,C: B$=A$: GET #1,C+1: B$=B$+A$:
IF B$="31" THEN PRINT F$;" is al gepatched": CLOSE: RETURN ELSE IF B$<>"30" THEN
 PRINT F$;" is niet versie 2.30": CLOSE: RETURN                                100
60 READ A: IF A=-2 THEN CLOSE: PRINT F$;" versie 2.31 is aangemaakt": RETURN ELS
E IF A=-1 THEN READ B,A                                                        113
70 LSET A$=CHR$(A): PUT #1,B: B=B+1: GOTO 60                                    31
80 DATA -1,35,59,131,-1,322,0,205,181,134,17,30,77,205,202,255,201,-1,175,49,-2    44
90 DATA -1,128,64,131,-1,415,0,205,186,134,17,30,77,205,202,255,201,-1,268,49,-2    54
```

Inhoudsopgave

Inleiding

Ten eerste danken wij u hartelijk voor het aanschaffen van de MST TSR ontwikkeldisk. Met dit pakket zult u in staat zijn Terminate and Stay Resident programma's volgens de MemMan-standaard te programmeren.

MemMan 2 vormt een aanvulling op de MSX-standaard, voor wat betreft het geheugenbeheer. Bovendien regelt MemMan de installatie, het aanroepen en het verwijderen van TSR programma's. Daardoor is het in principe mogelijk een onbeperkt aantal TSR's probleemloos samen te laten werken.
TSR programma's worden in door MemMan beheerde geheugensegmenten geplaatst, er worden – indien mogelijk – meerdere TSR's in één segment geplaatst.

Het grootste gedeelte van de MemMan code bevindt zich in een geheugensegment dat zich buiten het gewone DOS en Basic geheugen bevindt. Slechts enkele routines worden in geheugenpagina 3 geïnstalleerd, om DOS geheugen te besparen. Ook de TSR's worden in aparte segmenten geplaatst, indien mogelijk wordt één segment door meerdere TSR's gedeeld. Ook de ruimte die over is in het MemMan-segment kan door TSR's benut worden.

Het ombuigen van de hooks in het systeemgeheugen wordt geheel door MemMan afgehandeld, de TSR-programmeur hoeft alleen het adres van de af te buigen hooks op te geven. Ook de manier waarop toepassingsprogrammatuur de TSR's aanroept is gestandaardiseerd.

TSR's kunnen worden ontwikkeld met iedere assembler die relocatable files kan aanmaken. Deze bestanden worden vervolgens door MST's eigen linker – LinkTsr – samengevoegd tot een TSR-bestand.

Ten slotte wijzen we nog even op het feit dat MemMan vrij verspreid mag worden en zelfs meegeleverd mag worden met een commercieel pakket. We hopen zo een brede ondersteuning van de MemMan standaard te bewerkstelligen. Alleen deze handleiding en de diverse ontwikkel-tools op de disk mogen niet verspreid worden. Zie het bestandenoverzicht elders in deze handleiding.

Het MemMan ontwikkelteam

Bestandenoverzicht MST TSR ontwikkeldisk:

Bestandsnaam	PD	Omschrijving
MEMMAN.COM	Ja	MemMan versie 2.2, inladen vanuit MSX-DOS.
MEMMAN.BIN	Ja	MemMan versie 2.2, inladen vanuit MSX Disk Basic
TL.COM	Ja	TsrLoad, versie 1.20
TK.COM	Ja	TsrKill, versie 1.20
TV.COM	Ja	TsrView, versie 1.20
LT.COM	Nee	LinkTsr, versie 1.00
MM2INTRO.TXT	Ja	Gebruikershandleiding MemMan 2.2
MM22SPEC.TXT	Ja	Functieomschrijving MemMan 2.2
TSRFRAME.GEN	Nee	Raamwerk van een TSR-listing, voor de GEN80 assembler
TSRFRAME.MAC	Nee	Raamwerk van een TSR-listing, voor de M80 assembler
GTSR.BAT	Nee	Batchfile voor het assembleren/linken van een TSR, GEN80 versie
MTSR.BAT	Nee	Batchfile voor het assembleren/linken van een TSR, M80 versie
PB.TSR	Ja	Printerbuffer
PRINT.COM	Ja	PB-toepassingsprogramma

Nota bene: In de kolom 'PD' staat aangegeven of het bestand al dan niet Public Domain is.

MST's MemMan 2, de MSX Memory Manager

Begin 1990 riep MSX Computer Magazine voor het eerst de beste MSX programmeurs van Nederland bij elkaar met de bedoeling de MSX wereld nieuw leven in te blazen. De programmeursgroep maakte kennis en er werden ideeen uitgewisseld. Er bleek behoefte aan een Memory Manager, een programma dat het geheugen van de MSX beheert.

Met de Memory Manager worden twee doelen nagestreefd:

1) Het zoeken en gebruiken van geheugen wordt eenvoudiger. Het zoeken wordt door MemMan gedaan terwijl het gebruik van geheugen zoveel mogelijk wordt losgekoppeld van de configuratie: 'oude' uitbreidingen, een, twee of meer mappers, MemMan heeft er geen moeite mee.
2) Het wordt mogelijk meerdere programma's tegelijkertijd in het geheugen te laden zonder dat ze elkaar in de weg zitten. Hierbij wordt gedacht aan ramdisk's, printerbuffers en op de achtergrond werkende programma's.

Met versie 1 van MemMan – geïntroduceerd op 9 september 1990 – is de eerste doelstelling bereikt. Nu is de tweede doelstelling ook bereikt.
MemMan versie 2 kan meerdere programma's 'ergens' in het geheugen laden laten werken, zonder dat ze van elkaar last hebben. Op andere computer merken was deze techniek al langer bekend. Dergelijke programma's worden daar TSR's genoemd, vandaar ook hier: Terminate and Stay Resident programma's.

Hopelijk zullen nog meer programma's van MemMan gebruik gaan maken en bestaande programma's voor MemMan worden aangepast. Een direct voordeel is dat het programma dan ook direct met bijvoorbeeld 64 kB modules en zelfs met meerdere memory mappers kan werken, iets dat de meeste bestaande programma's niet of niet goed doen.

MemMan versie 2 zal net als de eerste versie als Public Domain de wereld in gestuurd worden. Dat wil zeggen dat iedereen vrij van MemMan gebruik mag maken. Het is zelfs toegestaan MemMan als onderdeel van een commercieel pakket te verkopen. Alleen zo kan het programma uitgroeien tot een aanvulling op de MSX standaard. Er zullen twee pakketten uitgebracht worden. De eerst is voor de gebruiker van MemMan. Dit pakket zal MemMan en een aantal tools voor de TSR's bevatten. Het tweede pakket bevat ontwikkel tools en technische documentatie over het programmeren van TSR's. Dit laatste pakket is geen Public Domain.

Aan MemMan werkten en dachten mee:

Ramon van der Winkel
Ries Vriend
Robbert Wethmar
Paul te Bokkel
Markus The
en een aantal anderen die met hun opbouwende kritiek MemMan hielpen worden tot wat het is.

Het configureren

MemMan is er in twee versies: een .BIN en een .COM file. Het zal duidelijk zijn dat de .BIN versie vanuit BASIC met een BLOAD"MEMMAN.BIN",R instructie geladen kan worden, terwijl de .COM vanuit MSXDOS gestart kan worden door simpelweg MEMMAN in te tikken. Beide versies keren na het laden – via een zogenaamde warm boot – automatisch terug naar BASIC. Als de .COM versie vanuit MSXDOS opgestart wordt, dan kan een commandline mee gegeven worden. Dit zijn commando's welke uitgevoerd zullen worden alsof ze ingetikt zijn. Een Return kan met het @ teken worden opgegeven. Er kunnen meerdere @ tekens in de commandline worden opgegeven, zodat meerdere commando's na elkaar uitgevoerd kunnen worden.

Bijvoorbeeld:

```
A>MEMMAN _SYSTEM@TL CAPS@
```

Na het opstarten van MEMMAN zal naar MSXDOS teruggekeerd worden en de TSR "CAPS" ingeladen worden.

Met behulp van CFGMMAN is het mogelijk een aantal instellingen van MemMan en een default commandline op te geven. CFGMMAN kan zowel de .COM als de .BIN versie configureren. Met betrekking tot de TSR's kunnen de volgende instellingen veranderd worden:

- Default command line
  Hier kan de standaard commando-regel ingevoerd worden. Deze commando-regel wordt uitgevoerd nadat MemMan geïnstalleerd is. Na het laden van de .BIN versie van MemMan wordt altijd deze standaard commando-regel uitgevoerd. De standaard commando-regel wordt niet uitgevoerd indien MemMan vanuit DOS opgestart wordt met een commando regel als argument. Na foutmeldingen van MemMan wordt geen commando regel uitgevoerd.

- Heap grootte
  Sommige TSR programma's hebben extra geheugen nodig in geheugen pagina 3, waar ze normaal gesproken geen toegang toe hebben. De heap is een stuk geheugen in pagina 3 dat wel voor TSR's toegankelijk is. Wanneer een TSR meldt dat er te weinig heap geheugen beschikbaar is dient deze waarde verhoogd te worden. Meestal zal het toevoegen van 100 extra bytes heap-geheugen de problemen uit de wereld helpen. Wanneer een TSR meer heap-ruimte nodig heeft dient de handleiding dat te vermelden.
  Elke verandering van de heap-grootte heeft slechts effect na het opnieuw laden van MemMan.

- Maximum aantal TSR's dat tegelijk aanwezig kan zijn
  Het aantal TSR's dat onder MemMan 2 geladen kan worden is beperkt. Wanneer de TSR Loader (TL) de melding 'TSR Table Full' geeft dient deze waarde verhoogd te worden.

- Maximum aantal hooks dat tegelijk afgebogen kan zijn

Het aantal hooks dat door alle in het geheugen aanwezige TSR's kan worden afgebogen is beperkt. Wanneer de TSR Loader de melding 'Hook Table Full' geeft dient deze waarde verhoogd te worden.

- Recursiediepte
  Wanneer TSR's elkaar of zichzelf te vaak aanroepen zal het systeem op een gegeven moment vastlopen. Door de maximale recursiediepte te verhogen kunnen deze problemen voorkomen worden. TSR's die zichzelf aanroepen dienen dat – met de benodigde recursiediepte – te vermelden in de handleiding.

Het Installeren

Om MemMem vanuit MSX-DOS te laden is het intikken van 'MEMMAN' achter de 'A:>'-prompt voldoende. Vanuit BASIC dient het commando BLOAD "MEMMAN.BIN",R ingevoerd te worden. Na de installatie van MemMan wordt in beide gevallen BASIC gestart, waarna de standaard commando regel uitgevoerd wordt, zoals opgegeven in het configuratie programma CFGMMAN. De standaard commando-regel wordt niet uitgevoerd indien MemMan vanuit DOS gestart wordt met een vervangende commando-regel als argument.

Versie 2 van MemMan neemt behalve een stuk BASIC-geheugen ook een 16 kB segment in beslag. Hierdoor blijft er onder BASIC en MSXDOS zoveel mogelijk geheugen beschikbaar. De ruimte die in het 16 kB segment over is wordt indien mogelijk gebruikt om TSR's in onder te brengen. De hoeveelheid BASIC-geheugen die MemMan gebruikt kan beïnvloed worden door middel van het configuratieprogramma CFGMMAN.

Wanneer MemMan onder DOS2 geïnstalleerd wordt blijven alle segmenten – behalve die ene die MemMan zelf nodig heeft – ook voor DOS2 beschikbaar. Het is dus zonder meer mogelijk eerst MemMan te installeren en daarna een DOS2 RAMdisk. Deze volgorde biedt het voordeel dat MemMan het geheugen dat na het verkleinen van de RAMdisk vrijkomt weer aan MemMan toepassingen kan toekennen. Als de DOS2 RAMdisk eerder geïnstalleerd wordt dan MemMan zijn de door de RAMdisk in beslag genomen segmenten onbereikbaar voor MemMan.
Alvorens zichzelf in het RAM te nestelen controleert MemMan natuurlijk of er al een versie van MemMan aanwezig is. In dat laatste geval verschijnen de info-regels, aangevuld met de mededeling dat MemMan reeds geïnstalleerd is. Verder gebeurt er niets. De commandoregel wordt gewoon uitgevoerd.

Terminate and Stay Resident programma's

Gewoonlijk zal een programma na uitvoering niet in het geheugen achterblijven. Programma's die dat wel doen worden aangeduid met de afkorting TSR: Terminate and Stay Resident. Voorbeelden van dergelijke programma's zijn: printerbuffers en RAMdisks. Maar ook andere toepassingen, zoals een rekenmachine of een kalender die met een enkele toetsdruk opgeroepen kan worden zijn denkbaar.
In het verleden zijn TSR's voor de MSX een tamelijk zeldzaam verschijnsel geweest. Het probleem was namelijk dat het geheugen dat de TSR gebruikt ook door andere programma's gebruikt kan worden. Er zijn in een standaard MSX machine geen mogelijkheden om een stuk geheugen voor een TSR te reserveren. Dit probleem wordt door MemMan uit de wereld geholpen. MemMan beheert het geheugen en zorgt er voor dat er geen geheugenconflicten optreden.
Dankzij MemMan is het mogelijk meerdere TSR's tegelijk in het geheugen te hebben, waarbij elke TSR maximaal 16 kB groot kan zijn. Op de standaard MSX is het laden van meer dan één TSR al lastig en alleen mogelijk als de TSR niet al te groot is. Met de invoering van MemMan 2 krijgt de MSX betere TSR mogelijkheden dan de alom gewaardeerde PC. Bovendien doen ze niet onder voor de 'Desktop Accesoires' zoals die op de Macintosh en de Atari ST gebruikt worden.
Bij MemMan worden twee eenvoudige voorbeeld TSR's geleverd. Ze doen weinig zinvols, maar demonstreren wel degelijk de kracht van Terminate and Stay Resident programma's. De voorbeelden zijn CAPS.TSR en COLOR.TSR. De eerste laat het Caps lampje knipperen, de tweede maakt dat het Basic commando CMD COLOR het scherm inverteert. In de toekomst zullen er echter meer en meer TSR's verschijnen, met mogelijkheden waar de MSX gebruiker tot voor kort alleen maar van kon dromen.

TSR's laden

TSR programma's zijn te herkennen aan de extensie van de bestandsnaam: ze eindigen op .TSR. Deze files bevatten naast de eigenlijke programmacode ook alle informatie die nodig is om de TSR goed in het geheugen te installeren.
Om bijvoorbeeld de demonstratie TSR 'CAPS' – die overigens niets anders doet dan het Caps lampje laten knipperen – te laden moet ingetikt worden:

```
TL CAPS
```

TL staat voor TSR-Load, het is het programma dat de TSR laadt en in het geheugen plaatst. Helaas werkt TSR-Load op dit moment alleen onder MSX-DOS. Vanuit Basic is het nog niet mogelijk TSR programma's te laden.
Zodra TL de TSR in het geheugen geïnstalleerd heeft zal het programma actief worden. In bovenstaand voorbeeld wil dat zeggen dat het CAPS lampje zal gaan knipperen en bij iedere toetsaanslag het cassetterelais aan of uit geschakeld wordt.
TL is een slim programma. Zolang er in het MemMan segment nog ruimte is voor TSR's zullen ze daar geplaatst worden. Alleen als dat ook echt nodig is wordt een nieuw segment gebruikt, dat voor overige toepassingen dan onbereikbaar gemaakt wordt. Dat gebeurt bijvoorbeeld als er een uitzonderlijk groot TSR programma wordt geladen. Wanneer er vervolgens weer een kleinere wordt geladen zal TL eerst alle bestaande TSR segmenten aflopen om te

zien of er ergens nog ruimte is. De volgorde waarin de TSR's geladen worden zal dan ook geen invloed hebben op het geheugengebruik.

TSR's bekijken

Het is ten aller tijde mogelijk te kijken welke TSR's er op dit moment in het geheugen actief zijn. Daartoe bevat het MemMan pakket de utility TV, TSR-View. Het gebruik is de eenvoud zelf: gewoon achter de DOS prompt intikken:

```
TV
```

Er zal een overzicht verschijnen van de op dit moment actieve TSR's, compleet met hun volledige naam. Deze naam moet voor iedere TSR uniek zijn, en zal dan ook vrijwel altijd de initialen van de programmeur bevatten. Deze naam is dus een andere dan de bestandsnaam!
Het is deze volledige naam – het TSR ID – die nodig is als een TSR uit het geheugen verwijderd moet worden. Ook programma's die direct met TSR's samenwerken kunnen deze naam gebruiken om te zien of een TSR in het geheugen aanwezig is.

TSR's verwijderen

Zoals gezegd is het ook mogelijk TSR programma's weer uit het geheugen te verwijderen. Het benodigde programma heet TK, TSR-Kill. TK zorgt er voor dat een TSR netjes verwijderd wordt. Alle andere TSR's blijven vlekkeloos doorwerken, als de TSR als enige in een segment stond wordt dat segment weer vrijgegeven voor gebruik door overige toepassingen.
Om bijvoorbeeld het het Caps lampje weer normaal te laten werken en het knipperen uit te schakelen is het verwijderen van de betrokken TSR voldoende. Daartoe tikt u:

