een volwassen computer voor beginners

boek 4

uitgeversmij.
elektuur b.v.

Junior-computer 4

een volwassen computer voor beginners

A.Nachtmann
G.H.Nachbar

Uitgave:
Uitgeversmaatschappij Elektuur B.V.
Postbus 75 6190 AB Beek (L)

Eerste druk oktober 1981

Printed in the Netherlands.
ISBN 90 70160 19 6

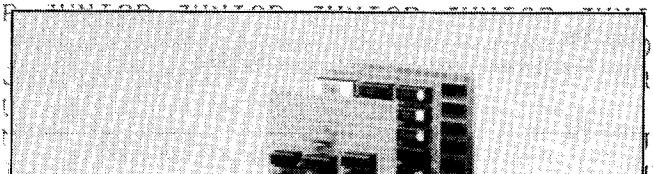
Junior-computer 4 . . .

. . . een boek dat volledig in het teken staat van software. Hoe kun je "ego-software" snel, nee: supersnel aan de junior-computer opgeven? En hoe zitten de diverse nieuwe systeemprogramma's in elkaar?
Allereerst maken we, in hoofdstuk 13, kennis met een spiksplinternieuw systeemprogramma. Het heet kort maar krachtig "PME" en kombineert de voordelen van de standaard-editor met de voordelen van PM. Met als gevolg dat het nu helemaal een fluitje van een cent is om grote gebruikers-programma's aan de junior-computer op te geven en ze vervolgens te assembleren.
En dan volgen drie hoofdstukken met een gedetailleerde beschrijving, byte voor byte, van de systeemprogramma's PM (hoofdstuk 14), PME (hoofdstuk 15) en TM, inklusief de twee cassette-"super-subroutines" (hoofdstuk 16).
Helaas is het in boek 3 toegezegde hoofdstuk over de VIA komen te vervallen. Waarom? Ten eerste omdat er geen plaats meer voor is en ten tweede omdat er over de 6522-VIA erg veel goede literatuur* bestaat.
In de aanhangsels, achterin dit boek, zijn uitgebreide programma-listings van de genoemde systeemprogramma's opgenomen.

A. Nachtmann
G.H. Nachbar

* Literatuur over de 6522-VIA:
R6522-Datasheet Rockwell;
SY6522/SY6522A-Datasheet Synertek;
Section 6 R6500 Hardware Manual Rockwell

Inhoud



13

Supersnel editen en assembleren

De PM-Editor

Het opgeven van een programma aan de computer is stukken minder leuk dan het bedenken ervan. Dus moeten we aan het ene zo min mogelijk tijd besteden en aan het andere zo veel mogelijk tijd. Want een hobby, óók de junior-computer-hobby, is bedoeld als aangenaam tijdsverdrijf.
De opgave van een programma verloopt al veel sneller als gebruik wordt gemaakt van de in boek 2 beschreven editor en assembler. En voortaan hoeven we een programma maar één keer byte voor byte op te geven. Want de cassette-band onthóudt het wel; voor ons en voor de computer.
Dat is nog niet alles. Het kan nòg sneller. In dit hoofdstuk staat hóe.
In boek 3 is de sprong gemaakt van standaard-monitor naar PM. In dit hoofdstuk maken we net zo'n sprong: van standaard-editor naar PM-Editor. De naam (afgekort: PME) zegt het al: een editor die is gebaseerd op PM. Die dus niet meer met het kleine toetsenbord werkt maar met het grote. Die dus niet meer iets laat zien op zes displays maar op het tv-scherm of op het papier van een printer. Maar die nog steeds wèl is gebaseerd op het gebruik van hexadecimale labels. En dat werkt met name in de ontwikkelingsfase van een programma beduidend sneller dan met een "grote" editor en assembler.
Er zijn nu meer toetsen beschikbaar. Die benutten we voor een deel voor allerlei nieuwe, komfortabele toetsfunkties. En er kan nu wel iets meer dan één instruktie tegelijk worden getoond.

Daar komt nòg iets bij: het aantal toetshandelingen is ten opzichte van de standaard-editor met ca. 30% gereduceerd.
Lees dit hoofdstuk en u weet alles van PME. Van de gelegenheid maken we gebruik om een aantal voorbeeldprogramma's met PM-subroutines te behandelen, als voortzetting van de programma's van hoofdstuk 12 van boek 3.
Het lezen van dit hoofdstuk en het leren omgaan met PME kosten u tijd. Tijd echter, die u snel zult terugverdienen. Dàt zal de tijd snel leren.

Junior groeit.
Weliswaar niet meer lichamelijk. Dat hebben we achter de rug in boek 3.
Maar wel geestelijk.
De software dus.
Zowel de TM-EPROM als de PM-EPROM vertoonden nog een leemte van pakweg driekwart K. Dat kon natuurlijk niet zo blijven. En dat is nu ook niet meer het geval voor de PM-EPROM. Want de resterende geheugenplaatsen zijn nu gereserveerd voor het systeemprogramma PME. Voluit: PM-Editor. Een editor dus die is gebaseerd op PM. Op basis van het grote toetsenbord en met meer ruimte om iets te laten zien; zowel in de breedte (meer op een regel) als in de lengte (meer regels).
Laten we eerst maar eens gaan bespreken waarom er zo nodig een nieuwe editor moest komen. Het argument dat er nog plaats voor is in een EPROM is op zich natuurlijk niet doorslaggevend. Want er zijn genoeg andere aanvullende systeemprogramma's denkbaar, die óók interessant zijn: óók die nu nog niet geheel gevulde TM-EPROM is geen oneindig lang leven beschoren.

Waaròm eigenlijk PME?

