KOPI

Vejledning

i

assemblerprogrammering

under Comal

Udarbejdet af:

John Plate
Laborantskolen i Roskilde

Januar 1983

INDHOLD

1. Indledning

Dette papir er en konkret vejledning i implementering af rutiner skrevet i symbolsk maskinsprog, afviklet under kontrol af et Comal-program.

Formålet med sådanne rutiner er at kunne indføre tidskritiske rutiner eller at udnytte maskinnære funktioner i værtsdatamaten som ikke er tilgængelige fra det ellers anvendte højere programmeringssprog.

Der kan f.eks. være tale om sammenkobling af en datamat med laboratorieudstyr, som datamaten skal styre eller opsamle data fra. Der kan også være tale om egentlig processtyring.

Vejledningen er rettet mod mikrodatamaten Comet (MPS 3000) og anvendelse af Comal-80 som værtssprog. Det symbolske maskinsprog er Z80.

Vejledningen er ikke tænkt som begyndervejledning eller lærebog. Udbytterig læsning forudsætter blandt andet:

1) kendskab til assemblerprogrammering, Z80

2) kendskab til Comal-80

3) kendskab til principperne for den maskinelle virkemåde i en datamat, herunder principper for ind-uddata kommunikation med ydre enheder

Modtagere af vejledningen bedes udfylde en KOPI af det på næste side aftrykte skema og sende det til den angivne adresse. Herved sikrer man sig modtagelse af rettelsesblade og nye udgaver. Forfatteren modtager med tak læseres kommentarer, kritik og ændringsforslag.

John Plate
Laborantskolen i Roskilde

Dato: _____

Til
Laborantskolen i Roskilde
Att: John Plate
Maglegårdsvej 8
4000 Roskilde

Opdateringer og nye udgaver af "Vejledning i assembler-programmering" (januar 1983) bedes sendt til:

Navn: _____

Skole: _____

Adresse: _____

By: _____

Dato:_____

Til
Laborantskolen i Roskilde
Att: John Plate
Maglegårdsvej 8
4000 Roskilde

Kommentarer til: "Vejledning i assemblerprogrammering"
(januar 1983):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Fra:

2. Comal faciliteter

Vejledningen omfatter præsentation af metoder til implementering af (eksterne) assemblerrutiner i Comal-80 på Comet (MPS 3000) mikrodatamaten.

Comal-80 stiller få og primitive faciliteter til rådighed til formålet. Disse er:

VARPTR(<variabel>)
- er en funktion der returnerer adressen i det interne lager på <variabel>. <variabel> må være simpel, struktureret eller indiceret.
Eksempel:
<ln> entry:=VARPTR(asm$(3))
<ln> entry:=VARPTR(x)

(<ln> er linienummer)

CALL <adresse>
- der gemmer den aktuelle programtilstand, lægger returadressen på stakken og hopper til adressen <adresse>. Alle registre kan således anvendes frit i assemblerrutinen og efterlades med udefinerede værdier.

CALL-instruktionen forudsætter at den kaldte rutine afsluttes med udførelse af instruktionen RET.
Eksempel:
<ln> CALL entry

PEEK (<adresse>)
- er en funktion der returnerer værdien af lagercellen (1 oktet) med adressen <adresse>.
Eksempel:
<ln> x:=PEEK(entry+5)

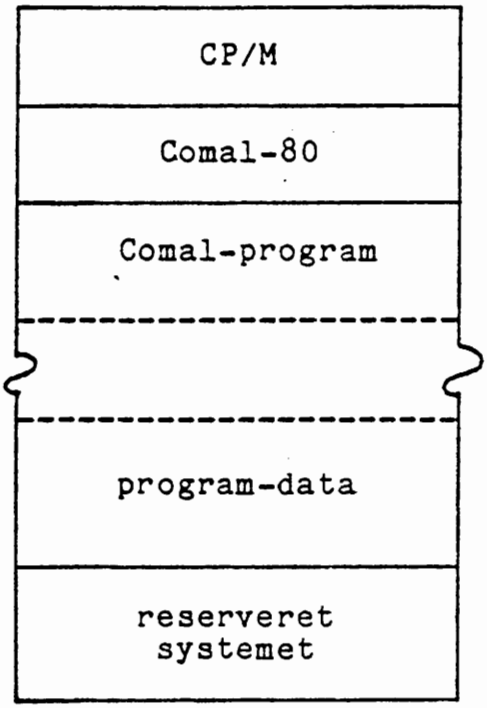
POOK <adresse>,<udtryk>
- der skriver værdien af <udtryk> på adressen <adresse> i det interne lager. Værdien af <udtryk> skal ligge i intervallet 0..255.
Eksempel:
<ln> POOK entry+9,n+p

INP (<portnummer>)
- er en funktion der returnerer indholdet (1 oktet) af port nummer <portnummer>.
Eksempel:
<ln> port8:=INP(8)

OUT <portnr>,<udtryk>
- der skriver værdien af <udtryk> på port nummer <portnr>. Værdien af <udtryk> skal ligge i intervallet 0..255.
Eksempel:
<ln> OUT 8,n

3. Placering af rutinen i Comalprogrammet

I princippet er det interne lager disponeret således:



Når en assemblerrutine skal udføres må den nødvendigvis befinde sig i det interne lager. Der kan f.eks. skabes et "hul" mellem de iøvrigt udnyttede dele af lageret. CP/M kan ændres til at disponere over et mindre adresserum, og det tiloversblevne kan så rumme assemblerrutinen.

Metoden opererer altså med en fast plads i lageret som permanent er reserveret, hvad enten den er i brug eller ej. Fordelen er, at det benyttede adresserum er kendt på forhånd. Imidlertid synes denne løsning at savne den fornødne generalitet og det kan let blive forvirrende med forskellige CP/M versioner kørende i flæng. Den vil derfor ikke blive behandlet yderligere i denne vejledning.