```
TK "MJVcapsblink"
```

Waarbij de volledige naam van de TSR tussen aanhalingstekens opgegeven dient te worden.
TSR-Kill kan behalve geheugen weer vrijmaken voor gebruik ook gebruikt worden om vastgelopen TSR's uit het geheugen te verwijderen. Een TSR die om welke reden dan ook niet meer vlekkeloos functioneert zal met behulp van TK meestal nog wel verwijderd kunnen worden. Vervolgens kan de TSR met TL weer geladen worden, op dezelfde manier als het opnieuw starten van gewone programma's nog wel eens wil helpen geldt dat ook voor TSR's.

MEMMAN versie 2.2 – specificaties

Dit hoofdstuk bevat alle noodzakelijke informatie om toepassingsprogramma's voor MemMan 2 te kunnen schrijven. Voor meer informatie over het programmeren en aanroepen van TSR's verwijzen we naar de betreffende hoofdstukken.

Wijzigingen ten opzichte van MemMan versie 2.0:

Het verschijnen van de MSX Turbo R machine in Nederland heeft nogal wat stof doen opwaaien, bij de MemMan programmeurs. Net na het uitbrengen van MemMan 2.0 bleek het gebruik van MemMan op de Turbo-R nogal wat problemen op te leveren, vooral bij gebruik van de Kanji schermmodes.

Deze problemen zijn deels door ontstaan doordat de Kanji routines in de Turbo-R de extended BIOS hook op adres &HFFCA afbuigen op een manier die niet door de MemMan programmeurs was voorzien. De Turbo-R verwacht namelijk dat iedere applicatie die zich aan de extended BIOS hook bevindt, door middel van een zogenaamde 'interslot call' aangeroepen wordt, terwijl MemMan juist via de veel snellere "jump" instruktie werkte. Dit had tot gevolg dat de Turbo-R de extended BIOS hook – waaraan MemMan gekoppeld is – niet goed afboog en vastliep. Nadat dit was opgelost bleek er nog een structurele fout te zijn gemaakt.

Wanneer MemMan 2.0 wordt geladen en daarna de Kanji driver middels een 'CALL KANJI' instruktie geactiveerd wordt, wordt na een aanroep van de extended BIOS hook de Kanji-ROM aangeschakeld in geheugenpagina 1. Wanneer de aanroep voor MemMan bestemd blijkt te zijn, wordt vervolgens MemMan aangeroepen. Op dat moment is de Kanji-ROM nog steeds aktief in pagina 1. Wanneer de MemMan gereed is, zal de interslot-call routine de slots terugschakelen in de stand die bekend was ten tijde van de hook-aanroep.

In sommige gevallen is dat herstellen van de slotstand echter niet gewenst. De MemMan 'use' funkties bijvoorbeeld, zijn namelijk speciaal bedoeld om de slot-stand te kunnen veranderen. Het effect van de 'use' funkties wordt echter direct weer ongedaan gemaakt door het terugschakelen van de slotstand door de interslot-call routine.

De 'use' funkties van MemMan kunnen dus beter niet via de extended BIOS hook aangeroepen kunnen worden. Ook 'restore slot' (funktie 41) lijdt aan hetzelfde euvel en kan beter niet gebruikt worden. Om compatibiliteit met voorgaande MemMan versies te behouden zijn voornoemde funkties nog wel aanwezig. Ter vervanging van de 'use' en '(re)store slot' funkties wordt het gebruik van de 'fastUse' en eventueel de 'fastCurSeg' funkties sterk aangeraden. Het adres waarop deze funkties aangeroepen kunnen worden kan worden opgevraagd door middel van funktie 50 'info'.

Verdere wijzigingen in MemMan 2.1 ten opzichte van versie 2.0:

- funkties 60 en 61 zijn vervallen wegens een structuurfout.
  Deze funkties gebruikten register IX als doorvoer parameter, dit register wordt gewijzigd door de interslot-call routine waarmee MemMan 2.1 aan de extended BIOS hook gekoppeld is.
- funkties GetTsrID (62) en TsrCall (63) vervangen nu de funkties 60 en 61.
  Het TSR-ID wordt bij deze funkties doorgegeven in register BC in plaats van in register IX.

Wijzigingen in MemMan 2.2 ten opzichte van versie 2.1:

- De funktie verwerkingsroutine van MemMan kan rechtstreeks worden aangeroepen.
  Het adres van deze routine kan worden opgevraagd via de info funktie (50).

- Het versienummer van MemMan kan via de info funktie (50) worden opgevraagd.
  Hierdoor kunnen programma's bepalen welke versie van MemMan is geïnstalleerd, zonder de IniChk funktie aan te roepen.

- Bug verwijderd inzake TSR-beheer op een systeem met meerdere memory-mappers

- Bug verwijderd betreffende de DeAlloc (20) funkties, bij het vrijgeven van PSEG's

Gebruikte terminologie

Segment	Geheugenblok van 16kB. Segmenten komen voor in Pagina specifieke segmenten (PSEG) en Flexibele segmenten (FSEG). De Flexibele segmenten kunnen op de pagina's 0,1 en 2 worden aangeschakeld. De Pagina specifieke segmenten alleen op hun eigen pagina. Er zijn drie soorten pagina specifieke segment: PSEG0000, PSEG4000 en PSEG8000. Ze zijn op respectievelijk pagina 0,1 en 2 aanschakelbaar. Indien DOS2 aktief is tijdens de installatie van MemMan, zal MemMan de segmenten die niet meer vrij zijn bij DOS2 niet in de segmenten tabel opnemen. Het geheugen dat door een voor MemMan geïnstalleerde RAMdisk in gebruik is, zal dus niet meer door MemMan beheerd kunnen worden. Wanneer een (Dos2-) RAMdisk na MemMan geïnstalleerd wordt, kunnen de segmenten die door de RAMdisk gebruikt werden wel weer door MemMan gebruikt worden, nadat de RAMdisk verwijderd is.
Heap	Blok geheugen in pagina 3 (ergens tussen &HC000 en &HFFFF) waarvan MemMan toepassingsprogramma's een stuk aan kunnen vragen en daarna vrij mogen gebruiken.
FastUse	Zelfde als Use, maar dan het adres waarop de routine direct aan te roepen is in pagina 3.
unCrash	Om te voorkomen dat segmenten aangevraagd zijn en door een crash van een programma nooit meer vrij zouden worden gegeven, voert de IniChk routine een unCrash uit. Hierbij worden alle segmenten weer vrijgegeven. Het unCrashen van een segment is te voorkomen door een segment de Reserved status te geven. Dit kan met de funktie SetRes (11). Normaal gesproken hoeft een segment niet de Reserved status gegeven te worden.

De principes

MemMan verdeelt het aanwezige geheugen in segmenten van 16 kB. Voordat een segment gebruikt mag worden moet het worden aangevraagd. Na gebruik dient het weer te worden vrijgegeven. Er zijn twee soorten segmenten: de zogenaamde pagina-specifieke ofwel PSEG's en de flexibele FSEG's.

PSEG's zijn segmenten die aangevraagd worden voor het gebruik op een bepaalde pagina, bijvoorbeeld van &h4000-&h7FFF of van &h8000-&hBFFF. Wanneer er een PSEG aangevraagd wordt zal MemMan zo mogelijk geheugensegmenten toewijzen die niet in een memory-mapper zitten.

FSEG's zijn segmenten die op elke willekeurige pagina kunnen worden ingeschakeld. Deze segmenten komen altijd uit memory mappers. Welk soort segment er ook aangevraagd wordt, MemMan zal een 16-bits 'segmentcode' teruggeven. Deze segmentcode is weer nodig bij het inschakelen of het weer vrijgeven van het segment. Wie alleen maar geheugen nodig heeft in het gebied van &h8000 tot &hBFFF kan dus het beste PSEG's aanvragen. MemMan gebruikt dan eerst zoveel mogelijk geheugen uit de 'oude' 16- en 64 Kb modules en gaat dan de mapper gebruiken.

Met behulp van MemMan hoeft er dus nooit meer naar geheugen gezocht te worden. Simpelweg een pagina aanvragen, gebruiken en uiteindelijk weer vrijgeven. Zo eenvoudig is dat.
Overigens is er een pagina die zich met MemMan niet laat schakelen. Pagina 3 bevat behalve de MemMan code zelf ook de stack (meestal) en een grote hoeveelheid systeemvariabelen. Er zitten nogal wat haken en ogen aan het wegschakelen van dat alles.

Funktie omschrijving MemMan 2.2

MemMan funkties kunnen worden uitgevoerd door een aanroep van de 'Extended BIOS' of EXTBIO hook, op adres &HFFCA. Het device ID van MemMan – 'M' oftewel &H4D – moet in register D worden geplaatst. Register E dient het MemMan funktienummer te bevatten. Na aanroep van een MemMan funktie kunnen alle registers gewijzigd zijn, behalve indien het tegendeel wordt vermeld bij de funktie-omschrijving.

Omdat de EXTBIO hook gebruikt wordt voor diverse systeem uitbreidingen zoals Kanji en RS232 interfaces, is het mogelijk dat MemMan funktie-aanroepen bijzonder langzaam verwerkt worden. De prestaties van de MemMan toepassings programma's kunnen aanmerkelijk worden verhoogd door funktie afhandelingsroutine van MemMan rechtstreeks aan te roepen, in plaats van de EXTBIO hook. Het adres waarop de funktie afhandelingsroutine aangeroepen kan worden, kan worden opgevraagd via de info funktie (50).

De meeste MemMan funkties bevinden zich in een apart geheugensegment in pagina 1. Deze funkties schakelen over op een een interne stack, waardoor MemMan toepassings programma's met een betrekkelijk kleine stack kunnen volstaan. Door een MemMan funktie worden maximaal twintig bytes op de stack van het toepassingsprogramma geplaatst. Dit geldt echter alleen indien de funktie rechtstreeks, of via de MemMan funktie-afhandelingsroutine wordt aangeroepen.

Een funktie-aanroep via de EXTBIO hook kan echter een bijzonder grote stack vereisen. Dit wordt veroorzaakt doordat alle uitbreidings-modules die aan de EXTBIO hook gekoppeld zijn elkaar aanroepen, net zo lang totdat één module de funktieaanroep herkent. Wanneer er tussendoor ook nog interrupts afgehandeld worden, kan het stackgebruik sterk oplopen. Al met al kan gesteld worden dat er bij een aanroep van de EXTBIO hook minimaal 150 bytes stackruimte beschikbaar moet zijn.

Het is derhalve verstandig om via de info funktie (50) het adres op te vragen van de routine die de MemMan funktie aanroepen afhandelt. Wanneer deze routine vervolgens rechtstreeks aangeroepen wordt, wordt de verwerkingssnelheid verhoogd en blijft het stack-gebruik beperkt.

De interruptstand blijft na een MemMan funktie-aanroep in meeste gevallen ongewijzigd. Sommige funkties zoals de diverse (Fast)Use funkties schakelen de interrupts echter altijd uit. Deze eigenschap is bijvoorbeeld van belang voor een TSR programma dat slechts een zeer kleine stack ter beschikking heeft. Zo lang de interrupts uit staan, kunnen alle MemMan funkties zonder problemen worden uitgevoerd, mits de funktie verwerkingsroutine van MemMan rechtstreeks wordt aangeroepen. Wanneer de interrupts echter aan staan is een grote stack vereist, omdat de interrupt-verwerkingsroutine enkele tientallen bytes op de stack plaatst.

Naam: **Use0**
Nummer: **0**
Funktie: **Aanschakelen van een segment op pagina 0 (adres gebied 0000..3FFF)**
In: **HL = Segmentcode**
Uit: **A = Resultaatcode (-1 = Mislukt, 0 = Gelukt)**

Het inschakelen van een segment in pagina 0 is alleen mogelijk indien het segment de MSX-standaard slot-schakel entry points bevat.

Opm: Deze funktie mag niet worden aangeroepen via de EXTBIO hook. Deze funktie mag alleen worden uitgevoerd door een rechtstreekse aanroep van de MemMan funktie afhandelingsroutine of de FastUse0 funktie. De adressen waarop deze routines aangeroepen kunnen worden, kunnen via de info funktie (50) worden verkregen.

Naam: **Use1**
Nummer: **1**
Funktie: **Aanschakelen van een segment op pagina 1 (adres gebied 4000..7FFF)**
In: **HL = Segmentcode**
Uit: **A = Resultaatcode (-1 = Mislukt, 0 = Gelukt)**

Opm: Deze funktie mag niet worden aangeroepen via de EXTBIO hook. Deze funktie mag alleen worden uitgevoerd door een rechtstreekse aanroep van de MemMan funktie afhandelingsroutine of de FastUse1 funktie. De adressen waarop deze routines aangeroepen kunnen worden, kunnen via de info funktie (50) worden verkregen.

Naam: **Use2**
Nummer: **2**
Funktie: **Aanschakelen van een segment op pagina 2 (adres gebied 8000..BFFF)**
In: **HL = Segmentcode**
Uit: **A = Resultaatcode (-1 = Mislukt, 0 = Gelukt)**

Opm: Deze funktie mag niet worden aangeroepen via de EXTBIO hook. Deze funktie mag alleen worden uitgevoerd door een rechtstreekse aanroep van de MemMan funktie afhandelingsroutine of de FastUse2 funktie. De adressen waarop deze routines aangeroepen kunnen worden, kunnen via de info funktie (50) worden verkregen.

Naam: Alloc
Nummer: 10
Funktie: Aanvragen van een segment
In: B = Segment voorkeuze code
Uit: HL = Segmentcode. (0000 = Geen segment meer vrij)
B = Segmentsoort code (-1 = FSeg, 0 = PSeg)

Segment voorkeuze code overzicht (Register B):