Wat hebben we al? De standaard-editor van de hoofdstukken 5 en 8 van boek 2. We beginnen met het vaststellen van één groot voordeel van die editor. Dat is het gebruik van hexadecimale labels. Het alternatief is: labels, bestaande uit woorden van maximaal zes letters. Labels, zoals u die tegen komt in de listings achterin dit boek en achterin boek 2. Listings, die zijn gemaakt met een grote assembler/editor. Zo'n grote editor annex assembler is wel een stukje groter dan de driekwart K van PME. Er is dus geen EPROM voor beschikbaar (afgezien van buskaart-EPROM). En als zo'n grote assembler/editor al zal worden gerealiseerd dan gaat de voorkeur uit naar een programma op de band in plaats van in relatief dure EPROM.
Maar er is nog een andere reden om niet al te hard te denken aan een grote editor.
Men moet daarbij namelijk uitgaan van een behoorlijk volledige dokumentatie vooraf van het gebruikersprogramma op papier. Alle (echte!) labels moeten bekend zijn. En verder moeten alle gebruikte geheugenplaatsen met een bekende naam (bijvoorbeeld POINTL of BYTES) vooraf worden

opgegeven; "gedeklareerd" heet dat deftig. Hetzelfde geldt voor subroutines waarvan het adres al vastligt, voor I/O, enzovoorts.
Stel nou dat u om vier uur een programma heeft bedacht en om vijf uur wilt weten of het werkt (dat wordt dus overwerken), of niet. Ga er verder van uit dat de kans groot is dat er naderhand nog wat moet worden gesleuteld aan het programma. We stellen dus vast dat: a) bij gebruik van een grote editor/assembler veel werk vooraf moet worden gedaan, en b) dat de kans groot is dat een deel van dat werk voor niets is verricht.
Geef mij dan maar een editor die werkt met hexadecimale labels, zoals de standaard-editor en zoals PME. Gegarandeerd zit dat programma van daarnet er om half vijf in (en zijn we vervolgens tot half zes bezig om het nèt nog even mooier, beter of uitgebreider te maken . . .).
Konklusie: werk met hexadecimale labels. Met name in de scheppingsfase van een gebruikersprogramma biedt dat alleen maar voordelen. Bovendien hoeven we dan geen nieuwe assembler te maken. De bestaande assembler van hoofdstuk 9 van boek 2 blijft voor de volle 100% bruikbaar. Er is slechts één mogelijk nadeel verbonden aan het gebruik van hexadecimale labels. Men moet geen labelnummers kiezen die gelijk zijn aan het rechter byte ADL van een absoluut adres; een adres dus dat al vóór het editen en assembleren vastligt. Maar dat soort weetjes heeft men gauw genoeg onder de knie. Dus dat is nauwelijks een nadeel te noemen. Bovendien is het aantal beschikbare labels niet oneindig groot: een dikke tachtig. Maar daar kunt u werkelijk làppen van programma's mee maken.
Tot zover een belangrijk voordeel van de oude en de nieuwe editor. Er zijn ook nadelen verbonden aan de standaard-editor. Die nadelen zijn tegelijkertijd de bestaansgronden van PME. We zetten de nadelen op een rijtje:

- Allereerst natuurlijk het beperkte aantal toetsfunkties en de beperkte weergavemogelijkheden. Dat is geen schande, want de standaard-editor hoort bij de standaard-junior-computer. Misschien moeten we het anders zeggen: Gegeven de nieuwe mogelijkheden kwa aantal toetsen en kwa weergave (na de uitbreidingen van boek 3) ligt het voor de hand om de standaard-editor niet langer als "standaard" te beschouwen.
- **Foutmeldingen.** De standaard-editor kent slechts één foutmelding: de kreet "EEEEEE" op het display. Dat is overigens alleen zichtbaar zolang de ingedrukte toets die aanleiding gaf tot de foutmelding, nog is ingedrukt. Er is dus een behoorlijk grote kans dat men die "EEEEEE" helemaal niet in de gaten heeft gehad. Een duidelijkere foutmelding is dus nooit weg. Verder willen we graag zien wàt er dan wel fout is is gedaan of gegaan.
- **Meer foutmeldingen.** Aan de standaard-editor kleeft één groot nadeel: **er wordt niet vastgesteld dat het via BEGAD en ENDAD gespecificeerde geheugenbereik vol is na het intoetsen (opgeven) van een te groot aantal labels en instrukties.** Dat is erg vervelend. Bijvoorbeeld in het voorbeeldprogramma PLAY op de pagina's 52 . . . 55 van boek 2. Het beschikbare geheugenbereik ØØØØ . . . ØØEØ is ruimschoots voldoende voor de geassembleerde versie van PLAY (labels eruit) maar niet voor de versie met de labels er nog in. We komen dan net een paar plaatsen te kort: de afsluitende 77 komt op geheugenplaats ØØE3 (BEGADH!). Daar is overigens wel een mouw aan te passen: laat het label 9Ø (PLAY) vervallen. Dat mag, want er wordt tòch niet naar gesprongen. Doen we dat

dan is er plaats genoeg, óók voor de versie met labels . . . ware het niet dat er tòch weer te weinig geheugenplaatsen beschikbaar zijn voor het eerste label dat tijdens het assembleren wordt gevonden (zie alvast figuur 1). Remedie: zet PLAY neer op pagina Ø2. Dáár is plaats genoeg. Overigens: we hebben nu de beschikking over een aaneengesloten RAM-geheugenbereik van Ø2ØØ . . . Ø7FF. Dat is anderhalve kilo. Het gevaar van een "volle geheugenherberg" is nu aanzienlijk minder groot geworden.

- **Minder toetswerk.** Verreweg de meest gebruikte toetsfunktie is INPUT. Dat wil zeggen: meestal gaat het om de opgave van een instruktie die in het geheugen op een plaats of plaatsen komt direkt na de plaats of plaatsen waarop zich de instruktie bevindt die als laatste op het display zichtbaar was. Dus het meeste toetswerk vindt plaats in het kader van het achter elkaar in het geheugen laden van een hele reeks instrukties en labels. Is het nou echt nodig om voor elke instruktie en label een INPUT-toets in te drukken? Nee, dat is niet nodig. Je kunt het systeemprogramma zo opzetten dat bij de opgave van numerieke data Ø . . . F automatisch sprake is van de INPUT-toetsfunktie, tenzij er een andere toetsfunktie is gekozen waarvan de uitvoering eveneens afhangt van de opgave van numerieke data. Bijvoorbeeld INSERT of SEARCH. Dit principe van "automatisch INPUT bij numerieke data, tenzij . . ." levert een aanzienlijke reduktie op van het aantal toetshandelingen.

PME. Aangenaam!