Det er også set, at assemblerrutiner lagres i datasætninger som konstanter. Fordelen er at assemblerrutine og program kan lagres sammen. Ulempen er at det kræver en del manuelt arbejde, når ændringer skal indføjes. Metoden er derfor kun aktuel for meget små ru-

tiner og vil derfor heller ikke blive yderligere omtalt her.

Eneste mulighed er herefter at placere assemblerrutinen i Comals dataområde. Den nødvendige plads må reserveres via Comal, f.eks. i en tabel eller en tekststreng:

```
DIM rutine$ OF 200
```

Størrelsen af datastrukturen skal være større end eller lig med det antal oktetter assemblerrutinen fylder.

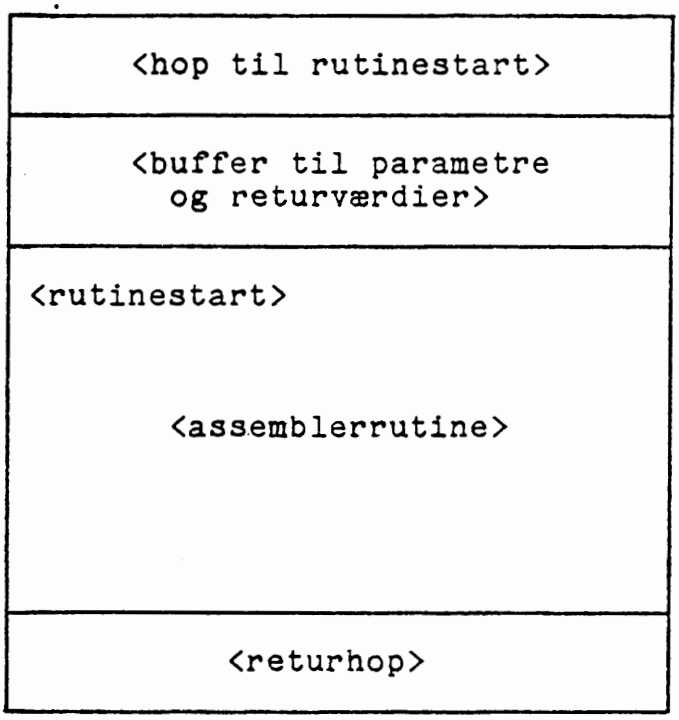
Når rutinen er placeret i teksten rutine$ kan den udføres med kaldet:

```
CALL VARPTR(rutine$)
```

- hvis startadressen er den mindste adresse i rutinen, dvs. rutine$(1).

4. Opbygning af assemblerrutiner

I denne vejledning foreslås en fast, standardiseret måde for opbygning af assemblerrutiner:



Foruden det her viste skelet, kommer visse tilpasninger af hensyn til problemfri indlæsning i Comal-programmet.

Assemblerrutinen vil efter assemblering (til relokerbart format) og sammenkædning (til absolut format, dvs. udførbart lagerbillede) kunne lagres i en fil (normalt med navneudvidelsen .COM). Denne fil kan indlæses i den ovenfor nævnte datastruktur.

Dette indebærer en række fordele: Begge kan ændres uafhængigt af hinanden og en gennemstandardiseret, fleksibel fremgangsmåde kan anvendes.

Assemblerrutinen lagres som nævnt i en fil, hvorfra den indlæses til Comal-programmet før rutinen kaldes (første gang).

5. Comals filsystem

Assemblerrutiner kan lagres i filer når de er assembleret. Før udførelse må de indlæses fra filen til Comal-programmet. Til dette formål anvendes de sædvanlige faciliteter i Comal.

Comal understøtter to filtyper:

1) ASCII filer

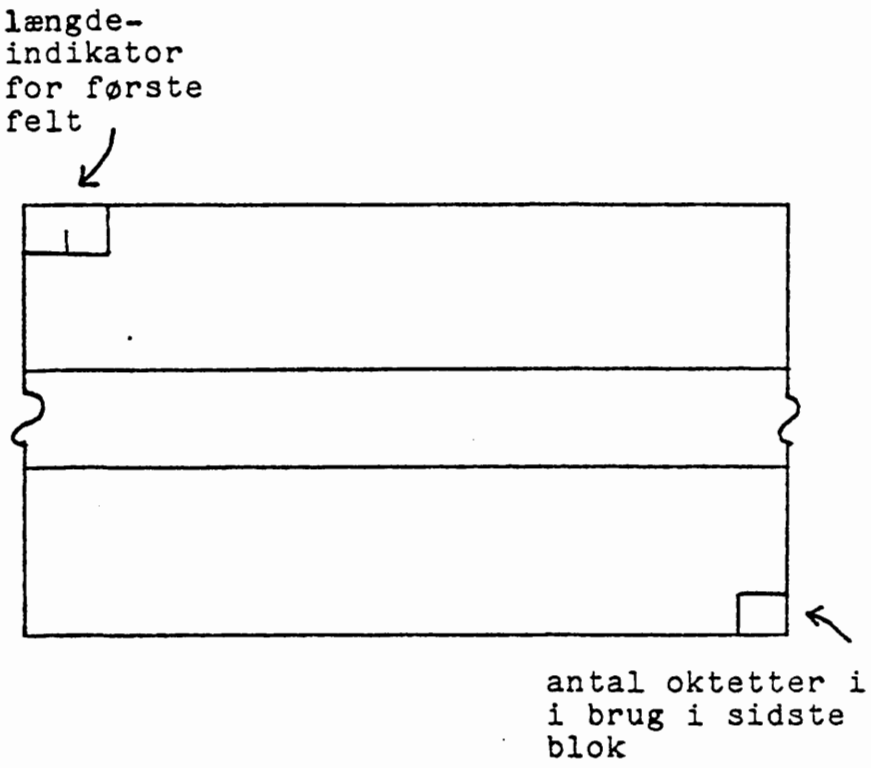
2) binære filer

ASCII-filer er sekventielle filer (såkaldte tekstfiler), hvor linieskift (CR,LF) opfattes som stoptegn og derfor ikke kan anvendes til lagring af assemblerrutiner.