```
Bit 7 6 5 4 3 2 1 0
    ^ ^ ^ ^ ^ ^ ^ ^
    | | | | | | | |
    | | | | | | +-+-- Segment Type. 00 = PSEG0000
    | | 0 0 0 0                       01 = PSEG4000
    | |                               10 = PSEG8000
    | |                               11 = FSEG
    | +---------------- 1 = Prefereer TPA oftewel het
    |                       standaard MSXDOS RAM slot
    +------------------ 1 = Prefereer ongeëxpandeerd (dus
                          snel) slot
```

De bits 5 tot en met 2 zijn niet gebruikt en moeten 0 zijn.

Mocht een PSEG type aangevraagd, maar niet beschikbaar zijn wordt – indien mogelijk – een FSEG ter beschikking gesteld die dan het PSEG kan vervangen.

Naam: SetRes
Nummer: 11
Funktie: Segment de Reserved status geven
In: HL = Segmentcode

Geeft een segment de 'Reserved-status'; zodat het segment niet automatisch wordt vrij gegeven na aanroep van de IniChk routine. Normaal gesproken hoeven programma's de reserved status niet te zetten, behalve als een programma – bijvoorbeeld een Ramdisk – een segment voor eigen gebruik zeker wil stellen.

Naam: DeAlloc
Nummer: 20
Funktie: Teruggeven van een segment
In: HL = Segmentcode

Bij het verlaten van een programma dient deze funktie gebruikt te worden om alle aangevraagde segmenten weer terug te geven aan MemMan. De eventuele reserved status van het terug te geven segment wordt door DeAlloc automatisch opgeheven.
Segmenten die ook door DOS2 beheerd worden, worden door de DeAlloc funktie weer ter beschikking gesteld van DOS2.

Naam: ClrRes
Nummer: 21
Funktie: Reserved status van het segment opheffen
In: HL = Segmentcode

Het is niet nodig deze funktie vlak voor DeAlloc aan te roepen. DeAlloc heft zelf de Reserved status van het segment op.

Naam:	**IniChk**
Nummer:	**30**
Funktie:	**Initialisatie MemMan voor een programma**
In:	**A = Controle code**
Uit:	**A = Controle code + "M"**
	DE = Versie nummer (format: Versie #D.E)

Deze routine telt de ascii-waarde van de letter "M" op bij de inhoud van register A. Hierdoor kan er een MemMan aanwezigheids controle uitgevoerd worden. Verder wordt er een un-Crash uitgevoerd en worden de segmentcodes van de aktief aangeschakelde sloten berekend en opgeslagen voor CurSeg.
De IniChk funktie mag slechts één keer door ieder MemMan toepassings programma aangeroepen worden. Dit aanroepen van IniChk dient te gebeuren voordat de overige funkties van MemMan aangroepen worden. TSR programma's mogen de IniChk funktie **nooit** aanroepen.

Naam:	**Status**
Nummer:	**31**
Funktie:	**Status gegevens van MemMan ophalen**
Uit:	**HL = Aantal aanwezige segmenten**
	BC = Aantal nog vrije segmenten
	DE = Aantal segmenten in dubbel beheer bij DOS2 en MemMan
	A = Connected Status van de aangesloten hardware.
	Bit Funktie
	0 1 = Dos2 Mapper Support Routines aanwezig
	1-7 Gereserveerd, altijd 0

Als bit 0 van de Connected Status gezet is, zijn de geheugenbeheer funkties van MSX-DOS2 aanwezig.
Het aantal nog vrije segmenten kan lager zijn dan is aangegeven in register BC, omdat sommige segmenten na de installatie van MemMan door DOS2 gebruikt zijn – om bijvoorbeeld een RAM-disk te installeren.

Naam:	**CurSeg**
Nummer:	**32**
Funktie:	**Segmentcode van een aangeschakeld segment opvragen.**
In:	**B = Paginanummer (0,1,2,3)**
Uit:	**HL = Segmentcode**
	A = Segmentsoort code (255 = FSeg, 0 = Pseg)

Deze routine geeft de huidige segmentcode terug van een van de vier pagina's.

TSR programma's mogen deze funktie niet gebruiken om het aktieve segment in geheugen pagina 0 te bepalen. Om tijd te sparen wordt deze stand niet automatisch bepaald en opgeslagen bij de aanroep van een TSR. De aktieve segmenten in de pagina's 1 en 2 worden echter bij iedere hook-aanroep opnieuw bepaald en kunnen ten alle tijde via deze funktie opgevraagd

worden. Omdat in pagina 3 altijd hetzelfde segment aktief is, is ook de segmentcode van pagina 3 altijd opvraagbaar.

Een snellere variant van deze funktie is FastCurSeg routine. gebruikt. Het adres waarop deze routine aangeroepen kan worden is via de Info funktie (50) op te vragen.

Naam: **StoSeg**
Nummer: **40**
Funktie: **Huidige segmenten stand opslaan**
In: **HL = Buffer adres (9 bytes groot)**

De voor MemMan bekende segmentcodes van de aktief aangeschakelde sloten worden opgeslagen in het buffer. Deze segmentcodes zijn in beginsel door IniChk berekend en later door de Use funkties geupdate. De opgeslagen stand is niet de huidige stand, maar de voor MemMan bekende stand. TSR's kunnen hiermee dus niet de aktieve stand opslaan.

Opm: Deze funktie mag niet worden aangeroepen via de EXTBIO hook. Deze funktie mag alleen worden uitgevoerd door een rechtstreekse aanroep van de MemMan funktie afhandelingsroutine. Het adres waarop deze routine aangeroepen kan worden, kan via de info funktie (50) worden verkregen.
Natuurlijk kunnen ook de (Fast)CurSeg funkties gebruikt worden om de momentele segment-stand op te vragen.

Naam: **RstSeg**
Nummer: **41**
Funktie: **Opgeslagen segment-stand aktief maken**
In: **HL = Buffer adres**

De in het buffer opgeslagen segment-stand wordt weer aktief gemaakt en wordt opgeslagen voor CurSeg.

Opm: Deze funktie mag niet worden aangeroepen via de EXTBIO hook. Deze funktie mag alleen worden uitgevoerd door een rechtstreekse aanroep van de MemMan funktie afhandelingsroutine. Het adres waarop deze routine aangeroepen kan worden, kan via de info funktie (50) worden verkregen.
Natuurlijk kunnen ook de (Fast)Use funkties gebruikt worden om een segment-stand te herstellen.

Naam: **Info**
Nummer: **50**
Funktie: **Geeft informatie over o.a. aanroep-adressen van MemMan funkties**
In: **B = Informatie nummer (0..7)**
Uit: **HL = Informatie**

Informatie nummer overzicht. Tussen haakjes staan de equivalente MemMan funktie codes:

0 Aanroepadres van FastUse0 (funktie 0)
1 Aanroepadres van FastUse1 (funktie 1)
2 Aanroepadres van FastUse2 (funktie 2)
3 Aanroepadres van TsrCall (funktie 63)
4 Aanroepadres van BasicCall
5 Aanroepadres van FastCurSeg (funktie 32)
6 Aanroepadres van MemMan, de funktie-afhandelingsroutine
7 Versienummer van MemMan, format: Versie #H.L

De bovengenoemde funktie-adressen mogen door een toepassingsprogramma of TSR rechtstreeks aangeroepen worden. Alle entry adressen liggen gegarandeerd in pagina 3.

De funkties worden snel uitgevoerd omdat de MemMan CALL naar de EXTBIO hook vervalt en de funktie-codes in registers D en E niet uitgeplozen hoeven worden. Een ander voordeel is dat parameters ook via het register DE doorgegeven kunnen worden, dit is vooral van belang bij de TsrCall en BasicCall funkties.

Bijvoorbeeld, de initialisatie routine van een TSR kan de benodigde funktieadressen via de Info (50) funktie opvragen en deze vervolgens voor later gebruik in de TSR programmacode opslaan, wat de snelheid van het TSR programma zeer ten goede kan komen.

Een exacte beschrijving van de bovenstaande funkties kan gevonden worden bij de MemMan funktie waarvan het nummer tussen haakjes is aangegeven.
Houd echter onder de aandacht dat de 'snelle' funkties op de volgende punten van de gewone MemMan funkties verschillen:

FastUse0-2: Schakelt een segment in in een bepaalde geheugen pagina. Zie de omschrijving bij de memMan 'Use' funkties.

TsrCall: Register [DE] wordt ongewijzigd aan de TSR doorgegeven. Dit in tegenstelling tot funktie 63 (TsrCall), register DE is dan al bezet om het MemMan funktienummer in op te slaan.

FastCurSeg: In register [A] komt geen zinnige waarde terug. De MemMan CurSeg funktie (32) geeft aan of het een FSEG/PSEG betreft.

BasicCall:	**Heeft geen MemMan funktie nummer.**
Funktie:	**Aanroepen van een routine in de BASIC ROM.**
In:	**IX = Call address in pagina 0 of 1**
	AF, HL, BC, DE = dataregisters voor de BASIC-ROM
Uit:	**AF, HL, BC, DE = dataregisters van de BASIC-ROM**
	Interrupts disabled

Via deze funktie kunnen TSR's een routine aanroepen die zich in pagina 0 en/of pagina 1 van het BASIC ROM bevindt. De bios moet al in pagina 0 aangeschakeld zijn. In pagina 1 wordt de BASIC ROM door MemMan aangeschakeld.
Dit is bijvoorbeeld noodzakelijk om de math-pack routines aan te kunnen roepen die in pagina 0 van de BASIC ROM zitten, maar tussendoor ook een aantal routines in pagina 1 aanroepen. De H.STKE (stack error) hook wordt afgebogen, zodat na een eventueel op getreden BASIC error de interne stacks van MemMan gereset kunnen worden.

MemMan:	**Heeft geen MemMan funktienummer**
Funktie:	**Rechtstreeks aanroepen van een MemMan funktie.**
In:	**E = MemMan funktienummer**
	AF, HL, BC = Dataregisters afhankelijk van de aan te roepen funktie.
Uit:	**AF, HL, BC, DE = Dataregisters afhankelijk van de aangeroepen funktie.**

Een aanroep van deze routine heeft hetzelfde effect als het aanroepen van een MemMan funktie via de EXTBIO hook. Doordat echter de aanroep naar de EXTBIO hook vervalt, worden de overige uitbreidingen die aan deze hook gekoppeld zijn niet aangeroepen. Hierdoor blijft het stack-gebruik beperkt en wordt de verwerkingssnelheid verhoogd.

Naam:	**GetTsrID**
Nummer:	**62**
Funktie:	**Bepaal TSR ID code**
In:	**HL = Pointer naar de TsrNaam (12 tekens).**
	Ongebruikte posities opvullen met spaties.
Uit:	**Gevonden: Carry clear (NC)**
	BC = TSR ID code
Anders:	**Carry set (C)**

Naam:	**TsrCall**
Nummer:	**63**
Funktie:	**Roep het driver-entry van een TSR aan**
In:	**BC = ID code van de aan te roepen TSR**
	AF, HL, DE worden ongewijzigd doorgeven aan de TSR.
Uit:	**AF, HL, BC, DE komen ongewijzigd terug van de TSR.**

Merk op dat alhoewel het DE register ongewijzigd aan de TSR wordt doorgegeven, het niet voor parameter-invoer benut kan worden. De Extended BIOS funktiecode van MemMan (D='M' E=63) moet namelijk in dat register geplaatst worden.
Bij de Fast-TsrCall routine treedt deze complicatie niet op; het adres van deze routine kan middels de Info (50) funktie opgevraagd worden.

Naam:	**HeapAlloc**
Nummer:	**70**
Funktie:	**Alloceer ruimte in de heap**
In:	**HL = Gewenste grootte van de ruimte (in bytes)**
Uit:	**Genoeg ruimte: HL = Startadres van de ruimte**
	Anders: HL = 0000

Door middel van deze funktie kan een stuk geheugen gealloceerd worden. Het geheugenblok zal zich gegarandeerd in pagina 3 bevinden.
De heap is vooral nuttig voor TSR programma's, die hem bijvoorbeeld als tijdelijke of permanente diskbuffer kunnen gebruiken. Ook andere buffers – waarvan het absoluut noodzakelijk is dat ze zich in pagina 3 bevinden – kunnen op de heap worden geplaatst.
Aangevraagde blokken geheugen uit de heap blijven onbruikbaar voor andere programma's totdat een 'HeapDeAlloc' is uitgevoerd (funktie 71).
De grootte van de heap kan worden ingesteld door middel van het configuratie programma CFGMMAN.

Naam:	**HeapDeAlloc**
Nummer:	**71**
Funktie:	**Geef geAlloceerde ruimte van de heap weer vrij**
In:	**HL = Startadres van de ruimte**

Naam:	**HeapMax**
Nummer:	**72**
Funktie:	**Geef de lengte van het grootste vrije blok geheugen in de heap terug**
Uit:	**HL = Lengte van het grootste vrije blok**

Gebruik van de stack onder MemMan

MemMan toepassingsprogramma's dienen de stack pointer (SP) bij voorkeur in pagina 2 of 3 (tussen &h8000 en &HFFFF) te plaatsen. Indien MemMan door een hook-aanroep geactiveerd wordt, wordt het huidige segment in pagina 1 (&h4000 tot &h8000) namelijk weggeschakeld om plaats te maken voor de TSR-Manager en de eventuele TSR's. Indien de stack zich op dat moment in pagina 1 bevindt zal de computer vastlopen.

Indien TSR's na een BDOS call of interrupt via een BIOS-hook worden aangeroepen treden geen stackproblemen op; ook niet indien de stack van het toepassingsprogramma in pagina 1 staat. De BDOS en interrupt funkties gebruiken namelijk hun eigen stack in pagina 3. De stack bevindt zich dan alsnog in pagina 3 op het moment dat de hook aangeroepen wordt.

Bestaande CP/M en MSX-DOS programmatuur is dus zonder problemen in combinatie met MemMan 2 te gebruiken – maar alleen indien de standaard BDOS calls gebruikt worden. Wanneer echter via een interslot call een BIOS routine rechtstreeks aangeroepen wordt, dient de stack in pagina 2 of 3 te staan. Reserveer in dat geval minimaal 150 bytes voor de stack.

BIOS aanroepen onder Turbo Pascal

Indien in een Turbo Pascal programma interslot-calls naar de BIOS gebruikt worden, is het belangrijk dat de stack in pagina 2 of 3 staat. Op het moment dat de BIOS dan een hook aanroept kan MemMan veilig de TSR's aktiveren. De positie van de stack is afhankelijk van het maximum programma adres dat tijdens de compilatie in Turbo Pascal is ingesteld. De stack bevindt zich in Turbo Pascal direkt onder het variabelen geheugen. Het variabelen geheugen dient bij programma's die de BIOS aanroepen dus ruim boven adres &h8000 geplaatst te worden.

Is geen source voorhanden, dan is het mogelijk om met een debugger het stack adres van Turbo Pascal programma's aan te passen. De initialisatie code van een TP programma ziet er als volgt uit:

```
start:  jp init
        ...
        ...
init:   ld sp,100h
        ld hl,nn
        ld de,nn
        ld bc,nn
        call yy
        ld hl,nn
        ld de,stack     ;DE bevat het stack adres, hoeft
        ld bc,nn        ; alleen aangepast te worden als het
        call zz         ; lager is dan &h80A0
        ...
```

Het stackadres in register DE kan bijvoorbeeld op &hC100 gezet worden.

Aanroepen van TSR's door middel van MemMan 2

TSR programma's worden door MemMan 2 in geheugensegmenten geplaatst die buiten de standaard 64 kB werkgeheugen van MSX-DOS liggen. Het aanroepen van de TSR's gebeurt dan ook altijd via MemMan. De TSR-aanroepen kunnen in twee categorieën worden onderverdeeld: Aanroepen vanaf hooks en aanroepen door toepassingsprogramma's.

In de hook-tabel van de TSR-file wordt opgegeven naar welke routines in de TSR de hooks moeten worden afgebogen. Op het moment dat één van de afgebogen hooks wordt aangeroepen, zal MemMan het segment waarin de TSR zich bevindt inschakelen. Vervolgens wordt de routine aangeroepen die de hook-aanroep verwerkt.

Verwerking van hook-aanroepen

Bij het schrijven van routines die een hook-aanroep verwerken dient met het volgende rekening gehouden te worden.

TSR's worden altijd aangeroepen met de interrupts uit. De interrupts dienen bij voorkeur zo snel mogelijk weer aangeschakeld te worden. Het is dan is dan ook raadzaam om TSR-routines met een EI instructie te beginnen.

Alleen de registers AF, HL, DE en BC worden door MemMan ongewijzigd aan de TSR doorgegeven.

De segment-stand van de geheugen-pagina's 0, 2 en 3 blijft ongewijzigd bij het aanroepen van de TSR. In pagina 1 wordt het segment ingeschakeld waarin de TSR zich bevindt. TSR's mogen door gebruik te maken van de (Fast-) CurSeg en Use funkties tijdelijk een segment in pagina 2 inschakelen.

De TSR-manager plaatst precies één returnadres op de stack. Direct daaronder bevindt zich de stack die actief was op het moment dat de hook werd aangeroepen. Onder het returnadres naar de manager staat dus het returnadres naar de BIOS routine die de hook heeft aangeroepen en de eventuele registers die door de BIOS op de stack zijn 'gepushed'.
Merk op dat hooks ook door gewone DOS-programma's kunnen worden afgebogen. Wanneer deze programma's de hook afbuigen door middel van een 'JP' instructie, zullen er geen extra returnadressen op de stack worden geplaatst. Wanneer het programma een kopie van de hook bijhoudt en deze vervolgens aanroept, zal bij aanroep van een TSR de stack er precies uitzien zoals in de voorgaande alinea omschreven is.
Sommige programma's buigen de hooks echter af door middel van een 'RST 30H' instructie, oftewel een interslot call. De interslot call routine plaatst diverse gegevens op de stack. Wanneer het programma vervolgens de hook aanroept die door MemMan werd geïnstalleerd, zal stack zijn vervuild door de interslot call waarmee het toepassingsprogramma werd aangeroepen. Zulke programma's kunnen niet gebruikt worden in combinatie met TSR's die een 'schone' stack vereisen.

Aanroepen van TSR's

Een TSR mag alleen worden verlaten door terug te springen naar de TSR-Manager. Bij het terugkeren naar de manager na een hook-aanroep dient in het schaduw-register A' een vlagwaarde te worden geplaatst. Deze funktie hiervan is als volgt.

Bit	Naam	Funktie
0	QuitHook	1 = Stop hook-verwerking
1 .. 7	Gereserveerd	Altijd 0

- quitHook: Dit bit dient te worden gezet om aan te geven dat MemMan de volgende TSR's en routines die aan de hook zijn gekoppeld niet meer mag aanroepen. Hierdoor is het mogelijk om vanuit de TSR direct terug te springen naar het programma dat de hook heeft aangeroepen.
  Dit kan bijvoorbeeld van toepassing zijn voor een TSR die Basic statements verwerkt via de CMD-hook. Indien de TSR het statement niet herkent, zal het de originele data-registers weer moeten herstellen en terugkeren met de quitHook vlag ge-reset. MemMan zal dan ook de overige TSR's die aan de CMD-hook zijn gekoppeld aanroepen.
  Wanneer de TSR het statement echter wel heeft herkend en verwerkt, zal de text-pointer naar het begin van volgende statement wijzen. De overige TSR's mogen dan niet aangeroepen worden en er dient direct terug te worden gesprongen naar de Basic-interpreter. Het zetten van de quitHook-vlag is voldoende om dit te bereiken.

- Overige: De overige bits zijn gereserveerd en dienen altijd 0 te zijn.

Voorbeeld 1

Stel dat een TSR-routine aan de interrupt hook H.TIMI is gekoppeld. Deze routine heeft tot funktie bij iedere interrupt een teller te verhogen. Deze routine zou er als volgt uit kunnen zien:

```
INTTSR:     EI          ;Dit is de routine die een teller
            LD HL,COUNT ; verhoogd
            INC (HL)

            EX AF,AF'   ;Bewaar register AF
            SUB A       ;Maak quitHook vlag en andere bits nul
            EX AF,AF'   ;Vlag naar A', herstel AF
            RET         ;Keer terug naar TSR-Manager
```

Voorbeeld 2

Stel dat een TSR aan de hook H.CHPU hangt. De BIOS routine CHPUT – die de hook H.CHPU aanroept – ziet er als volgt uit:

```
CHPUT:      PUSH HL       ;Bewaar data-registers
            PUSH DE
            PUSH BC
            PUSH AF
            CALL H.CHPU   ;Roep de CHPUT-Hook aan
            ...           ;Vervolg van de routine
```

Wanneer de hook H.CHPU aangeroepen wordt, zal de stack die MemMan aan de TSR doorgeeft er als volgt uitzien:

```
(SP+0)  = Terugkeer adres naar de TsrManager
(SP+2)  = Terugkeer adres naar het BIOS
(SP+4)  = AF
(SP+6)  = BC
(SP+8)  = DE
(SP+10)= HL
(SP+12)= Terugkeer adres naar de applicatie routine
(SP+14)= ...
```

Wanneer er na uitvoer van de TSR direct teruggesprongen moet worden naar het toepassingsprogramma dat de CHPUT routine heeft aangeroepen, moeten de op de stack geplaatste registers en het return-adres naar de BIOS van de stack verwijderd worden. Het terugkeer adres naar MemMan dient echter bewaard te blijven. Hiertoe kan de volgende routine in de TSR worden opgenomen:

```
CHARTSR:    EI            ;Dit is de routine die de H.CHPU aanroep
            ...           ; verwerkt
            ...
            ...
            POP IX        ;Haal terugkeer adres naar de TsrManager
            POP HL        ;Weg: Terugkeer adres naar het BIOS
            POP AF        ;Register AF herstellen
            POP BC        ;Register BC herstellen
            POP DE        ;Register DE herstellen
            POP HL        ;Reguster HL herstellen

            EX AF,AF'     ;Bewaar register AF
            LD A,1        ;Zet QuitHook vlag: Stop hook-verwerking
            EX AF,AF'     ;Vlag in A', herstel register AF