Bij de standaard-editor wordt het display afwisselend gebruikt voor de weergave van een toetsaktie van de gebruiker en voor de weergave van een reaktie van de junior-computer bij monde van de editor. Een totaaloverzicht ontbreekt dus. Herinneren we ons PM van boek 3, dan weten we dat er meer op een regel kan en dat er meer regels tegelijk zichtbaar gemaakt kunnen worden. Kortom: er is meer te zien. Verder herinnert u zich die kwestie van de hardware-echo in de UART van de Elekterminal. Die echo zorgt ervoor dat een ingedrukte toets automatisch op het tv-scherm of op het papier zichtbaar wordt. Zonder dat daarvoor één byte software nodig is.
Konklusie: het is dus mogelijk om op het tv-scherm of papier te werken met twee kolommen. Eén kolom is bestemd voor de toetsakties van de gebruiker. En de andere kolom is er voor de reaktie van de junior-computer via zijn spreekbuis PME. Dus de ene kolom vertelt ons wat we hebben gedaan en de andere kolom vertelt ons wat de junior-computer ervan vindt. **Dus twee kolommen. De linker kolom is bestemd voor de reaktie van de computer. De rechter kolom is bestemd voor de aktie van de gebruiker. Een aktie van de gebruiker uit zich op één bepaalde regel van de rechter kolom. De reaktie van de computer speelt zich af op of vanaf de volgende regel op de linker kolom.** (Uitzondering: het afdrukken van labels. Zoals zal blijken wordt daarvoor het tv-scherm over de volle breedte gebruikt).
Terugmeldingen. Bij PM hebben we verschillende soorten terugmeldingen leren kennen. Denk aan "JUNIOR" en "WHAT?". Bij PME is dat ook het geval. Er zijn algemene terugmeldingen en foutmeldingen. **De foutmeldin-**

gen zijn gespecificeerd. Dan weet je als gebruiker tenminste wàt je dan wel fout hebt gedaan.
Ook het adres zichtbaar. Het is een kleine moeite om, nu er toch zàt ruimte is, als er een instruktie wordt weergegeven (lees: afgedrukt) om dan meteen maar het adres erbij af te drukken waarop de opcode staat van die instruktie. Dat levert zeer nuttige extra informatie op, bijvoorbeeld over de hoeveelheid geheugen die op een bepaald tijdstip is "opgesoupeerd" (met inbegrip van de labels).
En nog veel meer. Zoals: sneller en komfortabeler BEGAD en ENDAD opgeven, vier startadressen in plaats van twee, veel meer toetsfunkties, sneller assembleren, adresinformatie van de labels, enzovoorts.
Hopelijk hebben we u erg nieuwsgierig gemaakt. Zullen we dan maar verder gaan?

PME. Een grondige kennismaking

A. Algemeen

Het systeemprogramma PME bevindt zich in de PM-EPROM. Dat is IC5 op de interface-kaart. Het bijbehorende bestelnummer was ESS 507 voor de EPROM met alleen PM en is ESS 507N voor de volle EPROM, dus met PM plus PME plus nog wat losse bytes (zie elders in dit boek). Sinds juni 1981 worden de door Elektuur in het kader van de ESS (507) aangeboden EPROMs "stilzwijgend" behandeld als ESS 507N. Mensen van het eerste uur kunnen hun 507-EPROM alsnog laten "opvoeren" tot een 507N-EPROM. Hoe dat moet kunt u in de nieuwste uitgave van het tijdschrift Elektuur lezen.
Het programma PME beslaat de geheugenplaatsen 14F8 . . . 17FD van de PM-EPROM. In PME wordt gebruik gemaakt van een aantal PM-subroutines. Dit houdt in dat **PME moet worden opgestart via PM (keuze uit vier startadressen).** Zou men PME opstarten via de standaardmonitor, dan is de I/O niet goed gedefinieerd. Vandaar.

B. Koude start

Startadres: $15ØØ

De koude start van PME is – net als bij de standaard-editor het geval is – bedoeld voor het prille begin van een te editen programma. Dus niet voor een gebruikersprogramma dat ooit al is ge-edit of misschien zelfs al geassembleerd en dat opnieuw moet worden ge-edit. Dáár hebben we andere startadressen voor. Zoals zal blijken.
Hoe gaan we te werk?
Eerst PM opstarten:
RST 1 Ø Ø Ø GO RUB OUT (= RES)
We zitten nu in PM. Nu de koude start:
1 5 Ø Ø SP R (SP: spatietoets; **niet**: "S", gevolgd door "P"!)
Nu is PME opgestart. Er volgt de terugmelding:
BEGAD, ENDAD:
Drie keer raden wat u nu moet doen? . . . Inderdaad, heel goed: Het eerste adres BEGAD en het laatste adres ENDAD opgeven van het stuk(je)

geheugen waarin het programma moet worden gezet, te beginnen vanaf BEGAD. Bij een koude start moet het altijd gaan om RAM-geheugen.
Hoe gaan we te werk? Net als bij de toets M van PM (hex dump). Eerst geven we BEGAD op, vervolgens drukken we de kommatoets in en dan geven we ENDAD op. Rest nog het indrukken van de toets CR. Linker nullen hoeven we niet op te geven. Een voorbeeld. We gaan uit van het buitengewoon populaire geheugenbereik Ø2ØØ . . . Ø3FF:

BEGAD, ENDAD: 2ØØ, 3FFCR
PM EDITOR
Ø2ØØ 77

De terugmelding "PM EDITOR" betekent dat we kunnen beginnen met de opgave van labels en instrukties. Die "77" op BEGAD is dezelfde 77 als die van de standaard-editor: het EOF-teken, dat naarmate er instrukties en labels worden opgegeven doorschuift naar een hoger adres. Het is dan ook niet verwonderlijk dat de eerste instruktie of label die na een koude start wordt opgegeven, moet worden opgegeven met de INSERT-toets I (zie punt G, nummer ⑤).
Al met al gaat een koude start nu heel wat sneller in zijn werk dan "vroeger", met de standaard-editor. Geen gedoe meer met ENDADL=$ØØE2, enzovoorts. Prima!
N.B. Indien men een ENDAD lager dan BEGAD opgeeft treedt er geen foutmelding op. Dit in tegenstelling tot de M-toets van PM. Het spreekt vanzelf dat ENDAD kwa adres hoger moet zijn dan BEGAD en dus kwa positie op de "memory map" lager.