Binære filer er opbygget nogenlunde således (der vises kun detaljer, som har betydning for denne sammenhæng):

De to første oktetter angiver længden af det efterfølgende felt, (et heltal i intervallet 0..65767, første oktet er mindst betydende (standard 8080 og Z80)). Den sidste oktet i den sidste blok (blokke har altid længden 128 oktetter) indeholder antallet af benyttede oktetter i den sidste blok.

Assemblerrutinen, der skal indlæses i et Comal-program, skal derfor opfattes som et felt i en binær fil. Filen bør være opbygget således:



(De her givne oplysninger er afgivet af Mogens Pelle, Metanic Aps, der har udviklet Comal-80 under CP/M).

Som følge heraf må der på assemblerniveau dannes et lagerbillede der svarer hertil.

De første to oktetter i assemblerrutinen skal derfor dannes ved direktivet (1):

```
(1)     LENGTH:  DW  LASTWD-$

            <assemblerrutine>

                 RET
        LASTWD:  NOP
```

Ved assembleringen vil indholdet af det første dobbeltord være rutinens længde i oktetter. Denne information kan tillige anvendes ved dimensioneringen af den datastruktur, der skal rumme assemblerrutinen i Comal-programmet. Den har endnu et formål: Comal forventer ved læsning af en tekst i en binær fil, at de første to oktetter er længden af den lagrede tekst.

Man bør manuelt sikre sig, at der er afsat tilstrækkelig plads i datastrukturen - Comals kontrol er noget

mangelfuld.
   For at sikre at indlæsningen kan ske uden problemer, fyldes den sidste sektor i .COM-filen med 0'er (nul) og den sidste oktet med værdien 127, (2):

```
        LENGTH:  DW    LASTWD-$

            <assemblerrutine>

                 RET
        LASTWD:  NOP
                 DS    127-((LASTWD-LENGTH+1) MOD 128)
   (2)           DB    127
```

6. Basisprogrammel til udviklingsarbejdet

Konstruktionen af assemblerrutinen sker ved indtastning af den symbolske maskinkode med et tilrådighed værende redigeringsprogram. Under nærværende udviklingsarbejde har Compas-Pascals indbyggede redigeringsdel været anvendt. Den assembler (og loader) der findes på CP/M system-disketten er ret ufleksibel og tillader kun assemblering af INTEL8080 kode. Det kan anbefales at anvende MACRO 80 fra Microsoft, der understøtter både 8080 og Z80. Dette forudsættes derfor i det følgende.

Beskrivelsen af Z80 instruktionsformat findes i: "Z80 Assembly Language Programming Manual". MACRO 80 mv. er beskrevet i "MACRO 80 Reference Manual". CP/M er beskrevet i "CP/M User Guide", men kan ikke siges at være fuldstændig. Lærebøger i Z80 programmering findes i mange udgaver i handelen. Der henvises i øvrigt til litteraturlisten bagerst i vejledningen.

Principielt bør man overlade så meget af programmeringen som muligt til Comal for at spare udviklingstid. Det kan herved være nødvendigt at opdele sine assemblerrutiner i flere små. Disse kan placeres samlet (assembleret samtidig) eller være uafhængige, selvstændig rutiner.

Anvendelse af Comal-faciliteterne PEEK, POOK, INP og OUT kan i et vist omfang reducere antallet og størrelsen af opgaver, der nødvendigvis må programmeres i maskinsprog.

7. Assembleringen

Når det symbolske assemblerprogram er skrevet og lagret i en fil (med navneudvidelsen <fil>.MAC) skal det assembleres. Dette kan ske således:

```
A>M80
*<fil>,<fil>=<fil>
*^C                              ^C betyder Ctrl-C
```

Herved produceres:

```
1 relokerbar fil: <fil>.REL
1 udskrift-fil:   <fil>.PRN
```

Udskriften fås på linieskriveren med kommandoen:

```
A>PIP PRN:=<fil>.PRN
```

Når assembleringen er tilfredsstillende kan den relokerbare fil omdannes til absolut format (lagerbillede). Dette sker ved kommandoen:

```
A>L80
*<fil>/N,<fil>/E
```

Herved fås:

```
1 fil i absolut format: <fil>.COM
```

Det er denne fil der skal indlæses som binær fil af Comal-programmet.

Ønskes filen udskrevet i hexadecimalt format, kan dette ske ved kommandoen:

```
A>DUMP <fil>.COM
```

DUMP er et program, der normalt findes på CP/M systemdisketten.

Der henvises iøvrigt til de til programmellet hørende vejledninger for yderligere detaljer.

8. Adresseafhængighed i assemblerrutiner

Programmering af assemblerrutiner kan ske på to principielt forskellige måder, enten adresseafhængig eller adresseuafhængig.

Ved adresseafhængighed forstås at assemblerrutinen er konstrueret til udførelse i et forud fastlagt adresserum.

8.1. Adresseafhængige rutiner

Adresseafhængige rutiner må på assembleringstidspunktet 'vide', hvor i lageret de kommer til at ligge når de skal udføres. Det vil sige at referencer til lageradresser nøjagtigt skal svare til det adresserum, rutinen aktuelt bliver lagret i. Med andre ord må den instruktion, der hoppes til ved kaldet

CALL <datastruktur>(<index>)

være den første, der ønskes udført. For at afgøre, hvor i lageret datastrukturen lægges af Comal-systemet, kan følgende program udføres:

```
100 // Programmet finder startadressen
200 DIM asm$ OF 150
300 // INGEN linier der reserverer lager
301 // foran linie 200
400 PRINT VARPTR(asm$)
999 END
```

Hvis udskriften f.eks. bliver 56026 må ORG direktivet se således ud:

ORG 56026-2

Ved assembleringen sikres da, at alle adresser vil være relative til den første, sat med ORG-direktivet, og derfor vil komme til at passe med de aktuelle adresser - der hvor ASM$ ligger i lageret - når rutinen skal udføres. (De -2 skyldes at længdeindikatoren ikke skal regnes med).