            JP (IX)       ;Keer terug via de TsrManager
```

Merk op dat de index- en schaduwregisters door TSR's vrij gebruikt mogen worden, ze hoeven niet ongewijzigd te blijven. Bij het aanroepen en verlaten van TSR's geeft MemMan namelijk alleen de registers AF, HL, DE en BC ongewijzigd door. Dit vanwege het feit dat de TSR's worden aangeroepen door middel van een inter slot call, deze routine behoudt alleen AF, HL, DE en BC.

Verwerking van TSR-aanroepen via TsrCall

De tweede manier waarop TSR-programma's aangeroepen kunnen worden is door middel van de MemMan funktie TsrCall (63). Door middel van deze funktie kunnen toepassingsprogramma's de TSR bepaalde acties laten uitvoeren en gegevens met de TSR uitwisselen.

In de header-tabel van de TSR-file moet worden vastgelegd welke routine de TsrCall-aanroepen verwerkt. Zie voor meer informatie hiervoor het hoofdstuk over de indeling van TSR bestanden. Indien de TSR niet door middel van TsrCall wordt aangesproken, moet de routine uit een simpele "RET" instructie bestaan.

Bij het schrijven van routines die door middel van de MemMan funktie TsrCall worden aangeroepen, dient met de volgende punten rekening te worden gehouden.

TSR's worden altijd aangeroepen met de interrupts uit. De interrupts dienen bij voorkeur zo snel mogelijk weer aangeschakeld te worden. Het is dan is dan ook raadzaam om TSR-routines met een EI instructie te beginnen.

Alleen de registers AF, HL, DE en BC worden door MemMan ongewijzigd aan de TSR doorgegeven. Merk hierbij op dat register BC gebruikt wordt om het TsrID van de TSR in aan te geven, register BC kan derhalve niet als invoer-register gebruikt worden.

De segment-stand van de geheugen-pagina's 0, 2 en 3 blijft ongewijzigd bij het aanroepen van de TSR. In pagina 1 wordt het segment aangeschakeld waarin de TSR zich bevindt. TSR's mogen, wanneer ze middels TsrCall zijn aangeroepen, van alle beschikbare CurSeg en Use funkties gebruik maken om tijdelijk andere segmenten in te schakelen.

Op het moment dat de TSR via TsrCall wordt aangeroepen is een interne stack van MemMan actief. Deze stack bevindt zich in pagina 3 en is 160 bytes groot. Dit zal voor de meeste toepassingen voldoende zijn. Merk echter op dat deze stack ook gebruikt wordt tijdens het uitvoeren van MemMan funkties en eventueel optredende interrupts, de beschikbare stackruimte is dus altijd minder dan de maximale 160 bytes.

De TSR mag alleen worden verlaten door terug te springen naar de TSR-Manager. Het returnadres naar de manager bevindt zich boven op de stack, het uitvoeren van een "RET" instructie zal dus in de meeste gevallen voldoen. Er hoeven geen vlag-waarden aan de manager te worden terug gegeven. Alleen de registers AF, HL, DE en BC worden ongewijzigd doorgegeven het programma dat de TsrCall heeft uitgevoerd.

Funktieaanroepen door TSR's

MemMan funkties kunnen op drie manieren worden aangeroepen. Via de EXTBIO hook (adres 0FFCAh), via de funktiehandelingsroutine in pagina 3 en door het rechtstreeks aanroepen van één van de zogenaamde 'fast'-routines. Het wordt sterk aangeraden om funktieaanroepen via de EXTBIO hook zoveel mogelijk te vermijden. Een aanroep naar EXTBIO kan – wanneer er veel uitbreidingen aan de hook gekoppeld zijn – zeer veel stackruimte kosten en zeer langzaam verwerkt worden. Bovendien dient bij het aanroepen van EXTBIO de stack in pagina 3 te staan. Bij het aanroepen van MemMan funkties via de funktieafhandelingsroutine is het ook toegestaan dat de stack zich in pagina 2 bevindt.
Het wordt sterk aangeraden om in de initialisatieroutine van een TSR het adres van de funktieafhandelingsroutine op te vragen. Dit kan gebeuren door via de EXTBIO hook de funktie Info (50) aan te roepen. De overige MemMan-funktieaanroepen kunnen dan via de funktieafhandelingsroutine worden uitgevoerd, in plaats van via de EXTBIO hook.
TSR-programma's mogen nooit gebruik maken van de funktie IniChk (30). Niet tijdens de initialisatie noch tijdens een aanroep via een hook of de funktie TsrCall.

Aanroepen naar de Main-ROM

Het aanroepen van routines in het BIOS-ROM is geen probleem. Hiervoor kan de interslot call routine op adres &H1C of &H30 worden gebruikt. Wanneer een TSR vanaf een hook worden aangeroepen zal meestal de BIOS-ROM al actief zijn in pagina 0. Het is echter mogelijk dat een (DOS) toepassingsprogramma de hook heeft afgebogen en zijn eigen routines in pagina 0 heeft ingeschakeld. Het is dus niet veilig om te veronderstellen dat de BIOS-ROM altijd actief is en aangeroepen kan worden door middel van een 'CALL' instructie.

Echter, wanneer een TSR aan een hook hangt waarmee Basic statements kunnen worden uitgebreid, mag worden verondersteld dat er geen DOS programma aan dezelfde hook actief is. Routines in de BIOS-ROM kunnen in dat geval dus wel via een simpele 'CALL' instructie worden aangeroepen.

Om routines in de Basic-ROM – die zich in de pagina's 0 en 1 bevindt – aan te kunnen roepen, dient gebruik te worden gemaakt van een speciale MemMan funktie: BasicCall. Deze funktie schakelt het segment waarin de TSR zich bevindt tijdelijk uit en plaatst de Basic-ROM in pagina 1.
Wanneer bijvoorbeeld dat een Basic-ROM routine – zoals een Math-Pack funktie – in pagina 0 aangeroepen wordt door middel van een interslot call, zal het systeem crashen. Dergelijke Basic-ROM routines roepen namelijk ook subroutines aan in pagina 1. Op dat moment moet vanzelfsprekend de Basic-ROM ook in pagina 1 actief zijn.
De BasicCall funktie van MemMan is geschikt om zulke problemen op te lossen. Alvorens de gewenste routine in het Basic-ROM wordt aangeroepen, zal in pagina 1 de ROM worden ingeschakeld, waardoor aanroepen naar iedere locatie in de Basic-ROM correct uitgevoerd worden.
De specificaties van de BasicCall funktie zijn de vinden in de funktiespecificaties van MemMan, bij de Info funktie (50).

TSR bestanden: Achtergrond

Bij het ontwikkelen van MemMan 2 en de TSR bestands-structuur is het standpunt ingenomen, dat de TSR's zowel eenvoudig ontwikkeld als eenvoudig gebruikt moesten kunnen worden. Iedereen die machtinetaal programma's voor de MSX kan ontwikkelen, zou ook TSR's moeten kunnen schrijven en iedereen die MSX programma's kan gebruiken zou ook TSR's moeten kunnen installeren. We zijn van mening dat MemMan 2 redelijk aan deze uitgangspunten voldoet. Maar desondanks verschilt een TSR op diverse punten van een gewoon machinetaal programma.

REL-Tabel

De TSR's kunnen door TsrLoad op iedere willekeurige plaats in pagina 1 (adresgebied van &H4000 tot &H7FFF) worden geplaatst. Daartoe gebruikt de TSR-Loader een tabel met alle programma-relatieve adressen. Deze tabel wordt door het programma LinkTsr automatisch in de TSR-file geplaatst.

De informatie die nodig is om machinecode op een willekeurig adres te kunnen laden wordt afgeleid uit een zogenaamde relocatable file. TSR's moeten daarom geprogrammeerd worden met een assembler die 'relocatable' files kan maken, kortweg '.REL' bestanden. Voorbeelden hiervan zulke assemblers zijn GEN80 uit het DEVPAC2 pakket van HiSoft en M80 uit de MSX-DOS tools van Ascii.

Een linker wordt normaliter gebruikt om verschillende REL-files samen te voegen tot een COM-file die op een vast adres moet worden opgestart. Alle informatie omtrend de benodigde aanpassingen in de code wordt op dat moment door de linker gebruikt en is niet meer aanwezig in de COM-file.

Om nu toch die informatie te kunnen behouden heeft het MST de reeds genoemde TSR-linker ontwikkeld: LinkTsr oftewel LT.COM. Deze linker koppelt de verschillende REL-files aan elkaar, net zoals een gewone linker dit zou doen. De TSR-Linker bewaart echter alle programma-relatieve adressen en plaatst deze apart in een tabel in de uiteindelijke TSR-file. Het aanmaken van de REL-tabel gebeurt geheel automatisch, voor de programmeur is de precieze opbouw en de positie van de REL-tabel niet van belang.

Initialisatie code

Ieder programma heeft wel een stuk initialisatie code, zo ook de meeste TSR's. Deze code wordt maar een keer gebruikt en is daarna ook overbodig. Om de initialisatie code na gebruik eenvoudig te kunnen verwijderen gebruikt de TSR-Loader een truukje. De initialisatie code wordt altijd aan het einde van het TSR programma geplaatst. Vlak voordat de TsrLoader de TSR op de hooks aansluit, wordt de initialisatie code aangeroepen. Na het uitvoeren van die code wordt het geheugen dat bezet werd door de initialisatie code weer vrijgegeven voor gebruik door andere TSR programma's. De hoeveelheid benodigd geheugen wordt zo tot een absoluut minimum beperkt.

Hook-tabel

De TSR's kunnen worden gekoppeld aan de hooks in het systeem-geheugen. In de TSR-file moet een tabel.worden opgenomen waarin vermeld wordt welke hooks door de TSR gebruikt worden en naar welke routine moet worden gesprongen als één van die hooks wordt aangeroepen. Net als de REL-tabel wordt de hook-tabel tijdens de installatie van de TSR door de Loader gebruikt. Deze tabellen worden niet opgenomen in de uiteindelijke programma-code van de TSR en kosten dus geen werkgeheugen in het TSR programma segment. Het is niet mogelijk om na de installatie van de TSR nog extra hooks af te buigen.

De hook-tabel begint met twee bytes die de lengte van de tabel aangeven. Bij het bepalen van de tabellengte worden deze twee lengte-bytes meegeteld. Na het lengte woord staan de hook-afbuig gegevens. Ieder element in de hook-tabel is vier bytes lang. De eerste twee bytes van het element geven het adres aan van de af te buigen hook. De volgende twee bytes geven het adres aan van de routine die door MemMan moet worden aangeroepen wanneer de hook aangeroepen wordt.

Header tabel

Iedere TSR begint met een speciale tabel: de zogenaamde file-header. Aan de hand van deze header kan de TsrLoader de positie bepalen van de programmacode van de TSR en de tabellen in de file.

Iedere TSR-listing dient dus te beginnen met de header tabel. De indeling van deze tabel is als volgt:

```
defb 'MST TSR',13,10     ;TSR file ID
defb 'MSTs TSRname'      ;TSR naam
defb 26                  ;Einde tekst markering
defw 0002                ;Versie nummer voor TsrLoad
defw tsrStart            ;Base-adres
defw init                ;Start initialisatiecode
defw kill                ;Destructie routine
defw talk                ;Interactie routine
defw tsrLen              ;Lengte van TSR-Code
defw iniLen              ;Lengte van init-Code
```

De betekenis van de elementen in de header tabel is als volgt:

- MST TSR file ID, 9 bytes.
  Aan deze identificatietekst herkent TsrLoad of het bestand al dan niet een TSR bestand is. Deze string mag dus niet worden vrij worden gekozen door de TSR-programmeur.

- TSR naam, 12 bytes
  Een zo uniek mogelijke ID-tekst gekozen door de programmeur. Toepassingen kunnen door middel van deze naam met de TSR communiceren. Ongebruikte posities opvullen met spaties.

- Einde tekst markering, 1 byte
  Afsluitend staat altijd Control-Z (^Z), voor het geval TYPE commando losgelaten wordt op de TSR programmacode.

- Vereiste versie nummer van de TSR-Loader, 2 bytes.
  De loader controleert dit nummer, om te voorkomen dat een 'verouderde' loader een TSR file in wil lezen die een niet-compatible struktuur heeft. De TSR-loader is upward-compatible, zodat ook eventuele TSR's van een vorige MemMan versie geïnstalleerd kunnen worden. Gebruikers kunnen dan volstaan met het vervangen van de loader en kunnen dezelfde TSR's blijven gebruiken.
  Het versienummer is 0001 indien door de TSR alleen MemMan 2.1 funkties aangeroepen worden. Indien ook van de specifieke MemMan 2.2 funkties gebruik wordt gemaakt, dient het versienummer 0002 in de header te worden geplaatst.

- Base-adres van de TSR programmacode, 2 bytes.
  Dit is het adres van het eerste byte van de TSR programma code. Dit is altijd het eerste byte na de header in de TSR file. Deze waarde wordt door de loader gebruikt om de programma relatieve adressen aan te passen.

- Initialisatie adres, 2 bytes.
  Wordt aangeroepen tijdens de installatie van de TSR. De Init-routine MOET helemaal achteraan het TSR-programma staan, zodat de loader de ruimte van de INIT routine weer eenvoudig kan vrijgeven nadat de TSR geïnstalleerd is.

Na de aanroep moet de initroutine in register A een vlag teruggeven. De definitie van dit vlaggenregister is als volgt:

Bit	Funktie
0	0 = Initialisatie gelukt 1 = Initialisatie mislukt
1	0 = Geen intro-tekst 1 = Pointer naar intro-tekst in DE
2 .. 7	Gereserveerd, altijd 0

Indien de TSR initialisatie routine met bit 0 van het A register aangeeft dat de installatie niet gelukt is, verwijdert de TSR-Loader de gehele TSR weer. Als de TSR met bit 1 aangeeft dat er een intro tekst is, dan wordt deze door de TSR-Loader afgedrukt. TSR's mogen nooit zelf teksten op het scherm afdrukken. Hierdoor is mogelijk om in de toekomst ook in grafische omgevingen de intro tekst af te kunnen drukken. Ook het onderdrukken van de intro teksten is dan eenvoudiger. De intro-tekst wordt afgesloten een met 0-byte.

De hooks worden pas na de aanroep van de initialisatie routine geïnstalleerd.

- Destruktie adres, 2 bytes.
  Wordt door het programma TsrKill aangeroepen bij het verwijderen van de TSR. Deze routine dient het geheugen – segmenten of heap ruimte – dat bij MemMan aangevraagd is vrij te geven. Op het moment dat deze routine aangeroepen wordt zijn de hooks al afgekoppeld.

- Interactie adres, 2 bytes.
  Wordt aangeroepen door toepassingsprogramma's, via de TsrCall funktie van MemMan.

- Lengte van de TSR-code, 2 bytes.
  Deze lengte betreft de TSR-programmacode, zonder de initialisatie routine.

- Lengte van de initialisatie routine, 2 bytes.
  De initialisatie routine staat helemaal achteraan de TSR, en wordt bij de initialisatie van de TSR in hetzelfde segment gezet als de TSR code zelf. De ruimte die de initialisatie routine in beslag neemt wordt na de aanroep ervan weer vrij gegeven.

File structuur

Ieder TSR programma bestaat uit onderdelen die elk op een vaste positie in het bronbestand moeten staan. Deze programma onderdelen zijn:

- Header tabel
- TSR programmacode
- TSR initialisatiecode
- Hook tabel

Deze volgorde dient strikt in acht te worden genomen. In de meeste gevallen zal de gehele TSR uit slechts één REL-file bestaan die in één keer wordt gelinkt. De programmeur hoeft er dan alleen op te letten dat de elementen in de juiste volgorde in de bron-listing staan. Indien de TSR uit meerdere REL-files is opgebouwd worden in de files in de opgegeven volgorde ingeladen en achter elkaar geplaatst, de TSR-linker voert hierop geen controles uit.

Het bestand TSRFRAME.GEN op de TSR ontwikkel-disk bevat het raamwerk van een TSR-listing. Door van dit raamwerk uit te gaan bij het schrijven van een nieuw TSR programma is het eenvoudig de juiste structuur aan te houden.

Interrupts

Menig machinetaal programmeur zal wel eens een programma hebben geschreven dat via de interrupt routine moest worden aangeroepen. Soms werkt zo'n programma om de één of andere duistere reden dan niet goed of helemaal niet. Tijd dus om eens wat licht te laten schijnen over de in het duister gehulde interrupts.

Een uitbreiding van de BIOS interrupt routine moet aan een van de twee interrupt hooks worden 'gehangen'. Er moet een keuze gemaakt worden tussen de hooks H.KEYI en H.TIMI. Beide worden door de interrupt routine aangeroepen, maar er zijn wel degelijk verschillen.
De hook H.KEYI wordt bij iedere interrupt aangeroepen, terwijl de hook H.TIMI met een vaste regelmaat wordt aangeroepen. Om precies te zijn 50 of 60 keer per seconde. Een achtergrond muziekje moet dus aan de hook H.TIMI hangen, omdat de afspeelsnelheid regelmatig moet zijn.

Een interrupt is – eenvoudig gezegd – een melding van een randapparaat aan de processor. Met zo'n interrupt geeft het randapparaat aan dat de processor in actie moet komen. Zo kan een modem met een interrupt aan de processor melden dat er een teken is binnengekomen dat moet worden opgehaald. Het programma dat een teken van het modem op moet halen moet dus bij iedere interrupt aangeroepen worden en moet daarom aan de hook H.KEYI hangen.

Opbouw

Interrupt routines moeten zo geschreven zijn, zodat ze op ieder moment aangeroepen kunnen worden. Ze mogen niets veranderen dat de werking van het hoofdprogramma ongewenst kan beïnvloeden. Zo moeten alle registers hun originele waarde behouden, anders zouden er wel eens hele vreemde resultaten uit een berekening van het hoofdprogramma kunnen komen op het moment dat een interrupt routine zomaar de inhoud van een aantal registers veranderd. Verder moet er rekening worden gehouden met de stand van de primaire en secundaire slot select registers en eventuele Memory Mapper instellingen.

Het opslaan van de registers wordt door de interrupt afhandelings routine in het BIOS verricht. Een programma dat aan deze hook 'hangt' hoeft dus niet zelf ook nog eens alle registers op te slaan. Hierop is echter één uitzondering: in programma's die aan de hook H.TIMI hangen moet register A bewaard worden. Dit register is een kopie van het VDP statusregister S#0, welke door de BIOS routine wordt uitgelezen voordat de hook H.TIMI aangeroepen wordt. Na terugkeer van de hook gebruikt de BIOS routine de inhoud van dit register.

Eén van de randapparaten die een interrupt opwekt binnen de MSX is de Video Processor. Deze geeft een interrupt op het moment dat de laatste lijn van het actieve deel van het scherm – daar waar tekst en grafiek kunnen verschijnen – geschreven heeft. Aangezien dit 50 keer per seconde gebeurt, komen er dus 50 interrupts per seconde van de VDP. Als de VDP op 60 Hz wordt ingesteld, dan komen ook de interrupts 60 keer per seconde.

Afhandeling

De interrupt routine in het BIOS roept als eerste de hook H.KEYI aan. Daarna wordt gekeken of de interrupt van de VDP afkomstig is. Als dit niet zo is, dan wordt de interrupt routine beëindigd, anders wordt het tweede deel van de routine uitgevoerd.

Dit deel omvat het aanroepen van de hook H.TIMI en het afwerken van een aantal standaard activiteiten zoals het toetsenbord uitlezen, Basic's ON INTERVAL GOSUB en ON STRIG GOSUB instructie uitvoeren, de Basic variabele TIME verhogen, het PLAY statement afwerken en ga zo maar door. In het overzicht bij dit artikel is precies te zien welke taken de interrupt routine allemaal afwerkt.

Om te bepalen of een interrupt van de VDP afkomstig is, wordt er gekeken naar de inhoud van het al eerder genoemde VDP register S#0. Bit 7 van dit register wordt door de VDP gezet op het moment dat de laatste regel van het scherm opgebouwd is en de interrupt opgewekt wordt. Nadat het register is uitgelezen, wordt het bit automatisch op 0 gezet. Zolang dit bit op 1 staat, zal de VDP geen nieuwe interrupts meer genereren. Bij iedere VDP-interrupt moet dit register dus uitgelezen worden, anders genereert de VDP geen nieuwe interrupts meer.
Bij de aanroep van de hook H.TIMI staat in register A de inhoud van het zojuist uitgelezen register S#0. Na terugkeer van de hook wordt de inhoud opgeslagen in de systeem variabele STATFL. Om die reden moet de inhoud van register A dus bewaard worden door het programma dat aan de hook H.TIMI hangt.

Soorten

Interrupts kunnen door de Z80 – of natuurlijk de compatible R800 – processor op drie verschillende manieren worden afgehandeld, waarvan er op de MSX maar twee echt worden gebruikt. De manieren waarop de processor de interrupts af kan handelen heten Interrupt Mode's. Ze zijn genummerd van 0 tot en met 2. Een interrupt mode kan gekozen worden door middel van één van de machinetaal instructies IM 0, IM 1 of IM 2.
De IM 0 wordt binnen een MSX systeem niet gebruikt. In deze mode kan een randapparaat de CPU een instructie laten uitvoeren. De meeste gebruikte Interrupt Mode op MSX computers is IM 1. In deze mode springt de processor bij een interrupt altijd naar een vast adres: 0038h. Op dat adres staat in elk MSX BIOS een sprong naar de interrupt routine.

De andere wel voorkomende interrupt mode is IM 2, deze wordt op de MSX gebruikt onder CP/M. In deze mode wordt het aan te roepen adres bepaald door het I register en een byte van het randapparaat. Het I-register kan door het programma worden ingesteld en bevat de hoge helft van een adres. Het lage deel wordt door het randapparaat gegeven. Het zo gevormde adres is nog niet het adres van de interrupt routine zelf, maar geeft aan waar het beginadres van de routine in het geheugen staat. Omdat het beginadres van de routine altijd op een even geheugenplaats moet staan kunnen er op deze manier 128 verschillende interrupt routines worden aangeroepen, het randapparaat bepaald welke.

ONDERBREKEN EN ONDERBROKEN WORDEN

Globaal overzicht van de activiteiten van de MSX BIOS interrupt routine:
Altijd:
- Alle CPU registers stacken
- Aanroepen hook H.KEYI
- Uitlezen VDP statusregister S#0
- Turbo R: Controle op PAUSE toets
- Controle op interrupt van de VDP

Bij VDP interrupt:
- Aanroepen hook H.TIMI
- Enable Interrupts (EI)
- Opslaan S#0 in STATFL
- ON INTERVAL GOSUB
- Verhogen TIME variabele (JIFFY)
- Afhandelen PLAY

Iedere tweede VDP interrupt:
- Toetsenbord scan
- Verwerking van de toetsen
- ON STRIG GOSUB
- Toets repetitie

Altijd:
- Alle CPU registers herstellen
- Enable Interrupts (EI)
- RETurn from Interrupt (RETI)

Onder CP/M worden alle 128 mogelijke geheugen adressen gevuld met dezelfde waarde, waardoor alle interrupts alsnog bij dezelfde routine uitkomen. Voor de zekerheid hebben de schrijvers van CP/M de interrupt routine laten beginnen op een adres waarbij het hoge en lage adresdeel hetzelfde zijn. Als er dan een interrupt zou optreden waarbij het randapparaat een laag adresdeel geeft waarvan toch bit 0 gezet is wordt toch de juiste interrupt routine aangeroepen. Een adres waarvan bit 0 gezet is kan al snel voorkomen als het randapparaat helemaal geen adresdeel aangeeft, omdat het verwacht dat Interrupt Mode 1 actief is.

De interrupts kunnen door het programma voorkomen worden. Daartoe moet het programma de CPU melden dat de interrupts, die door de randapparaten opgewekt worden, niet afgehandeld mogen worden. De CPU heeft hiervoor twee vlaggen met de namen IFF1 en IFF2. IFF staat voor Interrupt Flip Flop. Als de CPU een interrupt herkend, dan wordt er eerst gekeken naar de vlag IFF1. Als deze gezet is, dan wordt de interrupt afgehandeld, anders wordt hij genegeerd.

De machinetaal instructies om deze vlaggen te beïnvloeden zijn Disable Interrupts (DI) en Enable Interrupts (EI). Door een DI instructie worden beide vlaggen gewist en zal de CPU de interrupts negeren. Na een EI instructie worden de vlaggen weer gezet en interrupts geaccepteerd.

Naast de hierboven genoemde interrupts is er ook nog een type interrupt dat niet door de CPU genegeerd kan worden. Dit is de zogenaamde Non Maskable Interrupt of NMI. Een NMI wordt op een soortgelijke manier behandeld als een interrupt in Interrupt Mode 1, alleen roept de CPU nu adres 0066h aan. Het MSX BIOS bevat – hoewel ze op de standaard MSX niet voorkomen – op dat adres een routine voor de afhandeling van NMI's.

Onder MSXDOS is alleen voorzien in een afhandelings routine voor de 'gewone' interrupts. Deze routine schakelt het BIOS aan in pagina 0 en roept daar de interrupt routine aan. Aan het eind wordt het RAM in pagina 0 weer teruggeschakeld. Non Maskable Interrupts mogen dus niet voorkomen onder MSXDOS omdat er dan geen afhandelings routine voor is. Adres 0066h bevindt zich midden in het eerste FCB – File Control Block – buffer van MSXDOS, die begint op adres 005Ch.

De aanroep

Voordat de interrupt routine wordt uitgevoerd wordt de Program Counter opgeslagen op de stack. Na beëindiging van de interrupt routine wordt dat adres weer teruggehaald en wordt het hoofdprogramma weer vervolgd. De interrupt routine wordt dus uitgevoerd alsof het hoofdprogramma een CALL naar een subroutine uitvoert, in dit geval de interrupt routine.
Om nesting te voorkomen worden de vlaggen IFF1 en IFF2 door de CPU zelf gereset (DI). Voor de aanroep van een NMI routine wordt IFF1 in IFF2 bewaard en wordt alleen IFF1 gereset, zodat de stand van IFF1 hersteld kan worden aan het einde van de NMI routine.

Interrupt routines kunnen, net als gewone subroutines, met een RET instructie worden beëindigd. De CPU kent echter twee speciale instructies om interrupt routines te beëindigen: RETurn from Interrupt (RETI) en RETurn from Non maskable interrupt (RETN).
De RETI instructie kan door een randapparaat herkend worden op het moment dat deze door de CPU uitgevoerd wordt, waarna het randapparaat weet dat zijn interrupt afgehandeld is. Het is dus beter een RETI te gebruiken dan een RET, terwijl de werking verder hetzelfde is. Merk op dat IFF1 niet gezet wordt door een RETI instructie. Dit moet door een EI instructie in de interrupt routine gedaan worden.
De RETN instructie aan het eind van een NMI-routine is wèl nodig omdat de stand van IFF1 hersteld moet worden. De RETN zorgt ervoor dat de inhoud van IFF2 wordt gekopieerd naar IFF1, waardoor deze weer hetzelfde is als voor de afhandeling van de NMI. Verder gedraagt een RETN instructie zich hetzelfde als een RET instructie.

Nog eens VDP interrupts

Op het moment dat de VDP de laatste beeldlijn van het actieve schermdeel geschreven heeft wordt bit 7 van het VDP register S#0 gezet. Tegelijkertijd wordt het interrupt signaal actief gemaakt en herkent de CPU de interrupt. Op dat moment worden automatisch de vlaggen IFF1 en IFF2 gereset, waardoor de CPU geen interrupts meer af zal handelen.

Het interrupt signaal van de VDP blijft echter net zolang actief tot het VDP statusregister S#0 uitgelezen wordt. Als er in de interrupt routine dus een EI wordt gegeven om interrupts weer toe te staan terwijl S#0 nog niet uitgelezen is, dan zal de CPU meteen de interrupt van de VDP weer herkennen (deze was immers nog actief) en meteen overgaan tot het uitvoeren van de interrupt routine, welke nog niet eens beëindigd was.

Het resultaat mag duidelijk zijn: een oneindige lus waarbij er door de CPU bij iedere nieuwe interrupt een terugkeeradres op de stack wordt gezet. Hierdoor zal

Adressenlijst

FD9Ah	hook H.KEYI
FD9Fh	hook H.TIMI
FDD6h	hook H.NMI
F3E7h	variabele STATFL
FC9Eh	variabele JIFFY
0038h	Binnenkomst interrupts in IM 1
0066h	Binnenkomst Non Maskable Interrupts

het hele geheugen binnen de kortste keer vol staan met terugkeeradressen in plaats van programmacode en de computer zal zeer waarschijnlijk vastlopen.
Naast de interrupt die de VDP opwekt als het actieve schermdeel is opgebouwd, kan de VDP nog een interrupt opwekken. Het moment waarop de VDP dit doet kan softwarematig worden ingesteld. Hiervoor is het VDP register R#19 beschikbaar. In dit register kan een Y-coördinaat van het scherm worden opgegeven. Op het moment dat de VDP deze regel bij het opbouwen van het scherm bereikt, wordt er een interrupt gegenereerd.

Programmeertips

Bij het schrijven van een routine die bij een interrupt moet worden aangeroepen moet er een juiste keuze worden gemaakt tussen de twee mogelijke hooks. Een routine aan de hook H.KEYI mag nooit een EI geven, omdat de BIOS interrupt routine dan het VDP statusregister S#0 nog niet heeft uitgelezen. Een routine aan de hook H.TIMI mag dit wel doen, maar moet de inhoud van register A bewaren.
De BIOS interrupt routine leest het VDP statusregister S#0 uit door direct de commando poort van de VDP uit te lezen en niet door eerst het juiste statusregister te selecteren via register R#15 en pas daarna de poort te lezen. Programma's moeten dus altijd register R#15 op 0 houden als de interrupts aan staan. Moet een programma toch een ander statusregister lezen, dan moeten de interrupts uitgezet worden voordat het register gelezen kan worden.
Het is mogelijk een eigen interrupt routine te schrijven, die de bestaande BIOS routine volledig vervangt. Houdt er in dat geval echter rekening mee dat alle registers bewaard moeten worden en vergeet niet het VDP statusregister S#0 uit te lezen. Aan de hand daarvan kan eventueel nog een keuze worden gemaakt tussen verschillende subroutines binnen de eigen interrupt routine.
Voor het beëindigen van een 'gewone' interrupt routine moeten interrupts weer worden toegestaan door middel van een EI instructie. Om nesting te voorkomen negeert de CPU de interrupts nog één instructie na de EI instructie. Op die manier kan met een RETI eerst worden teruggekeerd naar het hoofdprogramma en loopt de stack niet over bij een snelle opvolging van interrupts. Die worden namelijk pas nadat de RET uitgevoerd is weer herkend. De laatste twee instructies in een interrupt routine zijn dus altijd:

```
EI
RETI
```

In een NMI-routine mogen geen DI en EI instructies worden gebruikt, omdat deze ook de IFF2 vlag wijzigen. Deze vlag moet juist bewaard worden omdat IFF2 tijdens de uitvoering van een NMI routine een kopie van IFF1 bevat, die na de NMI weer hersteld moet kunnen worden.

Interrupt routines – en daarmee alle routines die aan de interrupt hooks hangen – mogen niet te lang duren. Als de VDP alweer een nieuwe interrupt opwekt voordat de door de voorgaande interrupt gestarte routine beëindigd is, wordt deze na een EI instructie meteen door de CPU herkend. Het hoofdprogramma krijgt op die manier geen enkele kans meer nog een instructie uit te voeren.

Bij het gebruik van een eigen interrupt routine in RAM moet er bij de aanroep van BIOS routines op gelet worden dat tijdens de uitvoer van die routines pagina 0 niet het RAM slot met de nieuwe routine op adres 0038h naar de eigen routine staat. In de plaats daarvan zal bij de eerste de beste interrupt de interrupt routine uit het BIOS actief worden. De vlag IFF1 moeten op dat moment dus altijd door middel van een DI instructie gereset zijn. Houd er ook rekening mee dat er een aantal BIOS routines zijn die zelf de interrupts aanzetten. Dit wordt zeker gedaan door de SUBROM, bij iedere entry in de jump table staat namelijk een EI instructie!

Voorbeeld programma toepassing Interrupt Mode 2

```
;
; im2.gen - RWi
;
; Voorbeeld van het gebruik van Interrupt Mode 2.
;
intVecTabAd   equ 08000h              ;Daar de Interrupt Vector Tabel zetten
intVecHalfAd  equ 080h                ;Hoge geheugen adres pointer
intRoutStart  equ 08181h              ;Daar de routine laten starten
intRoutHalfAd equ 081h                ;Adres hoog en laag

im2:          di                      ;Geen interrupts tijdens switch

              ld hl,intVecTabAd       ;Daar de IVT opbouwen
              ld (hl),intRoutHalfAd   ;Dit als hoog en laag adresdeel nemen
              ld d,h                  ;Destination pointer overnemen uit
              ld e,l                  ; de source pointer
              inc de                  ;Destination 1 byte verder
              ld bc,128*2             ;128 vectors, 1 byte extra voor 256e
              ldir                    ;Maak de tabel aan

              ld hl,intRoutHere       ;Routine voor IM 2
              ld de,intRoutStart      ;Daar de routine neerzetten
              ld bc,intRoutLen        ;Lengte van de routine
              ldir                    ;Kopieer de routine

              ld a,intVecHalfAd       ;Dit als hoge adresdeel gebruiken
              ld i,a                  ;Hoge adresdeel zetten
              im 2                    ;Schakel om naar IM 2

              ei                      ;Nu mogen de interrupts weer

loop:         jp loop                 ;Eindeloze lus

intRoutHere   equ $                   ;Hier staat de code nu

              org intRoutStart

intRoutIM2:   push hl                 ;Te wijzigen registers opslaan
              push af

              ld hl,(tellerIM2)       ;Aantal interrupts teller
              inc hl                  ;Een verhogen
              ld (tellerIM2),hl       ;En weer opslaan

              in a,(099h)             ;Lees S#0 uit
              and a                   ;Komt de INT van de VDP (b7=1 - True)
              jp p,notFromVDP         ;Nee = Keer terug

              ld a,l                  ;Lage teller deel
              out (098h),a            ;Zet dat op het scherm

notFromVDP:   pop af                  ;Herstel de gewijzigde registers
              pop hl
              ei                      ;Interrupts mogen weer
              reti                    ;Keer terug naar het hoofdprogramma

tellerIM2:    defw 1                  ;Aantal interrupts teller

intRoutLen    equ $-intRoutIM2        ;Lengte van de routine code

              end                     ;im2.gen
```

```
; TSRFRAME - MST TSR
;
; Tsr frame file for GEN80 or M80 assemblers
;

; Macro definitions
;
tsrName         macro
                  db "MST TsrFrame"         ;TSR ID-Name
                endm
;
memMan          macro @fnc                  ;Macro for making MemMan
                  ld e,@fnc                 ; function calls
                  call memManEntry
                endm


;---------------------------------------;
;    Header for the TSR file            ;
;---------------------------------------;
;
                db "MST TSR",13,10          ;TSR identifier
                tsrName                     ;ID-Name
                db 26                       ;^Z
                dw 0002                     ;Header file versie, MemMan 2.2
                dw base                     ;Code base address
                dw init                     ;Init address
                dw kill                     ;Kill address
                dw talk                     ;TsrCall entry
                dw tsrLen                   ;Programma code lengte
                dw iniLen                   ;Init code lengte


;---------------------------------------;
;    Start of TSR-program code          ;
;---------------------------------------;
;
extBio          equ 0ffcah                  ;EXTBIO hook
;
hook            equ 0fd9fh                  ;Dummy: Hook to be bend by this TSR
;
info            equ 50                      ;MemMan function
getMemManEnt    equ 6                       ;Info subfunction
getTsrID        equ 62
;

;
; TSR Base Address
;
base            equ $                       ;First byte of program code
;
kill:           ret                         ;No destruction routine
;
talk:           ret                         ;No driver routine
;
program:                                    ;Hook-processing program start
                                            ;Insert your own routine here

endProgram:     ex af,af'                   ;Save AF
                ld a,0                      ;Reset flags for TsrManager
                ex af,af'                   ;Restore AF, flags to A'
                ret                         ;Return to TsrManager
```

Source TSRFRAME

```
; memManEntry
;
; Entry address for MemMan function calls
;
memManEntry:  jp 0                      ;Address filled in by init routine

tsrLen        equ $-base                ;Length of TSR programcode


;---------------------------------------;
;     Init-code                         ;
;---------------------------------------;
;
init:         ld b,getMemManEnt         ;Ask for MemMan entry
              ld de,256*'M' + info      ;Call the MemMan info function
              call extBio               ;Through the ExtBio hook
              ld (memManEntry+1),hl     ;Save the MemMan entry address

              ld hl,tTsrName            ;Pointer to TSR name-string
              memMan getTsrID           ;See if this TSR already exists
              jr nc,initDouble          ; Yes, => Double installed error

              ld a,0                    ;Flags in A, no messages
              ret                       ;Init ready, return to TsrLoad

initDouble:   ld de,tDouble             ;DE=Text pointer
              ld a,3                    ;Flags for TL: Print text & Abort
              ret                       ;Return to TsrLoad

;
tTsrName:     tsrName                   ;Macro for tsr-name string
;
tDouble:      db 'This TSR is already ' ;Error text, 0-terminated
              db 'installed',13,10,10,0
;
iniLen        equ $-init                ;Length of init-code
;
;---------------------------------------;
;    Hook Table                         ;
;---------------------------------------;
;
hooks:        dw hokTabLen              ;Length of HokTab
              dw hook
              dw program
;
hokTabLen     equ $-hooks               ;Length of hook-table
;
;---------------------------------------;
;
              end                       ;TSRFRAME
;
```

```
*S 15, U +, R +, G 0, Q -

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                               ;
;       MSX2 Printer Buffer     ;
;                               ;
;            19/5/1989          ;
;            20/8/1989          ;
;            24/5/1990          ;
;            22/6/1990          ;
;                               ;
;      TSR Versie 13/4/1991     ;
;                               ;
;   (c) MSX Computer Magazine   ;
;                               ;
;   geschreven met TED editor   ;
;      M80/DevPac assembler     ;
;                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; TSR fileheader
;
        db      'MST TSR',cr,lf         ;TSR file ID, 8 chars
        db      'MJV printbuf',ctrlZ    ;TSR Name, 12 chars + ^Z
        dw      0001h                   ;Version number of headerfile, 1=MM 2.1
        dw      base                    ;Address of first Tsr program byte
        dw      init                    ;Initialisation address
        dw      kill                    ;Destruction address
        dw      drvEnt                  ;Driver / interaction address
        dw      tsrLen                  ; TSR programcode length
        dw      inilen                  ; Initialisationcode length


; A' : Flagbyte for TSR manager
;       Bit     Name            Function
;        0      quitHook        1 = DO NOT execute previous hook-buffer
;       1..7    reserved        always 0
;
quitHook        equ 0                   ;Quit this hook

; A  : Flagbyte for TSR loader
;       Bit     Name            Function
;        0      quitLoad        1 = Do not instal the TSR
;        1      introText       1 = Print a intro text
;       2..7    reserved        always 0
;
quitLoad        equ 0                   ;Quit TSR-Loader
introText       equ 1                   ;Print intro message

; Ascii codes
;
bel     equ     7
tab     equ     9
lf      equ     10
cr      equ     13
ctrlZ   equ     26

; I/O poorten; NIET gegarandeerd door de MSX standaard, maar ja, je moet wat...
;
lpt_st  equ     090H            ;Line printer status port
lpt_dw  equ     091H            ;Line printer data write port

; Systeem adressen
```

```
;
calSlt  equ  01ch               ;Interslot call
lptOut  equ  0a5h               ;Outputs a chr to the line printer
breakx  equ  0b7h               ;Ctrl-Stop controle
valTyp  equ  0f663h             ;Value type
dac     equ  0f7f6h             ;Decimal Accumulator
h_timi  equ  0fd9fh             ;Interrupt hook
h_cmd   equ  0fe0dh             ;CMD hook
h_attr  equ  0fe1ch             ;ATTR$ hook
h_lpto  equ  0ffb6h             ;Zend karakter naar printer hook
h_lpts  equ  0ffbbh             ;Printer-status hook
expTbl  equ  0fcc1h             ;Slot van BASIC ROM
mainRom equ  exptbl             ;Slot van MAIN ROM
extBio  equ  0ffcah             ;Extended BIOS hook

; MemMan funktiecodes
;
alloc         equ 4D00H+10      ;MemMan.Alloc
setRes        equ 4D00H+11      ;MemMan.SetRes
deAlloc       equ 4D00H+20      ;MemMan.DeAlloc
iniChk        equ 4D00H+30      ;MemMan.IniChk
info          equ 4D00H+50      ;MemMan.Info
getBasCall    equ 4             ;MemMan.Info.getBasicCall
fastGetCur    equ 5             ;MemMan.Info.fastGetCur
getID         equ 4D00H+62      ;MemMan.getTsrID
heapAlloc     equ 4D00H+70      ;MemMan.HeapAlloc
heapDeAlloc   equ 4D00H+71      ;MemMan.HeapDeAlloc


; Print Buffer constanten
;
pbtVer        equ 0101h         ;Versie 1.1
pbStackSize   equ 32            ;Interne stack, te klein voor interrupts!