C. Warme start

Startadres: $1533

De warme start kennen we ook nog van vroeger. Deze is bedoeld om terug te keren naar de editer, waarbij de inhoud van de nu volgende adreswijzers wordt bepaald door de voorgeschiedenis:
BEGAD: beginadreswijzer
ENDAD: eindadreswijzer
CEND : variabele eindadreswijzer
CURAD: displaywijzer. Wees vroeger op het adres met de opcode van de instruktie of het label dat op het display zichtbaar was. Wijst nu, bij PME, op het adres met de opcode van de instruktie die als laatste door PME op de linker kolom van het tv-scherm of printerpapier is afgedrukt.

Na de opgave van het startadres 1533 (via PM!) en het indrukken van toets R volgt via PME de terugmelding:

PM EDITOR
XXXX YY YY YY

waarbij XXXX de inhoud van CURAD is en de Y-tjes de instruktie of het label voorstellen waarvan de opcode of FF op het adres CURAD is gehuisvest. Afhankelijk van de lengte van de instruktie dient men geen, de laatste twee danwel de laatste vier Y-tjes als niet afgedrukt op te vatten. In tegenstelling tot de koude start (punt B) wordt er dus nu geen 77 op het adres BEGAD neergezet.

D. Lauwe start

Startadres: $1667

Dit is helemaal nieuw. Dat kennen we nog niet van vroeger. Lauw betekent iets dat tussen warm en koud inzit. De overeenkomst met de koude start is dat eerst BEGAD en ENDAD moeten worden opgegeven en dat de eerste instruktie (die met de opcode op BEGAD) wordt afgedrukt. De overeenkomst met de warme start is dat er op het adres BEGAD geen 77 wordt geschreven. Bij de koude start is de variabele eindadreswijzer CEND "in den beginne" gericht op een adres, één hoger dan BEGAD. Bij de lauwe start wordt die CEND gelijk gemaakt aan ENDAD.
De startprocedure:
1 6 6 7 SP R (SP: spatietoets; **niet**: "S", gevolgd door "P"!)
BEGAD, ENDAD: 1CØØ, 1FFF CR
PM EDITOR
1CØØ 85
In het voorbeeld is een BEGAD 1CØØ en een ENDAD 1FFF opgegeven. Hé, maar is dat geen EPROM-geheugenbereik? Klopt. De lauwe start is met name bedoeld voor het bekijken van kant en klare programma's. Hetzij in RAM, hetzij in EPROM. Vooral de "bekijk-toetsfunkties" SP (zie punt G, nummer ①), Z (zie punt G, nummer ②), L (zie punt G, nummer ⑧), P (zie punt G, nummer ⑨) en S (zie punt G, nummers ⑥ en ⑦) zijn interessant bij een lauwe start. Deze startingang van PME heeft dus een grote betekenis bij het onderzoek (bijvoorbeeld om edukatieve redenen) of de dokumentatie (bijvoorbeeld een instruktielisting) van een al dan niet door u gemaakt programma, met inbegrip van systeemprogramma's in EPROM.

E. Warme CEND-start

Startadres: $17C5

In hoofdstuk 11 van boek 3 hebben we het gehad over het op de band zetten van een gebruikersprogramma-mèt-labels, en over de voordelen daarvan. Dat kan via het programma TM (toets SEF), maar ook via PM. In het laatste geval moet men behalve een ID een SA opgeven die gelijk is aan BEGAD en een EA die gelijk is aan CEND. Die CEND, daar komt men met PME snel achter, omdat deze gelijk is aan het adres dat één hoger is dan het adres waarop de rode lantarendrager 77 staat.
Okay, er staat een programma op de band waaraan we via PME willen sleutelen. We moeten dat programma eerst van de band teruglezen. Dat doen we via PM (toets G). Vervolgens moeten we PME warm starten. Dat betekent dat BEGAD, ENDAD en CEND elk een waarde moeten hebben die in overeenstemming moet zijn met het programma dat zojuist van de band is opgehaald. Verder moet CURAD een zinvolle waarde hebben. In hoofdstuk 11 is beschreven hoe dat allemaal moet. Je moest allerlei adressen noteren en zo. Niet zo handig, eerlijk gezegd.
Gebruiken we de warme CEND-start van PME, dan behoort ook dat noteren voor een groot deel tot het verleden. Uiteraard moeten de ID en BEGAD wèl bekend zijn; ENDAD kan men opnieuw definiëren (bijvoorbeeld hoger kiezen omdat er instrukties en labels bijkomen). Dat laatste kan bijvoorbeeld nodig zijn als men via de "ID-FF-truuk" meerdere

ge-edite programma's inleest die naadloos op elkaar aansluiten (tussenliggende 77's worden verwijderd).
Wat gebeurt er allemaal na de warme CEND-start van PME? Er gebeurt dit: na de opgave van BEGAD en ENDAD (gaat net zo als bij de koude en de lauwe start) gaat PME op zoek naar de geheugenplaats waarin de pseudo-opcode 77 staat. Zodra die is gevonden wordt het adres waarop 77 staat met één verhoogd en wordt CEND gelijk gemaakt aan dit verhoogde adres. Er volgt de terugmelding:

PM EDITOR
XXXX 77

en CEND staat gericht op het adres (XXXX+1).
De warme CEND-startingang kan in ieder geval worden gebruikt voor het opnieuw editen van programma's-mèt-labels die indertijd op de band zijn geschreven met SA=BEGAD en EA=CEND. Daar is die startingang primair voor bedoeld. Er zijn echter nog meer mogelijkheden. Die berusten op de vaststelling dat **na het assembleren van een programma het EOF-teken 77 niet verloren is gegaan.** Na het assembleren, dus na het verwijderen van de labels, is die 77 nog steeds rode lantarendrager: hij zit in de geheugenplaats die direkt volgt op de geheugenplaats(en) met de laatste instruktie van het programma. Indien we, om welke reden dan ook, een al geassembleerd programma later opnieuw willen editen (verwijderde labels kun je alsnog weer toevoegen!), kunnen we gebruik maken van de warme CEND-ingang van PME, mits we indertijd, bij het op de band zetten van dat programma óók de afsluitende 77 hebben meegenomen. Dat is heel eenvoudig een kwestie van EA één hoger kiezen dan een EA-waarde die hoort bij de laatste instruktie van het programma. Bij gebruik van de toets SEF van TM zit dat automatisch goed. Want bij gebruik van deze ingang van PME is het van essentieel belang dat een 77 wordt aangetroffen. Is dat niet het geval, dan volgt geen terugmelding "PM EDITOR" en gaat PME de mist in! Mocht dit optreden, dan moet men de toets RST indrukken, PM opstarten en vervolgens een andere startingang van PME kiezen.
Voor de praktijk van deze startingang: zie praktijkvoorbeeld 5, verderop in dit hoofdstuk.