Ulempen er at ændringer i Comal-programmet, f.eks. ved uforvarenhed, kan medføre uoverensstemmelser med de aktuelle adresser. For at opdage sådanne uoverensstemmelser kan der indbygges kontrolpunkter i rutinen, som undersøges af Comal-programmet, når rutinen er læst

ind. Dette giver en rimelig sikkerhed for at rutinen ligger i det forventede adresserum.

Eksempel: (her i Z80 kode)

```
(1)             .Z80
(2)             ASEG
(3)             ORG   56026-2
        LENGTH: DW    LASTWD-$
(4)             JR    BEGIN
(5)             DB    123
                DB    222
(6)     BUFFER:  -
(7)     BEGIN:   -
                 -
                 -
                 -
(8)             RET
        LASTWD: NOP
                DS    127-((LASTWD-LENGTH+1) MOD 128)
                DB    127
                END
```

Forklaring:

(1) assembler Z80 instruktioner
(2) der ønskes absolutte adresser
(3) sæt adressetæller (PSC) til startadressen for rutinen
(4) indhop fra Comal, svarer til rutine$(1), eller asm$(1) i det følgende eksempel
(5) kontrolpunkter
(6) buffer til udveksling af information mellem Comal-program og assemblerrutine, se nedenfor
(7) egentlige start på assemblerrutinen
(8) retur til Comal.

Efter indlæsningen af assemblerrutinen i Comal-programmet kontrolleres om kontrolpunkterne ligger rigtigt:

```
200 // Oktet nummer 3 og 4 er kontrolpunkter
210 // Instruktionen JR BEGIN fylder 2 oktetter
220 IF ORD(asm$(3))<>123 OR ORD(asm$(4))<>222 THEN
230   PRINT "Adressekonflikt i assemblerrutine"
240   STOP
290 ENDIF
300 CALL VARPTR(asm$(1)) // = CALL VARPTR(asm$)
```

Fordelen ved adresseafhængige rutiner er, at der kan anvendes 8080 kode og at programmeringen bliver enklere.

9. Adresseuafhængige rutiner

Vanskelighederne ved adresseuafhængig kode er den finurlige måde, hvorpå man må "undgå" kendskabet til de aktuelle lageradresser på assembleringstidspunktet. På kørselstidspunktet undgås det sjældent og programmet må derfor selv finde ud af, hvor i lageret det ligger. Herefter må alle referencer til lageradresser ske via (index-)registre.

Når man een gang for alle har fundet en løsning på dette problem og anvender den til alle rutiner, svinder besværet dog væsentligt. Fordelen er, at det er uden betydning for assemblerrutinen, hvor i lageret den befinder sig, når den skal udføres. Den er derfor også upåvirket af ændringer i Comal-programmet med forandret lagerbenyttelse til følge.

9.1. Principiel opbygning af rutiner

Som det også er nævnt ovenfor, foreslås en standardiseret måde til opbygning af assemblerrutiner. Denne er som følger:

```
                .Z80
                ASEG
                ORG   100H
      LENGTH:   DW    LASTWD-$
                JR    BEGIN
      BUFFER:   DS    10D
      STATUS:   DB    0
(1)   BEGIN:
(2)     <beregn aktuelt adresserum>
(3)     <assemblerrutine>
      RETURN:   RET
      LASTWD:   NOP
                DS    < ... >
                DB    127
                END
```

Forklaring:

(1) start på assemblerrutine
(2) beregning af aktuelt adresserum gives i næste afsnit
(3) den egentlige assemblerrutine, der refererer til lageradresser ved hjælp af indexregistre

Den plads der er afsat til BUFFER og STATUS er beregnet til udveksling af data med det omgivende Comal-program. Der kan lægges parametre til rutinen inden kaldet og der kan hentes returværdier efter kaldet. Dette kan ske med programstumpen:

```
110 // Aflever parameterværdier
111 // til rutinen i 'rutine$'
120 FOR x:=3 to 12
130   rutine$(x):=CHR$(<værdi>)
140 NEXT x
```

- og tilsvarende når værdier skal hentes efter kaldet.

```
110 // Hent returværdier efter
111 // udførelsen af 'rutine$'
120 FOR x:=3 TO 12
130   t(x):=ORD(rutine$(x))
140 NEXT x
```

På tilsvarende måde med værdien af STATUS, der ligger i rutine$(13).

Linie 130 i det første eksempel kunne også være programmeret således:

```
130 POOK VARPTR(rutine$(x)),<værdi>
```

- og tilsvarende i det andet eksempel:

```
130 t(x):=PEEK(VARPTR(rutine$(x))
```

I det ovennævnte eksempel er BUFFER 10 oktetter lang. Den kan naturligvis have en vilkårlig størrelse.

9.2. Beregning af det aktuelle adresserum

Det første assemblerrutinen må undersøge er, hvor i lageret den aktuelt befinder sig og derefter give indexregistrene passende værdier, f.eks. på BUFFER og STATUS.

Dette kan gøres ved udførelse af nedenstående instruktioner:

```
(1)             LD    DE,(38H)
                LD    HL,0E9E3H
(2)             LD    (38H),HL
(3)             RST   38H
(4)  FIX:       LD    (38H),DE
(5)             POP   DE
                PUSH  HL
(6)             LD    DE,FIX-BUFFER
(7)             XOR   A
(8)             SBC   HL,DE
                PUSH  HL
(9)             POP   IX
(10)            POP   HL
                LD    DE,FIX-STATUS
                SBC   HL,DE
                PUSH  HL
                POP   IY
```

Forklaring:

(1) gem indholdet af adresserne 38H og 39H
(2) gem instruktionerne 'EX (SP),HL' og 'JP (HL)' i adresserne 38H og 39H
(3) ved denne instruktion sker følgende: gem returhop på stakken, hop til 38H og udfør: 'EX (SP),HL' (HL bliver sat til 'returadressen', dvs. FIX) - og udfør 'JP (HL)' (hop til FIX)
(4) retabler det oprindelige indhold i 38H og 39H - det ligger i register DE
(5) ryd op på stakken - der blev jo ikke returneret på normal måde med RETurn
(6) beregn afstand til BUFFERs adresse
(7) sæt CARRY-flag = 0
(8) sæt HL:=adressen på BUFFER
(9) kopier værdien til IX (IX:=HL)
(10) retabler HL så den igen er adressen på FIX og beregn nu adressen på STATUS og gem den i IY

Efter udførelsen af ovennævnte instruktionssekvens kan BUFFER og STATUS adresseres via indexregistrene IX og IY, resp.