; Basic Tokens
;
printToken    equ 91h
clearToken    equ 92h
freToken      equ 8fh


; MemMan aanroep macro
;
mMan    macro @fnc
          ld   de,@fnc          ;Funktie code in DE
          call extBio           ;Roep EXTBIO aan
        endm


base    equ $                   ;**** TSR Base-address


; doInt
;
; Hangt aan de interrupt hook en zendt tekens uit de buffer naar de printer
;
doInt:  di                      ;Is al -tig keer hiervoor gedaan waarschijnlijk
        push af                 ;Bewaar VDP statusbyte, komt van de Main-ROM
        ld   a,(leegFl)
        or   a                  ;Buffer leeg?
        jr   z,quitInt          ; Ja, => Niet printen
        in   a,(lpt_st)
        bit  1,a                ;Printer klaar?
```

```
        jr   nz,quitInt          ; Nee, => Naar oude hook

        ;Print teken uit buffer

outBuf: call getCur2             ;Bewaar huidige segmentcode van page 2
        ld   b,30                ;Maximaal 30 tekens printen
prtLoop:push bc                  ;Bewaar teller
        ld   a,(botSeg)          ;Haal oudste bufferssegement
        call mmOut               ;Inschakelen via MemMan
        pop  bc                  ;Herstel aantal nog af te drukken tekens
        ld   hl,(botPtr)
        ld   a,(hl)              ;Haal teken op
        out  (lpt_dw),a          ;Stuur teken naar printer data poort
        xor  a
        out  (lpt_st),a          ;Strobe even op 0 zetten
        dec  a
        out  (lpt_st),a          ; en weer op 1
makRoom:inc  hl
        bit  6,h                 ;Pagina top bereikt (adres 0c000h)?
        ld   a,(botSeg)          ;Haal huidige mapnr.
        jr   z,noTop
        ld   hl,(maxSeg)
        cp   l                   ; Hoogste map.nr?
        ld   hl,8000h            ;       Pointer naar begin van Page #2
        inc  a                   ;  Nee, => Volgende map
        jr   c,notMax
        xor  a                   ; Ja, begin weer bij laagste map
notMax: ld   (botSeg),a          ;Vul huidige map nr. in
noTop:  ld   (botPtr),hl         ; en pointer in deze map
        ld   de,(topSeg)
        cp   e                   ;topSeg=botSeg?
        jr   nz,next             ; Nee, => Druk volgende teken af
        ld   de,(topPtr)
        sbc  hl,de               ;Buffer=leeg?   (NC door CP E)
        jr   z,zetleeg
next:   in   a,(lpt_st)
        bit  1,a                 ;Printer al klaar?
        jr   z,next1             ; Ja, => Print volgende teken
        ld   hl,(waitMax)        ;[L] keer door de lus
outWait:in   a,(lpt_st)
        bit  1,a                 ;Printer nu eindelijk klaar?
        jr   z,next1             ; Ja, => Volgende teken
        dec  l                   ;Verlaag teller
        jr   nz,outWait          ;Teller niet nul, => Wacht langer
        jr   endWait             ;Te lang gewacht, => Stop
next1:  djnz prtLoop             ; Ja, => Print volgende teken
endWait:call rstCur2             ;Herstel Pagina 2 segment en originele stack
quitInt:sub  a                   ;Vlag voor MemMan: Volgende hook-buffer mag ook
        ex   af,af'              ; uitgevoerd worden, vlag komt in A'
        pop  af                  ;Herstel VDP status-byte
        ret                      ;Terug naar de manager
zetLeeg:xor  a
        ld   (leegFl),a          ;vlag: Niets meer te printen
        jr   endWait


; toBuf
;
; Hangt aan de H.LPTO hook, bewaart het teken [A] in de printerbuffer
;
toBuf:  di                       ;Wegens pointer-geklooi geen char's printen
        ex   af,af'              ;Bewaar karakter even
        ld   a,(buffFl)          ;Is er een buffer aanwezig?
        or   a
```

```
        jr   nz,bufFnd          ; Ja, => Karakter opslaan
        ex   af,af'             ; Nee, zet 0 in A', en herstel A
        ret                     ;Terug zonder quitHook
bufFnd: pop  ix                 ;Haal ret. adres van de manager
        ex   (sp),ix            ;Return adres naar BIOS ROM weghalen
        ex   af,af'             ;Karakter terug halen in A
        push hl                 ;Main reg's moeten bewaard blijven
        push de
        push bc
        push af

        ld   (lineBuf+1),a      ;Bewaar karakter in regelbuffer
        ld   a,1
        ld   (lineBuf+0),a      ;Lengte van de te bufferen data is 1 karakter

        call lnToBuf            ;Regelbuffer naar buffersegment

        pop  hl                 ;Haal het karakter in H
        ld   a,h                ;Herstel karaktercode, behoud vlaggenregister
        pop  bc                 ;Haal overige main registers
        pop  de
        pop  hl
        ex   af,af'             ;Bewaar AF even in AF'
        ld   a,1 .shl. quitHook ;Vlag voor MsmMan: Volgende hook-buffer mag
        ex   af,af'             ; niet uitgevoerd worden, vlag komt in A'
        ret                     ;Terug naar de manager, carry set indien ^Stop


; lnToBuf
;
; Verplaatst de data in LineBuf naar de printerbuffer
; Indien de printerbuffer vol is, zal gewacht worden tot er weer ruimte is
;
; Uit: Carry set indien onderbroken met ^Stop
;
lnToBuf:xor  a
        ld   (linePnt),a        ;Pointer in linebuffer = 1e karakter
        inc  a
        ld   (leegFl),a         ;Vlag: Buffer nog niet leeg

        call getCur2            ;Bewaar aktieve segment, aktiveer interne stack

bufLnLp:ld   hl,lineBuf         ;HL=Pointer naar lengtebyte van regel
        ld   a,(linePnt)        ;A=Pointer naar te bufferen karakter
        cp   (hl)               ;Hele regel gebufferd?
        jr   nz,chToBuf         ; Nee, => Plaats karakter in de buffer
        call rstCur2            ; Ja, => Herstel segment in pagina 2
        or   a                  ;  moet via een "call" wegens stack-switch
        ret                     ;Terug zonder carry, geen errors

chToBuf:ld   a,(topSeg)
        call mmOut              ;Schakel segment met vrije ruimte aan

        ld   hl,(linePnt)       ;Haal pointer in linebuffer
        ld   de,lineBuf+1       ;Pointer naar 1e databyte
        add  hl,de              ;Bereken adres van huidige karakter
        ld   a,(hl)             ;Haal het te bufferen karakter
        ld   hl,(topPtr)        ;Haal pointer naar buffertop
        ld   (hl),a             ;Vul het in teken in het buffersegment in
        inc  hl                 ;Verhoog pointer
        bit  6,h                ;Einde page 2 bereikt?
        ld   a,(topSeg)         ;Haal huidige mapnr.
        jr   z,nietVol
        ld   hl,(maxSeg)
```

```
        cp   l                  ;Hoogste seg.nr?
        ld   hl,8000h
        inc  a                  ; Nee, => Volgende seg
        jr   c,nietMax
        xor  a                  ; Ja, begin weer bij laagste seg
nietMax:ld   (topSeg),a         ;Vul map. nr in
nietVol:ld   (topPtr),hl        ;Vul bufferpointer in pagina in
        ld   de,(botSeg)        ;Haal laagst gebruikte mapnr. in E
        cp   e                  ;topSeg=botSeg?
        jr   nz,nextChr         ; Nee, => Nog minstens 1 segment vrij
        ld   de,(botPtr)
        sbc  hl,de              ;Topprt=Onderkant van laaste segment? (CP E=NC)
        jr   nz,nextChr         ; Nee, => Buffer nog niet vol, volgende kar.

        ; Wacht net zo lang totdat de interrupt routine een karakter uit
        ; de buffer heeft gehaald zodat er weer minstens n byte vrij is.
        ; Het wachten kan worden onderbroken door de ^Stop toetscombinatie

        call retCur2            ;Herstel segmentstand en originele stack

fulWait:di                      ;Interrupts uit tijdens toets-scan
        ld   ix,breakx          ;Test op ^Stop
        ld   iy,(mainRom-1)
        call calSlt             ;Routine aanroepen in Main ROM
        jr   c,ctrStop          ;Stop ingedrukt, => return met carry-vlag
        ei                      ;Interrupts aan zodat karakters afgedrukt
                                ; kunnen worden via de interrupt
        ld   hl,(topPtr)        ;Haal buffer-einde pointer
        ld   de,(botPtr)        ;En start-pointer
        sbc  hl,de              ;Gelijk?        (NC door breakx)
        jr   z,fulWait          ; Ja, => Nog steeds niets afgedrukt

        call getCur2            ;Schakel weer over op interne stack

nextChr:ld   hl,linePnt         ;HL=Pointer naar huidige karakter
        inc  (hl)               ;Naar volgende karakter
        jr   bufLnLp            ;Buffer het volgende karakter uit de linebuffer

ctrStop:xor  a
        ld   (leegFl),a         ;Zet "buffer is leeg" vlag
        scf                     ;Terug met Carry set want ^STOP is ingedrukt
        ret                     ;Segment & stack standen zijn reeds hersteld


; bufStat
;
; Geeft de printer status in [A] + Zero flag
;
bufStat:ex   af,af'             ;Bewaar AF
        ld   a,(bufFl)          ;Is er een buffer aanwezig?
        or   a
        ld   a,0                ;Indien niet, geen quitHook uitvoeren
        jr   z,exRet            ; Nee, => Terug naar BASIC ROM
        pop  ix                 ;Returnadres naar manager ophalen
        ex   (sp),ix            ;Returnadres naar ROM wegmikken
        ex   af,af'             ;Herstel AF
        sub  a                  ;[A] wordt 0
        dec  a                  ;[A] wordt -1 en Zeroflag wordt 0
        ex   af,af'             ;Bewaar AF
        ld   a,1 .shl. quitHook ;Vlag voor loader: Stop met deze hook
exRet:  ex   af,af'             ;Loader vlag moet in A'; herstel AF
        ret                     ;Vlag: Printer gereed
```

```
; cmd
;
; Handelt de CMD hook af
;
cmd:    push af                 ;Bewaar vlaggen
        push hl                 ;En de text pointer
        cp   clearToken         ;CLEAR statement?
        jr   z,clear            ; Ja, => Voer het uit
nextCmd:sub  a                  ; Nee, zet 0 in A'
        ex   af,af'             ;Zodat de manager de volgende TSR aan de
        pop  hl                 ; cmd hook kan uitvoeren
        pop  af
        ret
clear:  call chkPbT             ;Check of printbuf tokens aanwezig zijn
        pop  af                 ;Gooi oude textpointer
        ex   (sp),hl            ; en het oude karakter weg, bewaar nieuwe txt
        call leegBuf            ;Wis de buffer
        pop  hl                 ;Herstel nieuwe text pointer
        call chrGtR             ;Lees karakter dat achter CLEAR PRINTBUF staat
        jr   z,quitCmd          ;Er staat niets, => Leeg de buffer

        call evalInt            ;Er staat wel wat, dus lees getal in DE
        push hl                 ;Bewaar de text pointer
        call instDE             ;Installeer (DE) kB
        pop  hl                 ;Herstel de text pointer

quitCmd:pop  de                 ;Haal return adres naar de manager
        pop  af                 ;Verwijder returnadres naar "illegal fnc call"
        push de                 ;Herstel return adres naar de manager
        ld   a,1 .shl. quitHook ; want CMD-commando is uitgevoerd
        ex   af,af'             ;Die vlag moet in A'
        ret                     ;Via manager terug naar BASIC


; attr
;
; Verwerkt de ATTR$ functie
;
attr:   push af                 ;Bewaar vlaggen
        push hl                 ;En de text pointer
        call chrGtR             ;Haal volgende character na ATTR$
        cp   255                ;Volgt funktie voorloop token?
        jr   nz,nextCmd         ; Nee, => Volgende TSR
        call chrGtR             ;Haal funktie token
        cp   freToken           ;FRE token?
        jr   nz,nextCmd         ; Nee, => Volgende TSR
        call chkPbT             ;Kijk of PRINTBUF volgt op ATTR$
        pop  af                 ;Gooi oude textpointer
        ex   (sp),hl            ; en het oude karakter weg, bewaar nieuwe txt

        ld   hl,valTyp          ;Haal pointer naar type van DAC
        ld   (hl),2             ;Aantal kB buffer is een integer-type
        ld   hl,(kbInst)        ;Haal omvang van de buffer
        ld   (dac + 2),hl       ;Integers daar bewaren
        pop  hl                 ;Herstel de text pointer
        call chrGtR             ;HL moet wijzen naar kar. achter "PRINTBUF"
        jr   quitCmd            ;Commando uitgevoerd


; chkPbT
;
; Checkt of vanaf (HL) de basic tokens voor "PRINTBUF" staan
;
chkPbT: call chrGtR             ;Haal volgende token op
```

```
        cp   printToken          ;PRINT statement?
        jr   nz,noPbT            ; Nee, => Niet voor ons bestemd
        inc  hl
        ld   a,(hl)              ; Ja, test of er "CMD CLEAR PRINTBUF" staat
        cp   'B'
        jr   nz,noPbT            ; Nee, => Onbekend
        inc  hl
        ld   a,(hl)
        cp   'U'
        jr   nz,noPbT
        inc  hl
        ld   a,(hl)
        cp   'F'
        ret  z                   ;Return indien PRINTBUF aanwezig is
noPbT:  pop  hl                  ;Verwijder returnadres
        jp   nextCmd             ;En voer volgende TSR uit


; leegBuf
;
; Wist de printer buffer
;
leegBuf:di                       ;Geen karakters meer printen
        ld   hl,(topPtr)         ;Begin en einde van circulaire buffer
        ld   (botPtr),hl         ; gelijk maken om de buffer te wissen
        xor  a
        ld   (topSeg),a          ;Dus voor- en achterste mapper pointer
        ld   (botSeg),a          ; ook gelijk maken, op 0 dus nooit te groot
        ld   (leegFl),a          ;Zet de "buffer is leeg" vlag
        ret


; evalInt
;
; Evaluates an expression pointed to by HL.
; Result will be a 16 bit integer
;
; In: HL=Text pointer
;
; Out: DE=16 bit result integer
;      HL=Updated text pointer
;
evalInt:call frmEvl              ;Evaluate expression
        ld   de,(dac + 2)        ;Get it
        ld   a,(valTyp)
        cp   2                   ;Is it a integer?
        ret  z                   ;Yes, => Done
        push hl                  ; No, Save text pointer
        call frcInt              ;Convert DAC to integer in DAC+2
        ld   de,(dac + 2)        ;Get it
        pop  hl                  ;Text pointer in HL
        ret


; frmEvl
;
; Evaluates an expression pointed to by HL. Result is placed into DAC
;  and valTyp
;
frmEvl: ld   ix,4c64h            ;Formula evaluator
        jr   basicCall           ;To BASIC interpreter


; chrGtR
```

```
;
; This routine gets the character at address HL+1 into the [A] register.
;  For MemMan-testing purposes, an interslot-call to the BASIC interpreter
;  is made.
;
; Note that the inter-slot call to "chrGtR" is a waste of time, in most cases
;  a simple routine like "inc hl  ld a,(hl)  ret" would give the same
;  results, apart from the blank-skipping and the 'end-of-statement' detection.
;
; However, the inter-slot call method is recommend by the MSX2 Technical
;  Handbook from ASCII. Also note that "chrGtR" in the BASIC ROM
;  calls some hooks, so that future (MST?)-extensions can be connected to it.
;
chrGtR: ld ix,4666h             ;Address of chrGtR in BASIC-ROM
        jr basicCall            ;Let MemMan set-up stack-error hook
                                ; and call the BASIC-ROM


; frcInt
;
; Converts argument in DAC to integer
;
frcInt: ld ix,2f8ah             ;CINT is in page 0
                                ;Falls into BasicCall

; BasicCall
;
; This routine should be used to make an inter-slot call to the BASIC inter-
; preter. MemMan installs a routine to the H.STKE hook, so that the internal
; stacks of the TSR-Manager will be reset when a BASIC error occurs.
;
basicCall:
        call 0                  ;Addr will be installed by init-routine
        ei                      ;Re-Enable interrupts
        ret


; instDE
;
; Installeert een buffer van (DE) kB
;
instDE: push de                 ;Bewaar opgegeven aantal kB
        call killSeg            ;Geef de gebruikte segmenten terug
        pop  de                 ;Haal gewenst aantal kB
        ld   a,d                ;Is er "0" opgegeven?
        or   e
        ret  z                  ; Ja, => Buffer is verwijderd
        dec  de                 ;Reken aantal kilobytes om in aantal pagina's
        srl  d
        rr   e                  ; DE / 2
        srl  d
        rr   e                  ; DE / 4
        srl  d
        rr   e                  ; DE / 8
        srl  d
        rr   e                  ; DE / 16
        inc  e                  ; DE + 1
        ld   a,e                ;A is nu aantal pagina's
        ld   (seginst),a        ;Daar opslaan
                                ;Ga verder met zoeken van segmenten

; zoekMem
;
; Zoek memory
```

```
;
zoekMem:ld   de,mmTabel          ;[DE]=MMTABEL, beschikbare Mapper segmenten
        ld   bc,(segInst)        ;C=Aantal gewenste segmenten
        ld   b,0                 ;B=Huidige segmentteller
        inc  c
        dec  c                   ;C=0?
        jr   nz,memMan2          ; Nee, => Start installatie
        ld   c,128               ; Ja, vul maximum in, 128 segmenten (2 Meg.)

memMan2:push de                  ;Bewaar positie in segmenttabel
        push bc                  ;Bewaar huidige segment teller
        ld   b,2 .or. 11000000B ;Allocate segment exclusief op 8000H (PSEG)
memMan3:mMan alloc               ;Alloceer
        pop  bc
        pop  de
        ld   a,h
        or   l                   ;Segment nummer = 0?
        jr   z,memMan4           ; Ja, => geen vrije segmenten (meer)
        ex   de,hl
        ld   (hl),e              ;Bewaar segment code
        inc  hl
        ld   (hl),d
        inc  hl
        ex   de,hl
        push de                  ;Bewaar tabel entry
        push bc                  ;Segment teller
        mMan setRes              ;Set system status
        pop  bc
        pop  de
        inc  b                   ;Verhoog segmentteller
        ld   a,b
        cp   c                   ;Maximum bereikt?
        jr   nz,memMan2          ; Nee, => Volgende segment
memMan4:ld   a,b                 ;Aantal segmenten naar [A]
        ld   (buffl),a           ;Vlag:  0 indien minstens 1 seg gevonden
        ld   h,a                 ;Maak HL alvast 0
        ld   l,a
        or   a                   ;Nul pagina's?
        jr   z,memMan5           ; Ja, => 0 kB vrij; geen buffer aanwezig
        dec  a                   ; Nee, maak aantal 0-based
        ld   (maxSeg),a          ;Vul aantal geheugen pagina's in
        inc  a
        ld   l,a                 ;Bereken hoeveel kB dat is
        xor  a
        sla  l                   ;Map. nr * 16 = kilobytes free
        rla                      ;*2
        sla  l
        rla                      ;*4
        sla  l
        rla                      ;*8
        sla  l
        rla                      ;*16
        ld   h,a                 ;HL=Buffer geheugen in kilobytes
memMan5:ld   (kbInst),hl         ;Zo groot is de buffer nu
        ret


; mmOut
;
; Schakel segment [A] in
;
; Wijzigt: Alle registers
;
mmOut:  ld   b,a                 ;Bewaar A in B
```