F. De BRK-toets

Het onderbreken van het afdrukk. . .

Van het programma PM herinneren we ons de BRK-toets. Die werd gebruikt om het afdrukken van langere tekst af te breken. Een soort grafische noodrem dus. Iets dergelijks doen we hier, bij PME, ook. Was het bij PM vooral de uitvoering van de M-toetsfunktie (hex dump) die nog wel eens tot het onderbreken met de BRK-toets aanleiding geeft, bij PME is dat vooral de uitvoering van de L-toetsfunktie (zie punt G, nummer ⑧). Men kan via het indrukken van toets L met zevenmijlslaarzen snel het programma bekijken. Is men ongeveer aangeland bij het gedeelte van het programma dat men wat rustiger en meer in detail wil bekijken, dan drukt men op de BRK-toets en vervolgens één of meer keren op de P-toets (zie punt G, nummer ⑨).
Na het indrukken van de BRK-toets volgt de terugmelding, bij monde van PME, van "PM EDITOR".

N.B. Indien men PME start via de koude, lauwe of warme CEND-start-ingang, dan is in de beginfase de BRK-sprongvektor nog gespecificeerd volgens PM. Onderbreekt men de melding "BEGAD, ENDAD:" door het indrukken van de BRK-toets, dan volgt de terugmelding "JUNIOR".

G. De toetsfunkties van PME

① Toets SP (spatie) (Skip)

Instruktie-plustoets

Dit is een oude bekende van de standaard-editor. We gaan ervan uit dat PME op de linker kolom van het tv-scherm of printerpapier een instruktie heeft afgedrukt. Bij PME mèt het adres met de opcode van die instruktie. De schrijfwijzer van de elekterminal of de wagenpositie van de printer staat "gericht" op de eerste positie van de rechter kolom van het tv-scherm of printerpapier; en wel op dezelfde regel als de laatste instruktie waar we het zonet over hadden. Drukken we nu de spatiebalk SP (space) in (niet te verwarren met toets S, gevolgd dooe toets P!), dan zien we niets verschijnen op die rechter kolom, die was en is bedoeld voor de weergave van onze toetsakties. Maar dat ligt uitsluitend aan het feit dat het een spatietoets was die ingedrukt werd!

Dat neemt niet weg dat het indrukken van die spatietoets wel degelijk iets teweeg brengt. Op de volgende regel en op de linker kolom wordt namelijk de volgende instruktie van het programma afgedrukt, of FF XX ØØ, als de volgende "instruktie" een label is. Aan het adres van de nieuwe instruktie of label kan men de instruktielengte van de vorige instruktie afmeten. Dat adres is één, twee of drie hoger dan het adres van de vorige instruktie.

Indien alle instrukties van het programma zijn doorlopen, of juister gezegd: indien door herhaald indrukken van SP de laatste "instruktie" 77 is afgedrukt, volgt na het vervolgens nògmaals indrukken van SP het volgende:

XXXX 77 SP
DONE
YYYY ZZ (ZZ(ZZ)

Voordat de nieuwe instruktie wordt afgedrukt volgt dus de terugmelding "DONE", ten teken van het feit dat de variabele eindadreswijzer CEND is gepasseerd. Het adres YYYY is één hoger dan het adres XXXX omdat aan de "instruktie" met de opcode 77 de instruktielengte één wordt toegekend. De instruktie "ZZ ZZ ZZ" is hoogstwaarschijnlijk een fantasie-instruktie. Dat hangt af van de voorgeschiedenis: tijdens een computersessie kan men voor de tweede keer de editor koud hebben gestart. In ieder geval hoort die "ZZ ZZ ZZ" niet bij het programma waarmee men bezig is. Het aantal Z-en hangt af van de instruktielengte van de al dan niet echte instruktie. Uit de bespreking van de subroutine OPLEN/LENACC (hoofdstuk 8 van boek 2) weten we dat aan elke kombinatie XX van twee datanibbles een opcode wordt toegekend.

N.B. Indien men na het skippen van de instruktie 77 méér dan één keer SP indrukt wordt telkens de volgende "instruktie" afgedrukt na de melding "DONE". Het programma PME zorgt dus niet voor een blokkade ("tot zover en niet verder"), maar geeft met de meldingen "DONE" wèl aan dat het dóórgaan met "skippen" niet zo erg zinvol is.

② Toets Z (Back)

Instruktie-mintoets

NIEUW!

Een verbetering van PM ten opzichte van de standaardmonitor was de mintoets naast de plustoets. Ten opzichte van de standaard-editor heeft PME er naast de skiptoets (instruktie-plustoets) een instruktie-mintoets bijgekregen.
Dat gaat zo:

Ø2AC FF 17 ØØ Z (label 17)
Ø2AB CA Z (DEX)
Ø2A9 A9 FF Z (LDA # FF)
Ø2A6 2Ø 14 ØØ (enzovoorts) (JSR-label 14)

Indien men zó vaak de Z-toets indrukt dat de instruktie of het label op het adres BEGAD is bereikt zal het opnieuw indrukken van de Z-toets opnieuw het afdrukken van de eerste instruktie of het eerste label tot gevolg hebben. Lager dan BEGAD (kwa adres) gaat gewoon niet.

③ Toets K (Kill) (Delete)

Instruktie of label verwijderen

Dit is een oude bekende. Het indrukken van toets K heeft tot gevolg dat de als laatste afgedrukte instruktie uit het geheugen wordt verwijderd, of juister: wordt overschreven door de volgende instruktie(s). Het programma wordt dus, afhankelijk van de lengte van de verwijderde instruktie, met één, twee of drie bytes ingekort. De instruktie die in het geheugen volgt op de verwijderde instruktie wordt afgedrukt. Werd de laatste instruktie verwijderd, dan ziet men het adres CEND min één met de daarbij horende data 77 afgedrukt.
Een voorbeeld:

Ø2A6 2Ø 14 ØØ SP
Ø2A9 A9 FF K
Ø2A9 CA SP
Ø2AA FF 17 ØØ SP
Ø2AD 77 SP
DONE
Ø2AE XX XX XX

We zien dat de instrukties CA en FF 17 ØØ twee plaatsjes lager (kwa adres) in het geheugen staan.