9.3. Kontrol af adresseUafhængigheden

Det er muligt at kontrollere om der uforvarent er anvendt absolutte adresser under programmeringen. Assembler rutinen med:

```
ORG 100H
```

- og derefter igen med f.eks.:

```
ORG 0H
```

- og undersøg på udskriften de hexadecimale koder - dvs. instruktionsækvivalenterne - (de står til venstre i udskriften af <fil>.PRN). Har de ændret sig i de to assembleringer er der anvendt absolutte adresser.

9.4. Begrænsninger

Adresseuafhængig programmering kan kun ske i Z80 assembler. Af særlig interesse er her instruktionerne:

JR (Jump Relative) og registrene IX, IY

Ved adresseuafhængig programmering skal alle hop ske relativt og al adressering af data skal ske via (indeks)-registre, f.eks. IX,IY og HL. Man kan ikke anvende instruktioner, der refererer til absolutte adresser, altså heller ikke f.eks. CALL og 'LD DE,<etikette>'.

Af særlig betydning bliver derfor stakken og dermed instruktiorne PUSH og POP. Det er klart at stakken skal afleveres til Comal som den blev modtaget!

Der kan også tænkes anvendt adressemodifikation i særlige situationer. Denne teknik er noget tricky og er derfor ikke medtaget her.