```
        ld   a,(curMap)          ;Haal huidige segment stand
        cp   b                   ;Staat het goede segment er al?
        ld   a,b                 ;       Bewaar het goede segment nummer in A
        ret  z                   ; Ja, => Terug
        ld   (curMap),a          ; Nee, bewaar huidige segment nummer

        ; Schakel segment via MemMan

        ld   de,mmTabel          ;Haal segment nummer uit tabel
        ld   hl,(curMap)         ;Haal segmentnummer (= tabel index)
        add  hl,hl               ;*2 bytes per segmentcode
        add  hl,de
        ld   a,(hl)              ;Haal segmentcode uit tabel
        inc  hl
        ld   h,(hl)
        ld   l,a                 ;Segement nr. in HL
        jr   fastUse2            ;Roep MemMan.Use2 aan


; getCur2
;
; Schakelt over op interne stack in pagina 3 zodat pagina 2 veilig
; weggeschakeld kan worden.
;
; Deze funktie mag niet recursief aangeroepen worden.
;
; Vraagt de huidige segmentcode van pagina 2 aan MemMan
; Dit mag via de 'gewone' FastGetCur omdat de manager netjes de huidige
; segmentcode berekent en in de adminstratie van GetCur opslaat, voordat er
; ook maar n TSR aangeroepen wordt.
;
; In: Niets
;
; Uit: HL=Huidige Segmentcode
;
getCur2:di                       ;Geen interrupts in eigen (te kleine) stack
        pop  hl                  ;Haal return-adres
        ld   (savSP),sp          ;Bewaar huidige stackpointer
        ld   sp,(pbStack)        ;Schakel over op stack in pagina 3
        push hl                  :Plaats het return-adres terug

        ld   b,2                 ;Haal curseg van Page 2
gtCurAd:call 0                   ;Adres wordt ingevuld door INIT-code
        ld   (oldSeg2),hl        ;Bewaren om straks te kunnen herstellen
        ld   a,-1                ;Aangeven dat het data-segment nog niet
        ld   (curMap),a          ; aangeschakeld is
        ret


; rstCur2
;
; Herstelt het segment in pagina 2 en de originele stack
;
; In/Uit: Niets
;
rstCur2:ld   hl,(oldSeg2)        ;Haal originele segment in pagina 2
        call fastUse2            ;Schakel dat weer in
        pop  hl                  ;Haal return-adres
        ld   sp,(savSp)          ;Herstel de originele stackpointer
        jp   (hl)                ;Spring terug naar het return-adres


; FastUse2
;
```

```
; Schakelt een segment in in pagina 2 via MemMan
;
; In:  HL=Segcode
; Uit:  A=Errorcode
;
fastUse2:jp  0                  ;Adres wordt ingevuld door INIT-code


; drvEnt
;
; Entry voor interactie met toepassings programma's
;
; In: [A] = Funktie code
;       0  = Geef PB versie-nummer in HL (nu: H=1 L=1  = Versie 1.1)
;       1  = Leeg de printerbuffer
;       2  = Installeer [HL] kB
;       3  = Geef de huidige omvang van de buffer in kB in [HL]
;       4  = Plaats een datablok in de printer buffer. De data moet in
;            geheugenpagina 2 of 3 staan (tussen &h8000 en &hffff)
;            In:  HL=1e byte van de data
;                 DE=Aantal databytes
;            Uit: Carry set indien onderbroken
;
drvEnt: ld   b,d                ;Bewaar parameter DE in BC (flush lengte)
        ld   c,e
        ex   de,hl              ;Bewaar parameter HL in DE
        or   a                  ;Funktie 0
        ld   hl,pbtVer          ;PB versie # H.L
        ret  z
        dec  a                  ;Funktie 1
        jp   z,leegBuf          ;Leeg de buffer
        dec  a                  ;Funktie 2
        jp   z,instDE           ;Installeer DE kilobytes
        dec  a                  ;Functie 3
        ld   hl,(kbInst)        ;Geef omvang van buffer in [HL]
        ret  z
        dec  a                  ;Functie 4
        jr   z,flushDE          ;Copieer string naar buffer
        ret                     ;Klaar


; flushDE
;
; Pomp een datastring naar de printerbuffer
;
; In: DE=Pointer naar de data
;     BC=Lengte van de data
;
; Uit: Carry set indien onderbroken
;
flushDE:bit  7,d                ;Kijk of de data op adres 8000h of hoger staat
        jr   nz,tstBuf          ; Ja, => Verplaats de data
        ld   de,ePage           ; Nee, zet een foutmelding op de printer
        ld   bc,errLen          ;Lengte in BC

tstBuf: ld   a,(buffl)          ;Is er een buffer aanwezig?
        or   a
        jr   z,toLpt            ; Nee, => Direct naar printer

flushLp:ld   h,b                ;HL = Aantal databytes
        ld   l,c
        ld   bc,255             ;Kijk of er minstens 255 databytes zijn
        or   a
        sbc  hl,bc
```

```
        jr   nc,copyDta         ;Genoeg bytes, => 255 bytes naar regelbuffer
        add  hl,bc              ;Minder dan 255 bytes, herstel dat aantal
        ld   a,h
        or   l                  ;Clear carry, test op 0
        ret  z                  ;Nul bytes, => Klaar
        ld   b,h                ;Aantal bytes te bufferen bytes naar BC
        ld   c,l
        ld   hl,0               ;Hierna geen bytes meer over

copyDta:push hl                 ;Bewaar 'overschot'
        ld   hl,lineBuf         ;Pointer naar de regelbuffer in pagina 1
        ld   (hl),c             ;Bewaar lengte van de regel
        inc  hl                 ;Pointer naar naar 1e databyte in regelbuffer
        ex   de,hl              ;Databronadres naar HL, Regelbuffer naar DE
        ldir                    ;Verplaats de string naar regelbuffer
        push hl                 ;Bewaar nieuwe bronadres
        call lnToBuf            ;Verplaats regelbuffer naar printerbuffer
        pop  de                 ;Bronadres terug in DE
        pop  bc                 ;Aantal bytes in BC
        jr   nc,flushLp         ;Niet onderbroken, => Verplaats de overige data
        ret                     ;Terug met carry indien onderbroken


; toLpt
;
; Stuurt data rechtstreeks naar de printer
;
; In:   DE=Start van data
;       BC=Lengte
;
; Out:  Carry set indien onderbroken met ^Stop
;
toLpt:  ld   a,b
        or   c                  ;Nul karakters?
        ret  z                  ; Ja, => Klaar, NC
        ld   a,(de)             ; Nee, haal kar.
        ld   ix,lptOut          ;Stuur naar printer
        ld   iy,(mainRom-1)
        call calSlt             ;Routine aanroepen in Main ROM
        ret  c                  ;Carry set indien onderbroken
        inc  de                 ;Volgende kar.
        dec  bc
        jr   toLpt


; kill
;
; Routine die de Heap en Printer Buffer segmenten de-activeert
;
kill:   ld   hl,(heapPnt)       ;Haal heap pointer voor deAlloc
        mMan heapDeAlloc        ;Geef heap-geheugen weer vrij

killSeg:call leegBuf            ;Alle datapointer terugstellen
        ld   hl,bufFl           ;Pointer naar "Buffer aanwezig" vlag
        ld   a,(hl)             ;Haal "Buffer aanwezig" vlag
        or   a                  ;Is er wel een buffersegment?
        ret  z                  ; Nee, => Klaar
        ld   (hl),0             ; Ja, dan is dat vanaf nu verleden tijd
        ld   a,(maxSeg)         ;Haal het aantal segmenten
        inc  a                  ;1-based maken
        ld   b,a                ;B=Teller
        ld   hl,mmTabel         ;Segmenttabel
freeLp: ld   e,(hl)             ;Haal segmentcode in DE
        inc  hl
```

```
        ld   d,(hl)
        inc  hl
        push hl                 ;Bewaar segment-tabel pointer
        push bc                 ;Bewaar teller
        ex   de,hl              ;Segment-code naar HL
        mMan deAlloc            ;Geef segment terug aan memman
        pop  bc                 ;Herstel teller
        pop  hl                 ;Herstel tabel pnt.
        djnz freeLp             ;Eventueel volgende segment teruggeven
        ld   hl,0               ;Er is nu 0 kB genstalleerd
        ld   (kbInst),hl
        ret                     ;Klaar

        ;Foutmeldingen gebruikt door de TSR-code

ePage:  db   bel,'PB TSR error: Data address below &H8000',cr,lf
errLen  equ  $-ePage            ;Lengte van de foutmelding

        ;Variabelen gebruikt door de TSR programma code

botPtr: dw   8000H              ;Adres van eerst geplaatste byte
topPtr: dw   8000H              ;Eerste vrije byte
waitMax:db   20                 ;Aantal tellen wachten op printer busy
segInst:db   4                  ;4 * 16k = 64 kB buffer gewenst
bufFl:  db   1                  ;0 indien er minstens 1 buffersegment is
kbInst: dw   0                  ;Grotte van de buffer in kB
curMap: dw   0                  ;Huidig pagina 2 segment (0..254)
oldSeg2:dw   0                  ;Oude pagina 2 mapper segment-code
mmTabel:ds   256,0              ;128 memMan segment-codes
savSp:  dw   0                  ;Originele stackpointer na hook-aanroep
pbStack:dw   0                  ;Top van interne stack op de heap (in pagina 3)
heapPnt:dw   0                  ;Start van interne stack op de heap
leegFl: db   0                  ;Vlag=0 als buffer leeg is
botSeg: db   0                  ;Memmap. pagina van BOTPTR
topSeg: db   0                  ;Memmap. pagina van TOPPTR
maxSeg: db   0                  ;Hoogste memmap. segment
bezig:  db   0                  ;0 indien TSR al bezig is

linePnt:dw   0                  ;Pointer in de regelbuffer (highbyte=altijd 0)
lineBuf:ds   1+255,0            ;Regelbuffer, 1 lengtebyte + 255 databytes

tsrLen  equ  $-base             ;Lengte van de TSR programmacode

; init
;
; initialisatie code
;
init:   ld   (initSp),sp        ;Voor als er errors optreden

        ld   hl,tPbNaam         ;Pointer is naar PB naamstring
        mMan getID              ;Kijk of deze TSR al bestaat
        jp   nc,pbDouble        ; Ja, => Error

        ld   b,2                ;Vraag adres van fastUse2
        mMan info               ; aan MemMan
        ld   (fastUse2 + 1),hl  ;Jump-adres invullen in TSR-code

        ld   b,fastGetCur
        mMan info               ;Vraag om adres van fast get current segment
        ld   (gtCurAd+1),hl     ;Call-adres invullen

        ld   b,getBasCall       ;Ask for Basic-call address
        mMan info
        ld   (basicCall+1),hl   ;Save that addr.
```

```
        ld   hl,pbStackSize     ;Allocate this many bytes stack in page 3
        mMan heapAlloc          ;Locate the stack in the MemMan heap
        ld   a,h
        or   l                  ;Allocation succeeded?
        jr   z,heapErr          ; No, => Start yelling

        ld   (heapPnt),hl       ;Save heap pointer for deAlloc

        ld   de,pbStackSize     ;Calculate top of pbStack
        add  hl,de
        ld   (pbStack),hl       ;Stack pointer at top of stack

        call zoekMem            ;Zoek memory-segmenten
        ld   hl,(kbInst)        ;HL=Buffer geheugen in kilobytes
        call deci               ;Decimale waarde naar tekstbuffer
        ld   de,tIntro
        ld   a,1 .shl. introText;Vlag voor de loader: Intro tekst in DE
        ret                     ;Terug naar de loader

        ;Afdruk routines

pbDouble:
        ld   a,cr               ;Plak "PB dubbel geinstalleerd" tekst
        ld   (tIntEnd),a        ; achter de intro-tekst
        ld   de,tIntro          ;Geen segmenten meer over
        jr   prtEnd             ;Afdrukken en nokken

heapErr:ld   de,tHeap           ;DE=Pointer naar error text

prtEnd: ld   sp,(initSp)        ;Herstel stack pointer
        ld   a,(1 .shl. quitLoad) .or. (1 .shl. introText)
        ret                     ;Vlag voor Loader: Niet installeren, druk text

        ;Plaats HL als decimaal getal in de tFree buffer

deci:   ld   de,tFree
        ld   b,5
        ld   a,' '
dc1:    ld   (de),a             ;Wis de buffer met spaties
        inc  de
        djnz dc1
dc2:    ld   e,0
        ld   b,16
        or   a
dc3:    rl   l
        rl   h
        rl   e
        ld   a,e
        sub  10
        ccf
        jr   nc,dc4
        ld   e,a
dc4:    djnz dc3
        rl   l
        rl   h
        ld   a,e
        add  a,'0'
        push hl
        ld   hl,tFree+3
        ld   de,tFree+4
        ld   bc,4
        lddr
        ld   (tFree),a          ;Vul teken vooraan in de buffer in
```

```
        pop  hl
        ld   a,h
        or   l
        jr   nz,dc2
        ret

        ;Initialisatie variabelen

initSp: dw   0                  ;Stack-pointer van TSR-Loader


        ;Schermteksten

tIntro: db   "   MSX Computer Magazine's",cr,lf
        db   '     Printer Buffer TSR',cr,lf
        db   '      13/4/91 - by MJV',cr,lf,lf
tFree:  db   '    0 kB buffer installed',cr,lf,lf
tIntEnd:db   0
tDouble:db   bel,'Printer buffer already installed.',cr,lf,0
tHeap:  db   bel,'Not enough heap-memory available.',cr,lf
        db   'PB needs 32 bytes heap-memory.',cr,lf
        db   'Use CFGMMAN to install more heap-memory.',cr,lf,lf,0
tPbNaam:db   'MJV printbuf'

iniLen  equ $-init              ;Lengte van init-code


; hokTab
;
; Af te buigen hooks
;
hokTab: dw   endHT-$            ;HookTabel lengte

        dw   h_timi             ;Interrupt hook
        dw   doInt              ;Daarheen

        dw   h_cmd              ;CMD hook
        dw   cmd

        dw   h_attr             ;ATTR$ hook
        dw   attr

        dw   h_lpt0             ;Line Printer Out hook
        dw   toBuf              ;Hierheen

        dw   h_lpts             ;Line Printer Status
        dw   bufStat            ;Bestemming

endHT   equ $                   ;Hook-Tab einde

        end                     ;PB
```

```
*s 15, U +, G 0, Q -, B 5
;
; PRINT versie 1.1
;
; MSX-DOS toepassing voor de "MJV Printbuf" TSR.
;
; 7 april 1991 - door Ries Vriend
;
; (c) MSX Computer Magazine 1991
;

; systeem entries
;
bDos            equ 5
cmdFCB          equ 5ch
extbio          equ 0ffcah

; Ascii codes
;
tab             equ  9
lf              equ 10
ff              equ 12
cr              equ 13

; MSX-DOS1 funktie codes
;
conOut          equ 02h
inNoE           equ 08h
strOut          equ 09h
fOpen           equ 0fh
fClose          equ 10h
sFirst          equ 11h
sNext           equ 12h
setDta          equ 1ah
rdBlk           equ 27h

; MSX-DOS2 funktie codes
;
fFirst          equ 40h
fNext           equ 41h
open            equ 43h
close           equ 45h
read            equ 48h
parse           equ 5bh
pFile           equ 5ch
term            equ 62h
dosVer          equ 6fh

; memMan funktie codes
;
iniChk          equ 04d00h + 30
info            equ 04d00h + 50
info_callTsr    equ 3
getTsrID        equ 04d00h + 62
callTsr         equ 04d00h + 63

; Printer buffer funkties
;
pbGetVer        equ 0
pbClear         equ 1
pbInstal        equ 2
pbGetFre        equ 3
pbFlush         equ 4
```

```
; Byte offsets in de 8-bit variabelen tabel (iy)
;
parseFl         equ 0                   ;Command line switches
fileFl          equ 1                   ;File status flags
sysFl           equ 2                   ;System status flags
fileHandle      equ 3                   ;DOS2 filehandle

; Bit offsets van command line switches in iy+parseFl
;
swClear         equ 0                   ;/c  Clear de buffer
swDestroy       equ 1                   ;/d  Geen vraag "Zeker weten? (j/n)"
swInstall       equ 2                   ;/i  Installeer nn kB
swVersion       equ 3                   ;/v  Show PB version
swSize          equ 4                   ;/s  Show PB size
swFormfeed      equ 5                   ;/f  Add formfeed
swHidden        equ 6                   ;/h  Print hidden files

; Bit offsets van file vlaggen in iy+fileFl
;
first           equ 0                   ;True indien naar 1e file gezocht wordt

; Bit offsets van systeem vlaggen in iy+sysFl
;
dos2            equ 0                   ;True indien DOS2 geboot


; Constanten
;
eos             equ "$"                 ;End of string character
dataBuf         equ 08000h              ;Adres voor databuffer
stackSize       equ 150                 ;Stack grootte


; MemMan aanroep macro
;
memMan          macro @fnc
                  ld de,@fnc
                  call callMemMan
                endm


; Start entry
;
print:          ld de,tIntro            ;Toon introtekst
                call printLine

                sub a                   ;Controle-byte op 0 zetten
                memMan iniChk           ;Roep init-routine van MemMan aan
                cp 'M'                  ;Gelukt?
                jp nz,noMemMan          ; Nee, => Toon foutmelding

                ld a,d                  ;Versie 2.x of hoger?
                cp 2
                jp c,noMemMan           ; Nee, => Fout melding
                jr nz,mmVersionOk       ;Versie 3.x of hoger => Ok
                ld a,e
                cp 1                    ;Versie 2.1 of hoger?
                jp c,noMemMan           ; Nee, => Verouderde MemMan versie

mmVersionOk:    ld hl,pbNaam            ;Hier staat de naam van de printerbuf.
                memMan getTsrID         ;Vraag om een ID-nummer
                jp c,noPb               ; Mislukt => Toon foutmelding
```

```
                ld (pbID),bc            ;Bewaar de ID-code

                ld b,info_callTsr       ;Haal aanroepadres van de "CallTsr"
                memMan info             ; functie
                ld (callTsrAddr),hl     ;Bewaren voor latere aanroep

                ld a,pbGetVer
                call callPB             ;Haal versienummer van PB.TSR
                ld a,h                  ;Versie 2.x of hoger?
                cp 2
                jr nc,pbVersionOk       ; Ja, => Ok
                ld a,l
                cp 1                    ;Versie 1.1 of hoger?
                jp c,noPB               ; Nee, => Verouderde PB versie

pbVersionOk:    ld c,dosVer
                call dos                ;Vraag MSX dos versie op
                ld a,b
                cp 2                    ;Dos 2?
                jr c,noDos2             ; Nee, => DOS 1
                set dos2,(iy+sysFl)     ; Ja, zet een vlaggetje

noDos2:         ld hl,(6)               ;Haal TPA-einde
                ld sp,hl                ;Stack daar plaatsen
                ld de,-dataBuf-stackSize;Start van databuffer en stack grootte
                add hl,de               ; er vanaf trekken
                ld (bufSize),hl         ;Bewaar de bufferlengte

                call parseParms         ;Ontleed de schakelaars in de cmd regel
                call execOptions        ;Voer de gekozen opties uit

                set first,(iy+fileFl)   ;Open de 1e file

bufferFilesLp:  call openFile           ;Open het te printen bestand of stop
                call flushFile          ;Plaats het bestand in de buffer
                call closeFile          ;Sluit het bestand
                jr bufferFilesLp        ;Print het volgende bestand


