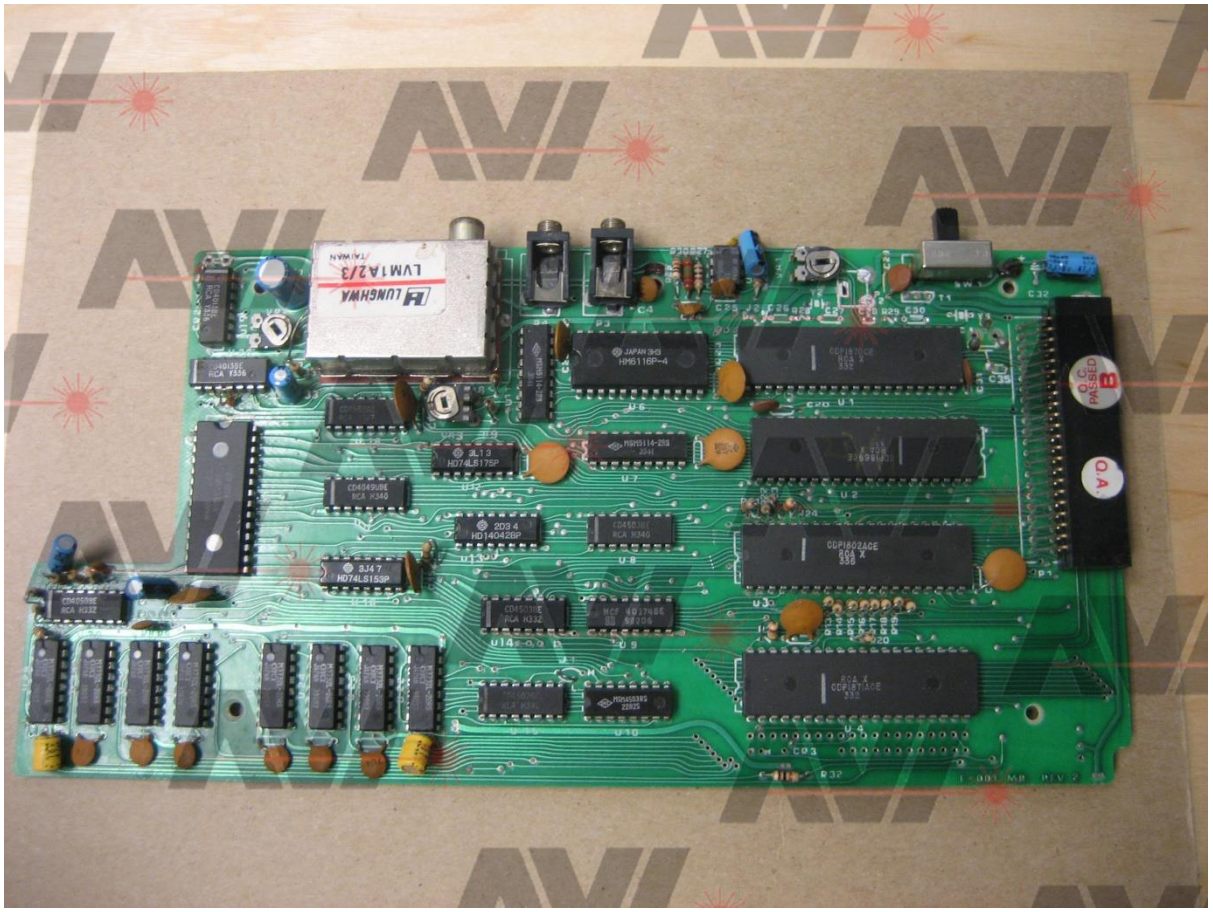

COMX 35
COMX 35



COMX CFORTH Assembler

De CFORTH Assembler, geschreven voor de COMX-35 homecomputer met 1802 processor, is niet zomaar een assembler. Zoals de naam al doet vermoeden is de assembler niet alleen in Forth geschreven maar is het ook Forth georiënteerd. Hiermee wordt bedoeld dat de assembler net zo werkt als Forth. Doordat de assembler in Forth geschreven is kan het alle eigenschappen bezitten van een normale assembler. De Cforth Assembler heeft echter meer in zijn mars:

1. **Het is uitbreidbaar!**
2. **Het bezit lus- en sprongstructuren!!!**

Wat dit allemaal mag inhouden wordt in de volgende paragrafen uitgelegd. Het zal duidelijk zijn dat enig begrip van Forth noodzakelijk is om met de Assembler te kunnen werken. Diegenen die in assembler willen leren programmeren maar niet met Forth bekend zijn moeten hierdoor niet ontmoedigd raken. Want als u enige moeite neemt om Forth te leren dan zult u merken dat deze Assembler gemakkelijk te temmen is. Het zal u opvallen dat door het gebruik van lus- en sprongstructuren het in assembler programmeren een stuk eenvoudiger wordt. Het voordeel dat er een Forth kern onder de Assembler aanwezig is zal tevens het grootste nadeel blijken te zijn.

Structuur, syntax

Voordat begonnen wordt om uit te leggen wat Cforth Assembler meer is dan een normale assembler is het nodig even naar een dergelijke assembler terug te kijken. Deze assembler bezit de volgende eigenschappen:

1) Het kan elke instructie van de processor verwerken.
2) Het kent de mogelijkheid om labels te definiëren, dwz. dat men plaatsen waar naar toe gesprongen kan worden of plaatsen waarnaar gerefereerd kan worden met een symbolische naam kan aangeven.
3) Er kunnen zelfs namen toegekend worden aan constanten met de zogenaamde „EQU", de equate opdracht, en de „EQUAL" opdracht.
4) Er zijn zogeheten pseudomnemonics, welke niet een directe overeenkomst hebben met een processorinstructie, maar bijvoorbeeld een handige groep instructies onder een naam samenvat, of die aan de assembler bepaalde gegevens vertellen (EQU, ORG, END)

De Cforth Assembler bezit ook deze eigenschappen.

De Cforth Assembler heeft door de op Forth georiënteerde bouw een aantal eigenaardigheden ten opzichte van een normale assembler.

a) de mnemonics eindigen allemaal op een komma. Dit is om eenduidigheid te verkrijgen tussen de mnemonics in de assembler woordenlijst en andere woordenlijsten. Verder geeft de komma aan dat deze woorden waarden in het geheugen achterlaten (assembleren), in analogie met andere woorden in Forth die op een komma eindigen.
b) de assembler gebruikt de postfix notatie, dwz. dat de argumenten de mnemonic vooraf gaan.
   bijvoorbeeld:
   in plaats van de „normale" notatie: SPRING naar ADRES (label)
   wordt dit nu: ADRES (label) naar SPRONG,

Opmerking:
Mnemonic komt van het griekse woord mnèmè dat herinnering betekent. Denk maar eens aan woorden als:
memo(randum) memoreren memorabel

voorbeeld:
definitie van Forth + mbv de 1802-Assembler twee getallen (#A en #B) worden bij elkaar opgeteld de getallen en het resultaat staan op de stack

```
HEX    : #A   9 ;      ( pointer to first # )
       : #B   8 ;      ( pointer to second #)

CODE   +
       #A  GLO,        ( copy stackpointer #A --> #B )
       #B  PLO,
       #A  GHI,
       #B  PHI,

       #B  DEC,        ( #B points to LSB second #    )
       #B  SEX,
       #A  INC,        ( #A points to LSB first  #    )

       #A  LD,
           ADD,        ( add LSB :  L.A+L.B           )
           STXD,       ( LSB L.A+L.B --> #B , #B:=#B-1)
       #A  DEC,        ( #A:=#A-1    point to MSB     )
       #A  LD,
           ADC,        ( add MSB :  M.A+M.B+CARRY     )
       #B  STR,        ( MSB M.A+M.B+CARRY --> #B     )
       #A  DEC,        ( put <stack>pointer on MSB result A+B )
       #A  DEC,
           RTF,        ( Return To Forth              )

ENDCODE
```

COMX CFORTH Assembler

LSB Laagst Significante Byte van een 16-bits getal (0-255)
MSB Meest Significante Byte van een 16-bits getal (256-tallen)

Definitie van de mnemonic's

Men zal zich wel afvragen hoe definieert men de mnemonic als een Forth woord?

Methode 1
Recht-
streeks mbv een „:" (co-
lon) definitie

```
HEX    : IDL,  Ø C, ;
       : ADD, F4 C, ;
       : BR,   3Ø C, C,   ;
       : LBR,  CØC, ,     ;
```

Dit is de makkelijkste manier om de assembler te definiëren. Het is echter niet de slimste en kortste manier.