10. Eksempel på adresseuafhængig rutine

Nedenstående rutine kommunikerer med en seriel port på Comet mikrodatamaten. Hensigten med eksemplet er at vise en rutine af en vis størrelse og de dertil hørende procedurer i Comal, som initierer og kalder rutinen. Disse Comal-procedurer er vist i det efterfølgende eksempel.

```
MACRO-80 3.36  17-Mar-80      PAGE  1

                ; COMAL VERSION
                ; Program name: SAR.MAC (source)
                ;               SAR.REL (relocated)
                ;               SAR.PRN (list)
                ;               SAR.COM (absolute)
                ;
                ; Programmer: John Plate
                ; Object computer: MPS 3000
                ; Language: Z80 assembler
                ;          Subroutine to Comal-80
                ; Version: january 1983
                ;
                ;
                ; This routine reads a weightresult from a Sartorius 2006 MP
                ; balance. It is created for inclusion as an assembly (Z80)
                ; procedure in a host program.
                ;    The bits PRINT and BUSY must be set (high) on the flagport,
                ; before calling the procedure.
                ;    The PRINT and BUSY flags are reset (low) for ca 50 ms, and
                ; are set (high) again.
                ;    Then the routine awaits the TACT-signal, and stores the
                ; value in the dataport, when it is found (high). The waiting
                ; for the TACT-signal may time out after repeated reading of the
                ; flagport for some seconds.
                ;    The routine is terminated normally after recieving 10
                ; pulses of data, 8 bcd-digits and, special signs and the
                ; place of decimal point.
                ;    The procedure is not dependent of the actual environment,
                ; and may therefore be placed anywhere in memory.
                ;
                ;                       Laborantskolen i Roskilde, dec, 1982 .
                ;
```

```
                        ; Tecnical conditions:
                        ; 1) PIO processor aust be init'ed before routine call
                        ; 2) The bits BUSY and PRINT aust be set (high) before calling
                        ; 3) 2.5 MHz CPU due to timing loop
                        ;
                        ; Registers in use: A, B, C, D, E, HL, IX, IY:
                        ;
                        ; Exit:
                        ;   BUSY and PRINT are set (high) again.
                        ; Result:
                        ;   status= 64: OK, the buffer (0..9) are filled with data from
                        ;               the balance. The digit-addresses are stripped off
                        ;   status=128: No data was recieved from the balance and the
                        ;               procedure timed out. This failure may be due to
                        ;               long time without stability in the balance, or
                        ;               balance overflow. The balance awaits stability
                        ;               before sending data.
                        ;   status=0..9 digit address error. The status value is the
                        ;               value of the digitcounter.
                        ;
                        ;
                        ;
                                .Z80                ; ASSEMBLE Z80 OPCODES
0000'                           ASEG                ; ABSOLUTE ADDRESSES
                                ORG   100H          ; PC=100H
                        ;
0000                    DATA    EQU     0D          ; SET CONSTANTS ACCORDING TO
0001                    FLAGS   EQU     1D          ; SARTORIUS MANUAL
0010                    BUSY    EQU    16D          ;
0008                    PRINT   EQU     8D          ;
0040                    TACT    EQU    64D          ;
0080                    DATAEND EQU   128D          ;
                        ;
                        ;
0100    009F            LENGTH: DW    LASTWD-$      ; LENGTH INDICATOR FOR COMAL FILE-INPUT
0102    18 0B                   JR    BEGIN         ; JUMP TO PROGRAM START
                        ;
                        ; Buffer space
                        ;
0104                    BUFFER: DS    10D           ; BUFFER FOR 10 BYTES
010E    00              STATUS: DB    0             ; STATUS VALUE UPON EXIT
```

```
                                ;
                                ;
                                ; First, load indexregisters with current buffer- and status
                                ; addresses. 1) save 38H, and 39H. 2) put instructions
                                ; EX (SP),HL  JP (HL) into 38H and 39H. 3) RST 38H. 4) restore
                                ; 38H and 39H.
                                ;
010F    ED 5B 0038      BEGIN:  LD      DE,(38H)        ; SAVE CONTENTS
0113    21 E9E3                 LD      HL,0E9E3H       ; LOAD INSTRUCTIONS
0116    22 0038                 LD      (38H),HL        ; PUT INSTRUCTIONS
0119    FF                      RST     38H             ; PC INTO HL AND RETURN
011A    ED 53 0038      RET:    LD      (38H),DE        ; RESTORE CONTENTS
011E    D1                      POP     DE              ; CLEAN STACK
                                ;
                                ; HL:=address(RET). Now, modify HL to point at BUFFER
                                ;
011F    E5                      PUSH    HL              ; SAVE RET
0120    11 0016                 LD      DE,RET-BUFFER   ; LOAD OFFSET TO BUFFER
0123    AF                      XOR     A               ; RESET CARRY
0124    ED 52                   SBC     HL,DE           ; AJUST HL=BUFFER
                                ;
0126    E5                      PUSH    HL              ;
0127    DD E1                   POP     IX              ; IX=BUFFER ADDRESS
0129    E1                      POP     HL              ; HL:=ADDR(RET)
012A    11 000C                 LD      DE,RET-STATUS   ; MODIFY HL
012D    ED 52                   SBC     HL,DE           ; HL=STATUS ADDRESS
012F    E5                      PUSH    HL              ;
0130    FD E1                   POP     IY              ; IY= STATUS ADDRESS
                                ;
                                ; IX=buffer address, IY=status address
                                ; Now, reset PRINT and BUSY flags
                                ;
0132    DB 01                   IN      A,(FLAGS)       ;
0134    D6 18                   SUB     PRINT+BUSY      ; REMOVE PRINTFLAG
0136    D3 01                   OUT     (FLAGS),A       ;
                                ;
                                ; Let PRINT and BUSY be lowered for 50 ms
                                ;
0138    06 FF                   LD      B,255D          ; B:=255
013A    0E 28                   LD      C,40D           ; C:=40
013C    05              COUNT:  DEC     B               ; COUNT TO 256
```

```
013D  20 FD                   JR    NZ,COUNT      ; 4+12*256 T-STATES=1.6 MS
013F  0D                      DEC   C             ;
0140  20 FA                   JR    NZ,COUNT      ; REPEAT B TIMES
                        ;
0142  F6 18                   OR    PRINT+BUSY    ; SET PRINTFLAG AGAIN
0144  D3 01                   OUT   (FLAGS),A     ;
                        ;
                        ; PRINT=1, BUSY=1. Now, fetch digits in the following bcd-stream
                        ; Wait for the first tact-signal (high >=200 ys)
                        ; - it may be delayed for some seconds...
                        ;
0146  06 FF                   LD    B,255D        ; START INITIAL LONG WAIT
0148  0E FF                   LD    C,255D        ;
014A  16 0A                   LD    D,10D         ; ca 20 seconds
014C  DB 01             LWTACT: IN    A,(FLAGS)     ;
014E  E6 40                   AND   TACT          ; IF TACT FLAG THERE?
0150  20 0B                   JR    NZ,FETCH      ; GO AND FETCH!
0152  05                      DEC   B             ; B:=B-1
0153  20 F7                   JR    NZ,LWTACT     ; = 46 T-STATES
0155  0D                      DEC   C             ;
0156  20 F4                   JR    NZ,LWTACT     ;
0158  15                      DEC   D             ;
0159  20 F1                   JR    NZ,LWTACT     ;
-------------------------------------------------------------------
015B  18 3D                   JR    TIMOUT        ; DON'T WAIT LONGER
                        ;
                        ; Fetch loop
                        ;
015D  1E 00             FETCH:  LD    E,00D         ; DIGIT-ADDRESS COUNTER
015F  06 FF             NEXT:   LD    B,255D        ; B:=255
0161  05                WTACT:  DEC   B             ; B:=B-1
0162  28 2B                   JR    Z,ERROR       ; IF B=0 THEN TIMEOUT
0164  DB 01                   IN    A,(FLAGS)     ;
0166  E6 40                   AND   TACT          ; GET TACT FLAG
0168  28 F7                   JR    Z,WTACT       ; WAIT IF TACT=0
                        ;
016A  DB 00                   IN    A,(DATA)      ;
016C  47                      LD    B,A           ; SAVE A IN B
016D  E6 F0                   AND   0F0H          ; MASK OUT DATA
016F  CB 0F                   RRC   A             ;
0171  CB 0F                   RRC   A             ;
0173  CB 0F                   RRC   A             ;
```

```
0175   CB 0F                    RRC    A            ; SHIFT HALF-BYTE
0177   DD 77 00                 LD     (IX+0),A     ; PUT DATA INTO BUFFER
017A   78                       LD     A,B          ; RESTORE DATA IN A
017B   E6 0F                    AND    0FH          ; EXTRACT DIGIT-ADDRESS FROM A
017D   BB                       CP     E            ; COMPARE DIGIT-ADDRESS
017E   20 0F                    JR     NZ,ERROR     ; =ADDRESS-ERROR
0180   FE 09                    CP     09D          ; IF 10 DIGITS RECIEVED
0182   28 10                    JR     Z,FINISH     ; THEN GO BACK
                         ;
                         ; Prepare for next data
                         ;
0184   DD 23                    INC    IX           ; IX:=IX+1
0186   1C                       INC    E            ; E:=E+1, DIGIT ADDRESS-COUNTER
0187   DB 01             WNOTACT: IN     A,(FLAGS)    ; AWAIT TACT SIGNAL OFF
0189   E6 40                    AND    TACT         ;
018B   20 FA                    JR     NZ,WNOTACT   ;
                         ;
                         ; Tact signal off now. Take next data.
                         ;
018D   18 D0                    JR     NEXT         ;
                         ;
                         ; Termination:
                         ;
018F   FD 73 00           ERROR: LD     (IY+0),E     ; STATUS=ERROR-DIGIT
0192   18 0A                    JR     RETURN       ;
                         ;
0194   FD 36 00 40        FINISH: LD    (IY+0),64D   ; STATUS=OK
0198   18 04                    JR     RETURN       ; GO BACK
                         ;
019A   FD 36 00 80        TIMOUT: LD    (IY+0),128D  ; STATUS=TIMEOUT
                         ;
                         ; Go back to Comal
                         ;
019E   C9                 RETURN: RET                ;
019F   00                 LASTWD: NOP                ;
01A0                             DS    127-((LASTWD-LENGTH+1) MOD 128)
                         ; Fill the rest of a file segment with zeroes. The last byte
                         ; should be 1H.
                         ;
01FF   7F                        DB    127D
                                 END
```