; parseParms
;
; Ontleed de commandoregel
;
parseParms:     ld hl,80h               ;Start van de regel
                ld c,(hl)               ;Aantal bytes
                ld b,h                  ; in BC
                inc hl                  ;Naar 1e databyte

parseLp:        inc c                   ;Nul bytes doorzoeken?
                dec c
                ret z                   ; Ja, => Snel klaar

                ld a,'/'                ;Slash gaat vooraf aan een 'switch'
                cpir                    ;Zoek naar een switch
                ret nz                  ;Return indien niet gevonden
                ret po                  ;Return indien geen kar na de '/'

                ld a,(hl)               ;Haal karakter na de switch
                or 00100000b            ;Force lower case letter

                cp 'c'                  ;Clear buffer?
                jr nz,parse_noClear     ; => Nee
                set swClear,(iy+parseFl); Ja, zet vlag
```

```
parse_noClear:cp 'd'                    ;Delete without prompt?
              jr nz,parse_noDelete
              set swDestroy,(iy+parseFl)

parse_noDelete: cp 'f'                  ;Add formfeed?
              jr nz,parse_noFeed
              set swFormfeed,(iy+parseFl)

parse_noFeed: cp 'h'                    ;Also load hidden files?
              jr nz,parse_notHidden
              set swHidden,(iy+parseFl)

parse_notHidden:cp 'i'                  ;Install nn kB?
              jr nz,parse_noInstall
              set swInstall,(iy+parseFl)
              ld (installPnt),hl        ;Bewaar text pointer naar aantal kB
              set swSize,(iy+parseFl)   ;Na installatie, aantal kB tonen

parse_noInstall:cp 's'                  ;Show buffer size?
              jr nz,parse_noSize
              set swSize,(iy+parseFl)

parse_noSize: cp 'v'                    ;Show version?
              jr nz,parse_noVersion
              set swVersion,(iy+parseFl)

parse_noVersion:jr parseLp              ;Ontleed volgende switch


; execOptions
;
; Voer de in de commandoregel gekozen opties uit
;
execOptions:  bit swVersion,(iy+parseFl)
              call nz,showVersion       ;Toon versie nummer
              bit swClear,(iy+parseFl)
              call nz,clearBuf          ;Leeg de buffer
              bit swInstall,(iy+parseFl)
              call nz,installBuf        ;Installeer nn kB
              bit swSize,(iy+parseFl)
              call nz,showSize          ;Toon buffer grootte
              ret


; showVersion
;
; Toon het versienummer van de printerbuffer
;
showVersion:  ld de,tVersion1           ;Intro regeltje
              call printLine
              ld a,pbGetVer             ;Vraag versie nummer op
              call callPB               ;Roep PB aan
              ld a,h
              add a,'0'                 ;Hoge versie nummer
              ld (verHigh),a            ;Karakter opslaan
              ld a,l
              add a,'0'
              ld (verLow),a             ;Lage deel ook
              ld de,tVersion2           ;Uittro
              jp printLine


; clearBuf
;
```

```
; Leeg de buffer
;
clearBuf:       call destroyPrompt      ;Verwijderen - zeker weten? (j/n)
                ret c                   ; Nee, => Niet legen
                ld a,pbClear            ;PB funktie 1: leeg buffer
                call callPB             ;Roep de PB aan
                ld de,tClear
                jp printLine


; installBuf
;
; Installeer de buffer
;
installBuf:     call destroyPrompt      ;Verwijderen - zeker weten? (j/n)
                ret c                   ; Nee, => Niet legen
                ld de,(installPnt)      ;Pointer naar 'i' in "/i" in cmd line
                inc de                  ;Pointer naar aantal kB
                call decToBin           ;Converteer naar binair getal in HL
                ld a,pbInstal           ;Installeer HL kB
                call callPB             ;Roep PB aan
                ret

; Show size
;
; Toon buffergrootte
;
showSize:       ld a,pbGetFre           ;Vraag vrije ruimte op
                call callPB             ;Roep PB aan, vrije ruimte naar [HL]
                call deci               ;Zet decimale waarde van HL in tFree
                ld de,tFree             ;En toon die waarde
                jp printLine


; destroyPrompt
;
; Vraag: Destroy all data?
;
; Uit: NC=Yes
;      C=No
;
destroyPrompt:or a                      ;Terug met 'Yes' indien /D gegeven is
                bit swDestroy,(iy+parseFl)
                ret nz                  ; /D = Retun met carry clear
                ld de,tDestroy          ;Toon vraag
                call printLine
getYesNoLp:     ld c,inNoE              ;Lees character, no echo
                call dos
                and 11011111b           ;Convert to upper case
                cp 'Y'
                jr z,printAnswer        ;Return NC if 'Yes'
                cp 'N'                  ;No?
                jr nz,getYesNoLp        ; => Unknown answer
                scf                     ;Return Carry if 'No'
printAnswer:    push af
                ld (tYesNo),a           ;Put character in text buffer
                ld de,tYesNo            ;Print answer
                call printLine
                pop af
                ret


; openFile
;
```

```
; Open het via de commandoregel opgegeven bestand
;
openFile:       ld a,(cmdFCB+1)         ;Haal le karakter van filename
                cp ' '                  ;Geen naam opgegeven?
                jp z,noFileName         ; Ja, => Toon hulp

                bit dos2,(iy+sysFl)     ;DOS2 aktief?
                jr z,openFCB            ; Nee, => DOS1 fcb

                bit first,(iy+fileFl)   ;First file?
                jr z,findNextFIB        ; Nee, => Volgende file info block

                ld de,80h               ;Naam staat op de commandline
srcPathStartLp: inc de                  ;Skip lengtebyte
                ld a,(de)               ;Zoek begin van pathname
                or a                    ;Einde van cmd line?
                jp z,noFileName         ; Ja, => Stop
                cp ' '                  ;Skip blanks
                jr z,srcPathStartLp
                cp tab
                jr z,srcPathStartLp

                push de                 ;Bewaar start van pathname
                ld bc,parse             ;Parse pathname, geen VOL-ID
                call dos                ;Zoek einde van pathname
                sub a
                ld (de),a               ;Vul daar een 0-karakter in

                pop de                  ;Herstel pathname string
                ld ix,fib               ;Bestemming voor file info block
                ld bc,fFirst            ;Find first, geen hidden files
                bit swHidden,(iy+parseFl);Ook verborgen bestanden laden?
                jr z,firstNotHidden
                ld b,2                  ; Ja, zet hidden attribuut
firstNotHidden: call dos                ;Maak een FIB aan
                jp nz,notFoundErr       ; Fout, => Toon foutmelding
                jr fibFound             ;Open het FIB

findNextFib:    ld ix,fib               ;Bron+Bestemming file info block
                ld c,fNext              ;Find next
                call dos                ;Maak volgende FIB aan
                jp nz,backToDos         ; Fout, => Klaar

fibFound:       res first,(iy+fileFl)   ;Volgende file is niet meer de le
                ld de,fib               ;Deze file openen
                ld a,1                  ;No write
                ld c,open
                call dos                ;Open fib
                jp nz,openErr
                ld (iy+fileHandle),b    ;Bewaar filehandle
                jp printCurFile         ;Toon naam van huidig bestand

; openFCB
;
; Open een DOS1 File Control Block
;
openFCB:        ld c,setDta             ;Open een DOS1 FCB
                ld de,fcb
                call dos

                ld de,cmdFCB            ;fcb met de le naam uit de commandline
                ld c,sFirst             ;Search for first file funktie
                bit first,(iy+fileFl)   ;First file?
                jr nz,searchFile        ; Ja, => Zoek ham
```

```
                inc c                   ; Nee, search next file
searchFile:     call dos                ;Zoek de file
                or a                    ;Gelukt?
                jr z,fileFound          ; Ja, => Open hem
                bit first,(iy+fileFl)   ;Was dit de 1e file?
                jp nz,notFoundErr       ; Ja, => Toon foutmelding
                jp backToDos            ; Nee, => klaar
fileFound:      res first,(iy+fileFl)   ;Volgende file is niet meer de 1e
                ld c,fOpen              ;Open file (DOS1)
                ld de,fcb               ;FCB address
                call dos                ;Open het bestand
                or a                    ;Gelukt?
                jp nz,openErr           ; Nee, => toon foutmelding
                ld h,a
                ld l,a
                ld (fcb+0ch),a          ;Extentbyte moet 0 zijn
                ld (fcb+21h),hl         ;Current block=0
                ld (fcb+23h),hl
                inc hl
                ld (fcb+0eh),hl         ;Blocksize = 1 byte
                jp printCurFile         ;Toon naam van huidig bestand


; flushFile
;
; Leest de data uit de text file in de buffer.
;
flushFile:      ld de,dataBuf           ;Buffer waarin de tekst gelezen wordt
                ld hl,(bufSize)         ;Omvang van de buffer
                call readFile           ;Lees HL bytes in de databuffer
                ld a,h                  ;Nul bytes gelezen?
                or l
                jr z,addFormFeed        ; Ja, => Einde van file bereikt

                ld d,h                  ;Zoveel bytes staan er in de
                ld e,l                  ; buffer
                ld hl,dataBuf           ;Data staat daar
                ld a,pbFlush            ;Verstuur data naar buffer
                call callPB             ;Roep PB aan
                jp c,abortErr           ;Nokken indien ^STOP werd ingedrukt
                jr flushFile            ;Volgende datablok naar buffer

addFormFeed:    bit swFormfeed,(iy+parseFl); Formfeed toevoegen?
                ret z                   ; Nee, => Doe niks meer
                ld hl,dataBuf
                ld (hl),ff              ;Plaats formfeed in databuffer
                ld de,1                 ;1 karakter
                ld a,pbFlush            ;Verstuur FormFeed naar buffer
                call callPB             ;Roep PB aan
                jp c,abortErr           ;Nokken indien ^STOP werd ingedrukt
                ret                     ;Klaar

; readFile
;
; Lees data uit de huidige file
;
; In:  HL=Aantal bytes
;      DE=bestemming
;
; Uit: HL=Aantal gelezen bytes
;
readFile:       ld b,(iy+fileHandle)    ;B=DOS2 file handle
                ld c,read               ;DOS2 read funktie
                bit dos2,(iy+sysFl)     ;DOS2?
```

```
                jp nz,dos               ; Ja, => Lees uit filehandle
                push hl                 ; Nee, lees uit FCB
                ld c,setDta
                call dos                ;Set Disk Transfer Address
                pop hl
                ld de,fcb               ;DOS1 file descriptor
                ld c,rdBlk
                jp dos                  ;Random block read


; closeFile
;
; Sluit de huidige file
;
closeFile:      ld b,(iy+fileHandle)    ;B=DOS2 file handle
                ld c,close              ;DOS2 close funktie
                bit dos2,(iy+sysFl)     ;DOS2?
                jp nz,dos               ; Ja, => Sluit filehandle
                ld de,fcb               ;DOS1 FCB address
                ld c,fClose             ;Sluit file
                jp dos


; PrintCurFile
;
; Toon Naam van huidig bestand in (FCB)
;
printCurFile:   ld de,tCurFile          ;Intro
                call printLine

                bit dos2,(iy+sysFl)     ;DOS1?
                jr z,printFCB           ; Ja, => FIB omvormen niet nodig

                ld de,fib+1             ;ASCIIZ filename in FIB
                ld hl,fcb+1             ;Omvormen naar DOS1 formaat
                ld c,pFile              ;Parse FileName
                call dos                ;Doe dat

printFCB:       ld hl,fcb+1             ;Start van Var-Name
                ld b,8                  ;8 kar's in naam
printNameLp:    ld a,(hl)
                call printChar          ;Print Character
                inc hl
                djnz printNameLp

                ld b,3                  ;Extensie is 3 karakters
                ld a,' '
                cp (hl)                 ;Is de extensie leeg?
                jr z,noDot              ; Ja, => Geen punt printen
                ld a,'.'
noDot:          call printChar
printExtLp:     ld a,(hl)               ;Print de extensie
                call printChar
                inc hl
                djnz printExtLp
                ld de,tNewLine          ;CR/LF afdrukken
                jp printLine


; deci
;
; Plaats HL als decimaal getal in de tFree buffer
;
deci:           ld   de,tFree
```

```
                ld   b,5
                xor  a
dc1:            ld   (de),a             ;Vul de buffer met 0-karakters
                inc  de
                djnz dc1
dc2:            xor  a                  ;No carry + a=0
                ld   e,a
                ld   b,16
dc3:            rl   l
                rl   h
                rl   e
                ld   a,e
                sub  10
                ccf
                jr   nc,dc4
                ld   e,a
dc4:            djnz dc3
                rl   l
                rl   h
                ld   a,e
                add  a,'0'
                push hl
                ld   hl,tFree+3
                ld   de,tFree+4
                ld   bc,4
                lddr
                ld   (tFree),a          ;Vul teken vooraan in de buffer in
                pop  hl
                ld   a,h
                or   l
                jr   nz,dc2
                ret


; decToBin
;
; Decimaal ASCII in (DE) naar binair in HL
;
; In:  DE=Pointer naar ASCII cijfers
; Uit: HL=Binair getal
;
decToBin:       ld hl,0
dbLp:           ld a,(de)               ;Haal cijfer
                sub '0'                 ;ASCII offset eraf
                ret c                   ;Klaar indien geen cijfer
                cp '9'+1
                ret nc
                ld b,h
                ld c,l
                add hl,hl               ;HL=HL*10
                add hl,hl
                add hl,bc
                add hl,hl
                ld c,a
                ld b,0
                add hl,bc               ;Cijfer erbij
                inc de                  ;Volgende cijfer
                jr dbLp


; PrintChar
;
; Print een karakter op de console
;
```

```
; In: A=Character
; Wijzigt: AF, DE
;
printChar:      push hl
                push bc
                ld e,a
                ld c,conOut
                call dos
                pop bc
                pop hl
                ret


; printLine
;
; Druk tekstregel (DE) af, $ terminated
;
printLine:      ld c,strOut


; dos
;
; Entry voor MSX-DOS aanroep
;
dos:            call bDos               ;Roep bDOS aan
                ld iy,bArea             ;Herstel boolean pointer
                ret


; callPB
;
; Entry voor Printer Buffer TSR aanroep
;
; In: AF, HL, DE = Data registers voor de TSR
; Uit: AF, HL, DE, BC = Data registers van de TSR
;
callPB:         ld bc,(pbID)            ;Printer Buffer ID code in BC
                ld ix,(callTsrAddr)     ;Haal adres van de "callTsr" routine
                call jpIX               ;Spring naar MemMan / PB
                ld iy,bArea             ;Herstel boolean pointer
                ret
jpIX:           jp (ix)                 ;Spring naar "CallTsr"


; callMemMan
;
; Entry voor MemMan / Extended BIOS aanroep
;
callMemMan:     call extbio             ;Roep extend bios functie aan
                ld iy,bArea             ;Herstel boolean pointer
                ret


; Foutmeldingen
;
noMemMan:       ld de,eNoMemMan         ;"Geen memMan versie 2.1"
                jr printErr
noPb:           ld de,eNoPb             ;"Geen PB 1.1"
                jr printErr
notFoundErr:    ld de,eNotFound         ;"File not found"
                jr printErr
openErr:        ld de,eOpen             ;"File not found"
                jr printErr
abortErr:       ld de,eAbort            ;"CTRL STOP pressed"
```

```
printErr:       call printLine          ;Druk tekst af
                jr backToDos            ;Terug naar MSX-DOS

; noFileName
;
; Toont hulpregel indien er geen filename of switch werd opgegeven.
;
noFileName:     ld a,(iy+parseFl)       ;Haal flag-byte met switches
                or a                    ;Meer dan nul switches opgegeven?
                jr nz,backToDos         ; Ja, => Geen hulptekst nodig
                ld de,tHelp             ; Nee, toon hulp-tekst

; printAndQuit
;
printAndQuit: call printLine            ;Print tekst
backToDos:      ld bc,term              ;Terug naar DOS2, no error code
                bit dos2,(iy+sysFl)
                jr nz,dos
                rst 0                   ;Terug naar DOS1

; Variabelen
;
pbID:           dw 0                    ;ID code van PB TSR
bufSize:        dw 0                    ;Omvang van databuffer
installPnt:     dw 0                    ;Te installeren hoeveelheid kB buffer
callTsrAddr:    dw 0                    ;Adres van de "CallTsr" routine

; Boolean variabelen
;
bArea:          db 0                    ;parseFl      Commandline switches
                db 0                    ;fileFl       File status flags
                db 0                    ;sysFL          System flags
                db 0                    ;FileHandle   DOS2 filehandle


; Naam van de TSR
;
pbNaam:         db  MJV Printbuf

; Error teksten
;
eOpen:          db 'Error opening file',cr,lf,eos
eNotFound:      db 'File not found',cr,lf,eos
eAbort:         db cr,lf
                db 'CTRL-STOP pressed: Buffer cleared',cr,lf,eos
eNoMemMan:      db 'MemMan version 2.1 not installed',cr,lf,eos
eNoPb:          db 'MJV Printbuf version 1.1 not installed',cr,lf,eos


; Overige teksten
;
tIntro:         db cr,lf
                db 'PRINT version 1.1 - by MJV',cr,lf
                db '(c) MSX Computer Magazine 1991',cr,lf,lf,eos

tHelp:          db 'Usage:'
                db tab,'PRINT filespec  /C /D /F /H /In /S /V',cr,lf
                db lf
                db tab,'filespec =  path- or filename of file(s) to be printed'
                db cr,lf
                db tab,'      /C  =  Clear the printer buffer',cr,lf
                db tab,'      /D  =  Suppress prompt when clearing buffer',cr,lf
                db tab,'      /F  =  Add formfeed at end of file',cr,lf
                db tab,'      /H  =  Also print hidden files (MSX-DOS2 only)',cr,lf
```

```
                db tab,'     /In =  Install n kB printer buffer',cr,lf
                db tab,'     /S  =  Show current size of printer buffer',cr,lf
                db tab,'     /V  =  Show version of PB.TSR',cr,lf
                db lf
                db tab,'All items are optional',cr,lf
                db eos

tClear:         db 'Printer buffer has been cleared',cr,lf,eos
tFree:          db '128   kB printer buffer installed',cr,lf,eos
tDestroy:       db 'Destroy all data in printer buffer? (Y/N) ',eos
tVersion1:      db 'MJV Printbuf-TSR version: ',eos
tVersion2:
verHigh:        db '1.'
verLow:         db '0'
tNewLine:       db cr,lf,eos
tYesNo:         db 'Y',cr,lf,eos
tCurFile:       db 'Loading file: ',eos

fib             equ $                   ;DOS2 64 byte buffer file info block
fcb             equ $                   ;DOS1 24 byte buffer File Control Block

                end
```

Trefwoorden index

Index MemMan funkties