Methode 2
a. bekijk de OPCODE'S van de processor (OPeration CODE)
b. hoeveel programma bytes zijn voor de opcode's nodig?
c. zijn er OPCODE'S waarvan in de CODE byte enkele bits later gezet worden?

Gebruik <BUILDS...DOES> om woorden te definiëren die de mnemonic's aanmaken (zie SCR # 101)

Voorbeeld:

```
:  BCODE <BUILDS     C,
         DOES>      CC    C, ;
```

mbv. BCODE kunnen nu OPCODE'S gedefinieerd worden die 1 byte lang zijn dus:

```
HEX     ØØ      BCODE  IDL,
        F4      BCODE  ADD,
```

Pseudo mnemonics

Wat zijn pseudomnemonics eigenlijk?
Pseudomnemonics zijn uitbreidingen aan de woordenschat van de assembler, met als taak:
1. de assembler iets mee te delen (woorden als ORG en END in een normale assembler, en woorden als EQU en ENDCODE in een Forth Assembler).
2. het gebruik van de assembler voor de gebruiker te vereenvoudigen, bijvoorbeeld door een aantal handige instructies onder een duidelijke naam te brengen.

Enkele voor de hand liggende pseudomnemonics van het tweede type voor de 1802-processor zijn:
CALL, en RTN, (= RTC,)
Deze twee „standaard" pseudo's dienen voor het nesten van de subroutines in machinetaal.

Minder voor de hand liggende pseudo's zijn:
RØ NEWR, LDR, PSHR, (= POPR,)
SØ, OLDR, ABSORB, PULR, PULS,

```
                LBL naam1          1 EQU een
                LBL naam2          2 EQU twee

                LABEL subroutine   ...mnemonic,s... RTC,


CODE voorbeeld    ...mnemonic,s... RTF,  ENDCODE
                     ↑   ↑   ↑   ↑   ↑   ↑
          hier nooit LBL , EQU , LABEL of EQUAL !!! declaratie
```

Deze pseudo's dienen als overbrugging tussen Cforth, de Assembler en enkele ROM-routines. Met deze pseudo's wordt het assembler programmeren een stuk eenvoudiger.
ABSORB, is hierbij een pseudo die voor de meeste mensen nieuw zal zijn. Het is in wezen niets anders dan de Assembler tegenpool van Forth's." (dot-quote):

- ." stuurt de ascii-codes van een tekst tussen aanhalingstekens naar het beeldscherm.
- ABSORB, zet de ascii-codes van een tekst tussen aanhalingstekens in het vrije RAM-geheugen achter het Forth.

ABSORB, is een hulpmiddel bij de ROM-routine MESOUT en vereenvoudigt de wijze waarop de tekst in het geheugen gezet kan worden.

voorbeeld:

```
      ." Naar beeldscherm"
ABSORB, "absorberen maar"
```

In het algemeen zullen dergelijke pseudomnemonics bestaan uit de organisatie van een aantal mnemonics. Eén van de grote verschillen tussen een Forth Assembler en een gewone assembler is de mogelijkheid om pseudomnemonics als gebruiker zelf te definiëren.

voorbeeld:

```
: PULS,        (REG PULS,   perform a PUL via SØ (=R9) )
    DUP        (copy designation register              )
    9 LDA,  PHI,    (PUL MSB register                  )
    9 LDN,  PLO,    (PUL LSB register                  )
    9 DEC,  9 DEC,   9 DEC,  ;
```

Labels en constanten

De labels (LBL en LABEL) en de constanten (EQU en EQUAL) zijn in de Assembler als volgt gedefinieerd:

```
: EQU     VARIABLE ; : EQUAL   <BUILDS , DOES> @ ;
: LBL   Ø VARIABBLE ;: LABEL   <BUILDS.WHERE.2+ , DOES> @ ;
```

EQU een variabele
LBR een variabele die voor de assembler code wordt gedefinieerd. De waarde van de variabele wordt pas later vastgelegd.
EQUAL een constante
LABEL een soort constante: wijst de plaats aan van het RAM-adres precies achter de LABEL declaratie. (WHERE is een adrespointer in de Assembler)

De waarden van EQU, LABEL en EQUAL zijn bij creatie bepaald, de waarde van LBL nog niet. Zowel LBL en LABEL alswel EQU en EQUAL moeten voor gebruik gedefinieerd worden en mogen nooit gebruikt worden tussen een Assembler CODE en ENDCODE definitie.

voorbeeld: zie hierboven

Het woord om de waarde van een LBL vast te leggen is: SET Het woord om de waarde van een LBL of EQU op stack te zetten, voor gebruik door een mnemonic, is: TO (= @)
De waarden van LABEL en EQUAL worden bij elke aanroep na hun creatie automatisch op stack gezet.