11. Eksempler på Comal-procedurer

Disse Comal-procedurer administrerer grænsefladen til mellem en vægt og mikrodatamaten. De er medtaget her for at vise, hvordan de to niveauer samarbejder. Med de viste funktioner, vil nedenstående program kunne udføres:

```
100 v:=0; s:=0
110 DIM sv$ of 10
120 WHILE s=0 DO
130   EXEC tarervægt
140   INPUT "Tast return, når der skal vejes ":sv$
150   EXEC vejning (v,s)
160   PRINT "Det vejede ";v;"gram"
170 ENDWHILE
180 PRINT "Status var ";s
999 END
```

```
PROC READASM(REF ASMPROC$, REF ENTRY, REF ASMNAVN$) CLOSED
 // Denne procedure indlæser en assemblerrutine i en tekst-
 // variabel, ASMPROC$. Assemblerrutinen hentes fra en fil,
 // hvis navn skal være i tekstvariablen ASMNAVN$. Navne-
 // udvidelsen skal være COM.
 //    Referenceparametren "ENTRY" sættes til startadressen
 // for assemblerrutinen.
 OPEN FILE 9, ASMNAVN$, READ
 READ FILE 9: ASMPROC$
 CLOSE FILE 9
 IF ORD(ASMPROC$(LEN(ASMPROC$)-2))<>201 THEN
  // er RET den sidste instruktion i ASMPROC$?
  PRINT
  PRINT "FEJL >>>> assemblerrutine i uorden"
  STOP
 ENDIF
 ENTRY:=VARPTR(ASMPROC$)
ENDPROC READASM
```

```
PROC I9999
 DIM ASMRUT$ OF 200
 EXECINIT:=0 // er udefinered hvis initvægt ikke er kaldt
 VEJERUTINE:=0
ENDPROC I9999
//
PROC INITVÆGT //CLOSED
 // Denne procedure klargør grænsefladen fra datamat til vægt.
 // Den skal kaldes før øvrige operationer udføres på vægten.
 // Lyset tænder i vinduet på vægten når denne rutine udføres.
 EXEC I9999
 DIM ASMNAVN$ OF 20
 ASMNAVN$:="SAR.COM"
 EXEC READASM(ASMRUT$,VEJERUTINE,ASMNAVN$)
 // klargør grænseflade
 BUSY%:=16; PRINTFL%:=8; TARA%:=4
 OUT 2, 79 // sæt dataport=input
 OUT 3, 207 // sæt flagport=mixed
 OUT 3, 227 // sæt flagport=4,3,2=output, resten=input
 OUT 3, 7 // ingen interrupt ønskes
 OUT 1, TARA%+PRINTFL%+BUSY% // sæt outputflags (high)
 // Herefter tænder vinduet på vægten.
ENDPROC INITVÆGT
//
PROC TARERVÆGT CLOSED
 GLOBAL EXECINIT
 // Denne procedure tarerer vægten, dvs nulstiller den.
 // Ved tareringen bip'er vægten.
 TARA%:=4; FLAGPORT%:=1
 T%:=INP(FLAGPORT%)
 OUT FLAGPORT%, T%-TARA%
 FOR Q:=1 TO 500 DO
 NEXT Q
 OUT FLAGPORT%, T%
ENDPROC TARERVÆGT
```

```
PROC VÆGTTÆNDT(REF V) CLOSED
 // Denne procedure undersøger om vægten er tændt. Værdien
 // af parametren V er efter kaldet 1 hvis vægten er tændt,
 // ellers 0.
 V:=(INP(3) DIV 2) MOD 2
ENDPROC VÆGTTÆNDT
//
PROC VEJNING(REF V, REF STATUS) CLOSED
 GLOBAL EXECINIT, VEJERUTINE, ASMRUT$
 // Denne procedure henter det aktuelle vejeresultat fra
 // Sartorius vægten. Efter kaldet har parametren V værdien.
 // Vejningen sker når vægten er stabil (rolig). Hvis der
 // opstår forstyrrelser eller der er mere end 170 gram på
 // vægtskålen, kan der ikke hentes noget vejeresultat fra
 // vægten.
 //   Efter kaldet skal det derfor kontrolleres om parametren
 // STATUS er 0, hvilket betyder at alt er iorden. Har STATUS
 // enten værdi end 0, er der opstået en fejl undervejs.
 CALL VEJERUTINE
 STATUS:=ORD(ASMRUT$(13))
 IF STATUS=64 THEN STATUS:=0
 V:=0
 FOR Q:=10 DOWNTO 3 DO
  V:=V*10+ORD(ASMRUT$(Q))
 NEXT Q
 V:=V/10^ORD(ASMRUT$(12))
 IF ((ORD(ASMRUT$(11)) DIV 8) MOD 2)=1 THEN V:=-V
ENDPROC VEJNING
```

12. Bilagsmateriale til eksemplerne

For fuldstændighedens skyld aftrykkes her de relevante afsnit af vejledningen til vægten.