④ Toets T (Top of file)

Terug naar het begin

NIEUW!

Niet zo'n spektakulaire nieuwe toetsfunktie, die T-toets, maar toch reuze handig. Het indrukken van de T-toets zorgt ervoor dat de instruktie of het label op het eerste adres BEGAD van het programma wordt afgedrukt. Dat is bijvoorbeeld van belang als men het programma wil kontroleren met toets SP, toets L of toets P.

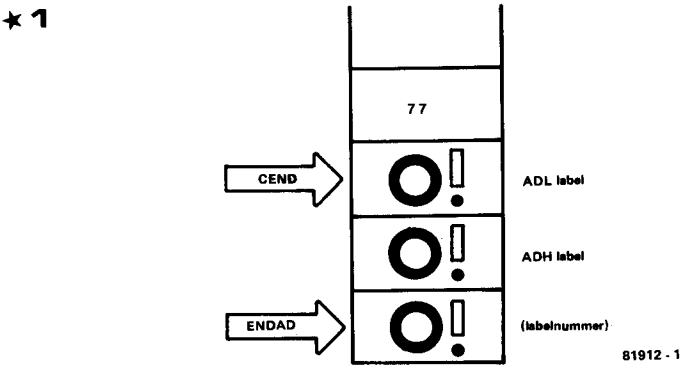
Een voorbeeld:
Ø2AE XX XX XX T
Ø2ØØ FF 1Ø ØØ
Waaruit blijkt dat BEGAD in dit programmavoorbeeld gelijk is aan Ø2ØØ en dat het programma begint met label 1Ø. Bij de oude editor kon men BEGAD bereiken via Search FF 1Ø of door te skippen. Bij PME is er nog een alternatief in de vorm van "SFF 1Ø ØØ" (zie punt G, nummers ⑥ en ⑦), maar die ene T indrukken betekent toch beduidend minder werk. En op het punt van dom toetsenwerk luidt ons devies, zoals bekend: hoe minder werk, hoe liever.

⑤ Toets I (Insert)

Instruktie of label tussenvoegen

Deze toetsfunktie is niet nieuw; ook de standaard-editor is er mee uitgerust. Het indrukken van toets I plus de daarop volgende ingetoetste numerieke data (die hoort bij de tussen te voegen instruktie) heeft tot gevolg dat de zojuist opgegeven instruktie in het werkgeheugen wordt gezet op een plaats of plaatsen direkt vóór de plaats of plaatsen die bezet zijn door de als laatste afgedrukte instruktie. Dat gebeurt echter pas **nadat een aantal numerieke toetsen is ingedrukt** dat **overeenkomt met de lengte van de tussen te voegen instruktie.** De als laatste afgedrukte instruktie en de daarop volgende instrukties schuiven in het geheugen op naar een hoger adres.
Indien men na het indrukken van toets I en vóór de volledige opgave van de instruktie (twee, vier of zes numerieke toetsen indrukken) een andere dan één van de toetsen Ø . . . 9 en A . . . F indrukt volgt de **foutmelding:**
ILLEGAL KEY
en wordt opnieuw de instruktie afgedrukt die we "laatste instruktie" hebben genoemd. Men kan per ongeluk na I een niet-numerieke toets indrukken. Maar het kan ook bewust gebeuren. Bijvoorbeeld omdat men meteen al in de gaten heeft dat er foutieve data is opgegeven. Na de terugmelding "ILLEGAL KEY" kan men dan gewoon opnieuw beginnen. Wèl dan opnieuw beginnen met het indrukken van toets I, want het ging om de Insert-toetsfunktie en niet om de Input-funktie.
Ook nu geven we een klein voorbeeldje. Gaan we uit van de situatie, direkt na het verwijderen van de instruktie A9 FF, zie punt G, nummer ③. Op adres Ø2A9 komt dan de instruktie CA te staan. We willen de instruktie A9 FF vervangen door A9 ØØ. Daartoe verwijderen we A9 FF. Dàt is gebeurd via het indrukken van de K-toets. Nu drukken we de toetsen I, A, 9, Ø en Ø en gebeurt er het volgende:
Ø2A9 CA IA9 ØØ
Ø2A9 A9 ØØ SP
Ø2AB CA SP
Ø2AC FF 17 ØØ
U ziet dat alle instrukties na de LDA-immediate weer twee plaatsen zijn opgeschoven, dus naar een adres dat twee hoger is. Trouwens: is het u óók opgevallen dat PME ervoor zorgt dat de bytes van een instruktie door een spatie gescheiden worden weergegeven? En dat niet alleen op de linker "reaktiekolom" maar ook op de voor de gebruiker bestemde rechter