Lus- en sprongstructuren

Door gebruik te maken van labels en constanten is het in de Assembler alleen mogelijk om sprongen terug (backward branching) te maken. Soms echter is het gewenst om een (grote) sprong voorwaarts (forward branching) te maken. Het grote probleem bij een voorwaartse sprong is dat het adres verderop nog niet bekend is. Men kan het adres natuurlijk even uitrekenen, maar wat als er ergens tussen sprong en adres nog wat extra code tussengepropt moet worden of ergens in de code nog een fout zit die verbeterd moet worden? Terug naar af en alle voorwaartse sprongen opnieuw uitrekenen? Een nogal omslachtige manier van een programma schrijven niet?

Door gebruik te maken van lus- en sprongstructuren kunnen veel adressen, zowel voor als achterwaarts, in Forth berekend worden zonder dat de Assembler pro-

grammeur er iets van merkt. Aan de Assembler zijn de volgende lus- en sprongstructuren (pseudomnemonic,s) toegevoegd:

```
BEGIN,  .....mnemonic,s.....  AGAIN,

BEGIN,  .....mnemonic,s.....  UNTIL,      (D=Ø)
                             Ø<UNTIL,     (D<Ø)
                              <UNTIL,     (D<?)
                            Ø>=UNTIL,     (D>=Ø)
                             >=UNTIL,     (D>=?)

BEGIN,  .....mnemonic,s.....  WHILE,      (D=Ø)
        .....mnemonic,s.....  REPEAT,     (BEGIN,)

IF, ..(D=Ø)..mnemonic,s.....  ELSE,
    ..(D≠Ø)..mnemonic,s.....  THEN,

Ø<IF, ..(D<Ø)..        Ø>=IF, ..(D>=Ø)..
 <IF, ..(D<?)..         >=IF, ..(D>=?)..

CASE,   ..(D=X1)..  OF,  .....mnemonic,s.....  ENDOF,
        .
        .
        OTHERWISE,  .....mnemonic,s.....  ENDCASE,
```

De lus- en sprongstructuren in de Assembler werken nagenoeg hetzelfde als de lus- en sprongstructuren in Forth. In plaats van een vlag wordt nu het dataregister D getest.

voorbeeld
definitie van HDUMP mbv de 1802-assembler
aanroep : addr N HDUMP
output : hexadecimale geheugendump vanaf addr
gebruik : EQU ; EQUAL ; LBL ; LABEL PSEUDO-MNEMONIC,s

```
HERE ORG,   HEX
     2E42 EQU   CRLF
     320F EQUAL HEXOUT (=EOM1 voor Comx-35)
     322B EQUAL MESOUT
          LBL   SPC
          LBL   LINEADDR

LABEL LINEOUT          ( output Ascii-characters in line )
  MESOUT    CALL,   ABSORB, "  "
  LINEADDR  SET
            ABSORB, "........"
            NUL,
  LINEADDR  TO         ( reset lineaddr to first char of line )
          9 LDR,
  CRLF   TO CALL,      ( output memory addr              )
          7 GHI,  HEXOUT CALL,
          7 GLO,  HEXOUT CALL,
  SPC       SET        ( output a blank                  )
  MESOUT    CALL,
            ABSORB, " "
            NUL,
            RTC,
```

```
LABEL DMP              ( output content memory pointed by R7 )
  SPC    TO CALL,
          7 LD,   HEXOUT CALL,
            RTC,

LABEL CHR           ( store Ascii-char or dot in outputline )
          7 LDA,
            CASE,
                 20  <OF,  2E LDI,  9 STR,  ENDOF,
                 60 >=OF,  2E LDI,  9 STR,  ENDOF,
                 OTHERWISE,    9 STR,
            ENDCASE,
          9 INC,
            RTC,



CODE HDUMP          ( ADDR N  --->   )
          8 PULS,   ( R8 counter     )
          7 PULS,   ( R7 addr pointer)
            NEWR,
          9 PSHR,
LINEADDR TO 9 LDR,
          8 DEC,
                BEGIN,
                    7 GLO,    ( new line,line feed ? )
                   0F ANI,    ( get low nibble R7.0  )
                      CASE,
                         0 OF,  LINEOUT CALL,  ENDOF,
                         8 OF,  LINEOUT CALL,  ENDOF,
                      ENDCASE,
                  DMP CALL,
                  CHR CALL,
                    8 DEC,
                    8 GHI,
                   FF XRI,
                UNTIL,          LINEOUT CALL,
          9 PULR,
            OLDR,
            RTF,
ENDCODE
```