13. Technical data

Model	2006 MP6	
Weighing range	170	g
Readability	0.1	mg
Standard deviation	≦0.1	± mg
Electronic measuring range	30	g
Max. linearity deviation	≦0.1	mg
Built-in switch weights, fully automatic	20 ... 140	g
Accuracy of built-in weights	≦0.05	± mg
Taring range (subtraction)	170	g
Taring time approx.	20	ms
Measuring time in the electronic measuring range	3	sec.
Integration time	automatically optimized	
Ambient temperature range	283 K ... 313 K (10°C ... 40°C)	
Sensitivity drift	≦ ± $3 \cdot 10^{-6}$ /K	
Line voltages	100, 120, 220, 240 V (selectable) − 15% ... + 10% 50/60 Hz	
Input approx.	10 VA	
Data output	BCD 1-2-4-8 code bit parallel/decade-serial	

14. Data Output (Standard)

Data are released after a low signal > 50 ms (print command) which has to be triggered externally (pin 15).
Before data output the balance interrogates the busy input (pin 14) whether the data receiver is ready for data acceptance.

Busy = low: data receiver not ready } for data
Busy = high: data receiver ready } acceptance

If a print command is released when busy = low, data output takes place automatically as soon as busy = high and the stability indicator signals stability (standard models).

In case data are to be released before the balance reaches stability, select operating mode using switch "Stability Range" (position 6 or 7) (ill. 3, item 7).

Stability indicator (g) and display are blanked out until balance has reached stability.

In case of overflow (balance display indicates —H.) no data are released.

Data are valid with clock = high (pin 12).
At the end of each data transmission the signal "data end" (pin 11) is released.

The data block is released only once.
Information is available at the output in a bit-parallel and decade-serial format.

Data lines: A = 2^0 – pin 1
B = 2^1 – pin 2
C = 2^2 – pin 3
D = 2^3 – pin 4

Adress lines: A = 2^0 – pin 6
B = 2^1 – pin 7
C = 2^2 – pin 8
D = 2^3 – pin 9

Data are released in 10 decades.

time $t_0 - t_7$: weight value $10^0 - 10^7$
(10^0 = lowest decade)

Time t_8: special characters:
% stability indicator (only in connection with data input)
g stability indicator
Vz signs

Time t_9: DP decimal point

The balance can be externally tared (pin 16) with a low signal > 50 ms.
Supply voltages of + 5 V (pin 19) and 8.5 V (pin 17 and 18) are available at the data output.

The connections pin 1 . . . pin 11, pin 13 . . . 34 are designed for data input keyboards 714301, 71430/1/2/3/4, and data print 7179, 71810/1/2/3.

If the balance is operated in connection with a data input, data print, or special program, the constants 0 . . . E (hex-code) which are defined in the program can be released.

Data output is valid with clock C = high (pin 13).

For the digital inputs and outputs of the data output the specifications of the C-MOS-series 4000 are applicable.

Datenausgang Data Output

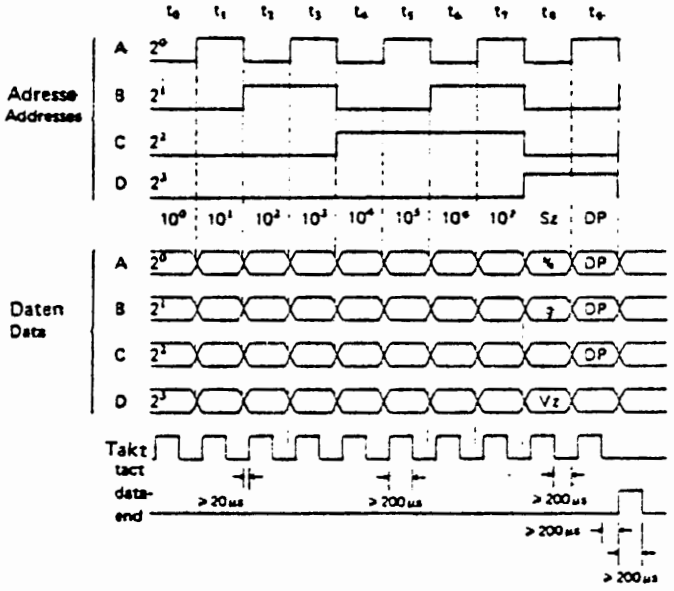
Pin
- 1 – 2^0 } Daten / Data
- 2 – 2^1
- 3 – 2^2
- 4 – 2^3
- 5
- 6 – 2^0 } Adresse / Addresses
- 7 – 2^1
- 8 – 2^2
- 9 – 2^3
- 10
- 11 – Data-end
- 12 – Takt/Tact
- 13
- 14 – Busy
- 15 – Print
- 16 – Tara ext.
- 17 – } 8.5 V~
- 18 – }
- 19 – + 5 V
- 20 – CM

Erforderlicher Anschlußstecker: 3M 3461-0001
Codierung zwischen Pin 3 und 5

Connection required: 3M 3461-0001
Coding between pin 3 and pin 5

Datenausgang-Zeitdiagramm

Data output-timing diagram



Sz = Sonderzeichen/Special signs

2^0 high = %
2^1 high = g
2^3 high = −Vz

DP = Dezimal-Punkt/Decimal point

2^2	2^1	2^0		
0	0	0	±	x x x x x x.
0	0	1	±	x x x x x. x
0	1	0	±	x x x x. x x
0	1	1	±	x x x. x x x
1	0	0	±	x x. x x x x
1	0	1	±	x. x x x x x
1	1	0	±	.x x x x x x

Positive Logik/Positive Logic
high = + 3,5 V . . . 5 V ; C-MOS komp.
low = 0 V . . . + 1,5 V ; C-MOS comp.

Bemerkungen: Daten können vom Empfänger auf beiden Flanken des Taktsignals übernommen werden.

Remarks: Data can be accepted at the leading or trailing edge of the tact.

13. Litteraturliste

Z80 Microcomputer Data Book
Mostec 1981

Z80 Assembly Language Programming Manual
Zilog 1977

Thom Hogan:
CP/M User Guide
Osborne/McGrav-Hill, 1981