"aktiekolom"? En waarom? Omdat het zo een stuk overzichtelijker wordt. Samen met de Input-toetsfunktie (zie punt G, nummer ⑩) is de Insert-toetsfunktie er voor de opgave van instrukties en labels. In beide gevallen wordt de geheugenruimte die het programma in beslag neemt groter. Die geheugenruimte bestaat uit de adressen BEGAD tot en met het adres waarop zich de hekkesluiter 77 bevindt. Dus van BEGAD tot CEND.
Nu is de voor het programma gereserveerde geheugenruimte – inklusief de labels! – niet oneindig groot. We hebben immers een ENDAD opgegeven. Dat is niet voor niets gebeurd. Bij de standaard-editor speelt die ENDAD overigens geen enkele rol, bij de assembler daarentegen wèl. Evenals bij PME. Want we gaan het nu hebben over een foutmelding die wordt afgedrukt indien het beschikbare (lees: het opgegeven) geheugenbereik dreigt te worden overschreden. We gaan het dus hebben over iets dat nieuw is ten opzichte van de standaard-editor.
Zoals zal blijken en trouwens al eerder is opgemerkt gebruiken we de assembler van de hoofdstukken 5 en 9 van boek 2 óók voor programma's die via PME zijn opgegeven. Tijdens de eerste fase van het assembleren wordt elk gevonden label (opcode FF) opgeslagen in de label-geheugenruimte en vervolgens uit het programma verwijderd: het programma wordt met drie bytes ingekort. Vóórdat het eerste label kan worden verwijderd moet het worden weggeschreven. Is er plaats voor dàt label, dan is er ook automatisch plaats voor de andere labels van het programma.
Het eerste label moeten we dus kwijt kunnen op de geheugenplaatsen ENDAD en de twee daaronder gelegen geheugenplaatsen (lager kwa adres; hoger op de "memory map"). Deze geheugenplaatsen mogen dus niet worden gebruikt voor instrukties of labels van het te editen programma, en ook niet voor het EOF-teken 77. Dit betekent dat de stand van CEND zoals in figuur 1 aangegeven de laagst mogelijke stand is; CEND staat dan op het hoogst mogelijke adres gericht, heeft dan de maximale inhoud.
(Tussenopmerking. De op pagina 169 en pagina 14 van de eerste druk van boek 2 genoemde minimale aantallen vrije geheugenplaatsen zijn dus te groot. Men moet drie plaatsen reserveren voor het eerste label en één plaats voor de 77. Het adres ENDAD telt mee bij de bepaling van het totale aantal geheugenplaatsen).
Terug naar de kontrole of er nog plaats is. Zodra een instruktie of label is opgegeven – via een Insert of via een Input – bekijkt PME of de daarmee gepaard gaande verhoging van CEND (het programma wordt immers langer) niet een nieuwe CEND tot gevolg heeft die kwa adres hoger is dan de CEND van figuur 1. Is dat wèl het geval, dan volgt de terugmelding "FULL". De als laatste opgegeven instruktie wordt niet in het geheugen gezet. Daar is immers geen plaats meer voor. Wèl wordt er na de terugmelding "FULL" een instruktie afgedrukt. In het geval van de Insert is dat de instruktie die als laatste werd afgedrukt voordat de nieuwe instruktie via toets I werd opgegeven – en "geweigerd". Met andere woorden: de "displaywijzer" CURAD is ongewijzigd gebleven. In het geval van een opgegeven en geweigerde instruktie via Input wordt CURAD wèl verhoogd met de instruktielengte van de laatste afgedrukte instruktie. Voor het verschil tussen Input en Insert: zie ook tabel 2.
N.B. Uit figuur 1 kan men opmaken dat de laatste geheugenplaats die beschikbaar is voor het te editen programma, een adres ENDAD-4 heeft.



Figuur 1. De "onderste" vier geheugenplaatsen (met de hoogste vier adressen) van het door BEGAD tot en met ENDAD vastgelegde geheugenbereik kunnen niet worden gebruikt voor het te editen programma. Zodra de inhoud van CEND kleiner is dan de inhoud van ENDAD minus twee zorgt PME voor de foutmelding "FULL". Het verkeersbord in de onderste drie geheugenplaatsen betreft uitsluitend de situatie tijdens het editen. Na de eerste fase van het assembleren komt het eerstgevonden label op deze geheugenplaatsen te staan (achtereenvolgens labelnummer, ADH en ADL).

Het kan zijn dat er noodgedwongen nog meer geheugenplaatsen ongebruikt blijven dan de vier van figuur 1 (de geheugenplaats met 77, alsmede geheugenplaats ENDAD inbegrepen). Bijvoorbeeld in het geval dat geheugenplaats ENDAD-4 weliswaar nog vrij is (alle plaatsen van BEGAD tot en met ENDAD-3 zijn dan al met instrukties en labels bezet), maar dat men vervolgens een twee- of driebyte-instruktie wil opgeven.
Nòg een N.B. Nadat een opgegeven instruktie wegens plaatsgebrek is geweigerd is de variabele eindadreswijzer CEND verhoogd in overeenstemming met de lengte van de geweigerde instruktie. Het EOF-teken 77 blijft echter op zijn plaats!

⑥ Toets S (Search)

⑦ Toets Y (Yes)

De Search-toetsfunktie kennen we al van de "oude" editor. Daar moest men twee bytes opgeven. Dus vier numerieke toetsen indrukken. Die twee bytes konden een twee-byte-instruktie voorstellen. Of de eerste twee bytes van een drie-byte-instruktie of van een label. Op zoek naar dat twee-byte-patroon, vanaf BEGAD, werd er gestopt zodra dat patroon werd gevonden. Men kon niet nagaan of dat patroon misschien nòg wel een keer voorkwam. Denk aan instrukties zoals A9 ØØ. Eén-byte-instrukties kon je helemáál niet opsporen en bij drie-byte-instrukties kon het vriezen of dooien.
Dat gaat nu wel even anders.
De Search-toetsfunktie van PME heeft twee belangrijke eigenschappen, die beide nieuw zijn ten opzichte van de standaard-editor.

Ten eerste: We zoeken niet een twee-byte-patroon op, maar een instruktie, die één, twee of drie bytes lang mag zijn. Men kan dus letterlijk alles opsporen wat men wil.
Ten tweede: Indien men wil nagaan of een gezochte – èn gevonden – instruktie verderop in het door BEGAD en CEND bepaalde geheugenbereik nogmaals voorkomt, dan is dat mogelijk. We kunnen dus niet alleen alle verschillende instrukties en labels opsporen maar ook alle adressen waarop onderling identieke instrukties staan (het is voor u te hopen dat de labels elk maar één keer voorkomen!).

Hoe gaan we in de praktijk te werk?

1. Eerst drukken we toets S in. Die wordt afgedrukt op de rechter kolom.
2. Vervolgens geven we de op te sporen instruktie op. Dat gebeurt door het indrukken van twee, vier of zes numerieke toetsen. Het aantal is in overeenstemming met de lengte van de op te sporen instruktie of het op te sporen label. Pas nadat de instruktie volledig is opgegeven gaat PME op zoek naar die instruktie. De opgegeven instruktie wordt al tijdens de opgave naast de S op de rechter kolom afgedrukt; PME zorgt voor spaties tussen de bytes.

Drukt men na de S-toets per ongeluk of bewust een niet-numerieke toets in, dan volgt de terugmelding:

ILLEGAL KEY

en moet men opnieuw gaan "searchen", te beginnen met het indrukken van S. Men kan bewust een niet-numerieke toets indrukken als men in de gaten krijgt dat men een verkeerde instruktie aan het opgeven is. Ook nu wordt de opgegeven instruktie afgedrukt met een spatie tussen de bytes van de instruktie.