De voorwaartse sprong

Op het eerste gezicht lijkt het onmogelijk te springen naar een adreslabel die pas verderop in de code op een waarde gezet wordt. Met de volgende woorden is het mogelijk om vooruit te springen naar een nog onbekende label (LBL).

De woorden zijn:
?TOBR ?TOLBR +SET

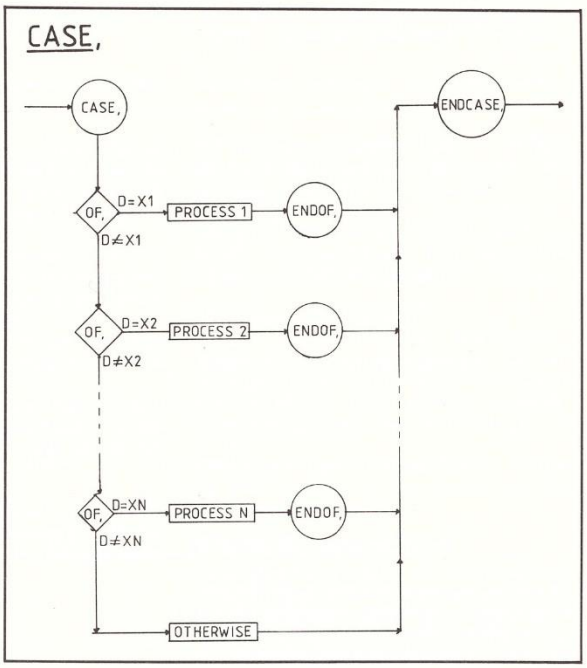
Het gebruik van deze woorden wordt echter afgeraden daar een programma er niet overzichtelijker op wordt. Dit zal hieronder met een voorbeeld aangetoond worden.

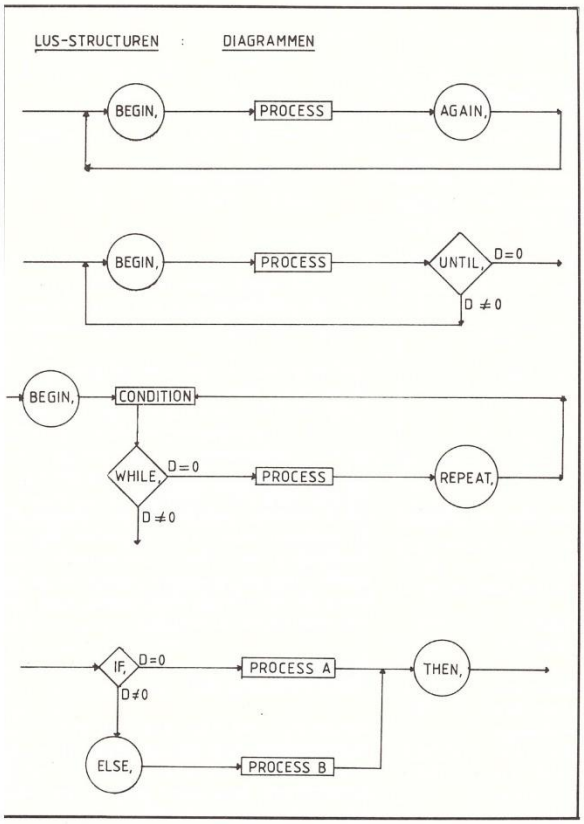
COMX CFORTH Assembler

voorbeeld:

```
        LBL SLIERT1       LBL SLIERT2
        LBL SLIERT3       LBL SLIERT4

        CODE SPAGHETTI
             ..mnemonic,s..
        SLIERT1 ?TOLBR LBR,
             ..mnemonic,s..
        SLIERT3 ?TOBR  BNZ,
             ..mnemonic,s..
        SLIERT1 ?TOBR  BNF,
             ..mnemonic,s..
        SLIERT4 ?TOLBR LBNQ,
             ..mnemonic,s..
             ..mnemonic,s..
        SLIERT1 SET
             ..mnemonic,s..
        SLIERT2 SET
             ..mnemonic,s..
        SLIERT3 SET
             ..mnemonic,s..
        SLIERT4 SET
        SLIERT4 1 +SET
        SLIERT1 1 +SET
        SLIERT3 1 +SET
        SLIERT1 1 +SET          ..mnemonic,s.. etc.     ENDCODE
```





Epiloog

Hoewel een Forth Assembler vele positieve eigenschappen bezit zullen er ongetwijfeld ook negatieve aanwezig zijn, waarvan de meeste juist veroorzaakt zullen worden doordat de assembler Forth gericht is. In dit artikel zijn meer de positieve eigenschappen belicht. Een negatief punt bijvoorbeeld is dat assembler code programma's over het algemeen vrij lang zullen zijn. Als deze programma's aangemaakt worden in de verschillende Forth schermen (screens) dan zult u in het begin merken dat het niet prettig leest, daar het niet echt overzichtelijk lijkt (maar alles went).

Tot slot een nog onbesproken item: ORG,
Met ORG, is het mogelijk om de geassembleerde code geschikt te maken voor een andere plaats dan in het vrije RAM-geheugen direct achter het Forth woordenboek. U geeft dan voorafgaand aan de assembler code: addr ORG, Anders geeft u : HERE ORG,

Ik hoop dat ik met dit artikel over een Forth Assembler de drempelvrees van (beginnende) microfanaten voor zowel het Forth alswel het Assembler programmeren verminderd heb. ■

Yuen Keong Ng

standaard printerinterface

```
  OK
7 PTRMODE

START:71F8
END  :7350


71F8  43 52 4C C6 67 E8 49 E0  CRL...I.
7200  2E 42 84 45 4F 4D B1 71  .B.EOM..
7208  F7 52 82 6E 8F 32 0F 86  .R...2..
7210  4D 45 53 4F 55 D4 72 02  MESOU...
7218  52 82 6E 8F 32 2B 83 53  R...2+.S
7220  50 C3 72 0F 49 E0 72 66  P...I...
7228  88 4C 49 4E 45 41 44 44  .LINEADD
7230  D2 72 1E 49 E0 72 4C 87  ...I..L.
7238  4C 49 4E 45 4F 55 D4 72  LINEOU..
7240  28 52 82 6E A7 72 47 D4  (R....G.
7248  32 2B 20 20 2E 2E 2E 2E  2+  ....
7250  2E 2E 2E 2E 00 F8 4C A9  ......L.
7258  F8 72 B9 D4 2E 42 97 D4  .....B..
7260  32 0F 87 D4 32 0F D4 32  2...2..2
7268  2B 20 00 D5 83 44 4D D0  + ...DM.
7270  72 37 52 82 6E A7 72 78  .7R.....
7278  D4 72 66 07 D4 32 0F D5  ..:..2..
7280  83 43 48 D2 72 6C 52 82  .CH...R.
7288  6E A7 72 8C 47 BE 9E FF  ....G...
7290  20 C3 72 9A F8 2E 59 C0   .....Y.
7298  72 A8 9E FF 60 CB 72 A6  ........
72A0  F8 2E 59 C0 72 A8 9E 59  ..Y....Y
72A8  19 D5 85 48 44 55 4D D0  ...HDUM.

72B0  72 80 72 B4 49 B8 09 A8  ....I...
72B8  29 29 29 49 B7 09 A7 29  )))I...)
72C0  29 29 E2 9C 73 8C 73 F8  )).....
72C8  02 AD F8 33 BD 99 73 89  ...3....
72D0  73 F8 4C A9 F8 72 B9 28  ..L....(
72D8  87 FA 0F BE 9E FF 00 CA  ........
72E0  72 E8 D4 72 47 C0 72 F4  ....G...
72E8  9E FF 08 CA 72 F4 D4 72  ........
72F0  47 C0 72 F4 D4 72 78 D4  G.......
72F8  72 8C 28 98 FB FF 3A D8  ..(...:.
7300  D4 72 47 60 72 A9 F0 B9  ..G.....
7308  F8 1C AD F8 9C BD 60 72  ........
7310  AC F0 BC E9 DC 87 50 54  ......PT
7318  52 4D 4F 44 C5 72 AA 73  RMOD....
7320  21 E2 9C 73 8C 73 F8 02  !.......
7328  AD F8 33 BD 49 B8 09 A8  ..3.I...
7330  29 29 29 D4 C0 40 F8 1C  )))..@..
7338  AD F8 9C BD 60 72 AC F0  ........
7340  BC E9 DC 07 50 54 52 4D  ....PTRM
7348  4F 44 45 20 20 20 20 20  ODE
7350  20 20 20 20 20 20 20 20
  OK
0 PTRMODE
```

geheugendunp m.b.v HDUMP

```
  OK
HEX     71F8 160 HDUMP       DECIMAL
  ........
71F8  43 52 4C C6 67 E8 49 E0  CRL...I.
7200  2E 42 84 45 4F 4D B1 71  .B.EOM..
7208  F7 52 82 6E 8F 32 0F 86  .R...2..
7210  4D 45 53 4F 55 D4 72 02  MESOU...
7218  52 82 6E 8F 32 2B 83 53  R...2+.S
7220  50 C3 72 0F 49 E0 72 66  P...I...
7228  88 4C 49 4E 45 41 44 44  .LINEADD
7230  D2 72 1E 49 E0 72 4C 87  ...I..L.
7238  4C 49 4E 45 4F 55 D4 72  LINEOU..
7240  28 52 82 6E A7 72 47 D4  (R....G.
7248  32 2B 20 20 32 2B 20 20  2+  2+
7250  32 2B 20 20 00 F8 4C A9  2+  ..L.
7258  F8 72 B9 D4 2E 42 97 D4  .....B..
7260  32 0F 87 D4 32 0F D4 32  2...2..2
7268  2B 20 00 D5 83 44 4D D0  + ...DM.
7270  72 37 52 82 6E A7 72 78  .7R.....
7278  D4 72 66 07 D4 32 0F D5  .....2..
7280  83 43 48 D2 72 6C 52 82  .CH...R.
7288  6E A7 72 8C 47 BE 9E FF  ....G...
7290  20 C3 72 9A F8 2E 59 C0   .....Y.
7298  72 A8 9E FF 60 CB 72 A6  ........
72A0  F8 2E 59 C0 72 A8 9E 59  ..Y....Y
72A8  19 D5 85 48 44 55 4D D0  ...HDUM.
72B0  72 80 72 B4 49 B8 09 A8  ....I...
72B8  29 29 29 49 B7 09 A7 29  )))I...)
72C0  29 29 E2 9C 73 8C 73 F8  )).....
72C8  02 AD F8 33 BD 99 73 89  ...3....
72D0  73 F8 4C A9 F8 72 B9 28  ..L....(
72D8  87 FA 0F BE 9E FF 00 CA  ........
72E0  72 E8 D4 72 47 C0 72 F4  ....G...
72E8  9E FF 08 CA 72 F4 D4 72  ........
72F0  47 C0 72 F4 D4 72 78 D4  G.......
72F8  72 8C 28 98 FB FF 3A D8  ..(...:.
7300  D4 72 47 60 72 A9 F0 B9  ..G.....
7308  F8 1C AD F8 9C BD 60 72  ........
7310  AC F0 BC E9 DC 87 50 54  ......PT
7318  52 4D 4F 44 C5 72 AA 73  RMOD....
7320  21 E2 9C 73 8C 73 F8 02  !.......
7328  AD F8 33 BD 49 B8 09 A8  ..3.I...
7330  29 29 29 D4 C0 40 F8 1C  )))..@..
7338  AD F8 9C BD 60 72 AC F0  ........
7340  BC E9 DC 05 48 44 55 4D  ....HDUM
7348  50 20 20 20 20 20 20 20  P
7350  20 20 20 20 20 20 20 20
7358   OK
0 PTRMODE
```

HCC Forth gebruikersgroep