3. De instruktie is opgegeven en PME gaat ernaar op zoek. Beginnend vanaf het laagste adres BEGAD. Er zijn nu twee mogelijkheden: De gezochte instruktie is aanwezig in het door BEGAD en CEND gespecificeerde geheugengedeelte; PME reageert door op de volgende regel en op de linker kolom de instruktie af te drukken, mèt adres. Is daarentegen de gezochte instruktie of het gezochte label niet aanwezig in dat geheugengedeelte, dan volgt de terugmelding:

DONE
XXXX ZZ ZZ ZZ

Waarbij XXXX het adres is dat wordt bereikt na het overschrijden van CEND en ZZ ZZ ZZ de instruktie is waarvan de opcode op adres XXXX staat (deze hoeft niet per se drie bytes lang te zijn, zoals de zes Z-en suggereren).

4. De nu volgende toetsakties zijn uitsluitend van belang indien er, tijdens punt 3, een instruktie is gevonden. Men kan nu twee verschillende wegen inslaan:

4a. Men wil nagaan of de instruktie vaker voorkomt in het onderhavige geheugengedeelte. **Men drukt daartoe toets Y in.** Die Y zien we op de rechter kolom verschijnen. Nu kan PME zich op twee manieren terugmelden (op de linker kolom). Indien de gezochte instruktie nòg minstens één keer voorkomt wordt deze afgedrukt, mèt het bijbehorende adres. Dat adres verschilt dus van het adres van de "eerste vondst" (zie punt 3); dat adres is hoger dan het eerste adres. Het kan ook voorkomen dat de gezochte instruktie slechts één keer voorkomt. In dat geval volgt de

terugmelding:
DONE
XXXX ZZ ZZ ZZ
(zie punt 3).

4b. Men wil **niet** nagaan of de instruktie vaker voorkomt in het onderhavige geheugengedeelte. **Men drukt daartoe een willekeurige toets van het ASCII-toetsenbord in die bij indrukken een ASCII-kode genereert, echter met uitzondering van de toets Y.** Er volgt dan de terugmelding:
DONE
XXXX ZZ ZZ ZZ
waarbij ZZ ZZ ZZ de in punt 3 gevonden instruktie is en XXXX het bijbehorende adres.

Over de praktijk van het "instruktietje zoeken" moeten we nog een paar belangrijke opmerkingen kwijt:

a. Men kan het zoeken naar een instruktie op een willekeurig moment afbreken. Aan de instruktie en het adres die na de terugmelding "DONE" kan men zien of er niet verder gezocht is danwel dat er verder niets is gevonden.

b. Een instruktie-zoekaktie (lees: de Search-toetsroutine) is pas dan volledig afgesloten indien de terugmelding "DONE" plus de afdruk van een instruktie heeft plaatsgevonden. Daarbij is het niet van belang of die "DONE" tot stand komt omdat er (verder) niets is gevonden of omdat we verder niets willen zoeken (via het indrukken van een willekeurige toets ≠ de Y-toets).

c. De Y-toets heeft uitsluitend betekenis nadat toets S is ingedrukt en nadat de op te zoeken instruktie is opgegeven. Heeft men niet gekozen voor de S-toetsfunktie en drukt men de Y-toets in, of doet men dat terwijl de op te zoeken instruktie nog niet volledig is opgegeven, dan volgt de terugmelding "ILLEGAL KEY".

d. Tijdens het afbreken van een zoekaktie hebben alle software-toetsen op Y na dezelfde betekenis. Er volgt dan geen terugmelding "ILLEGAL KEY". De oorspronkelijke funktie van de toetsen van PME – óók S, maar Y uitgezonderd – is tijdens het opzoeken, dus na het indrukken van toets S, maar vóór een afsluitende terugmelding "DONE", niet van toepassing. Andere toetsen, die normaal gesproken geen funktie hebben en dus aanleiding geven tot de terugmelding "ILLEGAL KEY", hebben na het indrukken van S en nadat een instruktie minstens één keer is gevonden, daarentegen tijdelijk wèl een funktie.

Een en ander zal u helemáál duidelijk worden in hoofdstuk 15, bij de bespreking van de PME-software.

De hoogste tijd voor een praktijkvoorbeeld.

We doen dat aan de hand van de listing op printerpapier van tabel 1.

U ziet in tabel 1 dat we zijn begonnen met een lauwe start van PME. De adreswijzers BEGAD en ENDAD zijn zodanig gekozen dat de standaard-EPROM, op de standaardkaart van de junior-computer wordt bestreken. Na het indrukken van CR (is niet zichtbaar in tabel 1) volgt netjes de terugmelding "PM EDITOR" en wordt de eerste instruktie "1CØØ 85 F3" afgedrukt.

We willen nagaan hoe het zit met het voorkomen van de instruktie STAZ-POINTL en toetsen S85FA. De spatie tussen 85 en FA wordt door PME

Tabel 1. Een voorbeeld van het gebruik van de lauwe startingang van PME en van het gebruik van ondermeer de Searchtoets S.

```
JUNIOR                    ①

1667
1667 20 R
BEGAD,ENDAD: 1C00,1FFF
PM EDITOR
1C00 85 F3          S85 FA
1C08 85 FA          Y
1C83 85 FA          Y
1CB0 85 FA          Y
1CDD 85 FA          Y
1D3D 85 FA          Y
1E37 85 FA          Y
1FDA 85 FA          Y
1FEA 85 FA          Y
DONE
2000 0A             SC8
1CBF C8             Y
1CEA C8             Y
1D55 C8             Y
1E0E C8             Y
1E56 C8             Y
1F70 C8             Y
1FB6 C8             Y
1FBF C8             Y
1FC4 C8             Y
DONE
2000 0A             T
1C00 85 F3          P
1C02 68
1C03 85 F1
1C05 68
1C06 85 EF
1C08 85 FA
1C0A 68
1C0B 85 F0
1C0D 85 FB
1C0F 84 F4
1C11 86 F5          →②
```

```
1C13 BA                    ②
1C14 86 F2
1C16 A2 Ø1
1C18 86 FF
1C1A 4C 33 1C        S8D 83 1A
1C1F 8D 83 1A        Y
DONE
2ØØØ ØA              S8E 83 1A
DONE
2ØØØ ØA              S8C 83 1A
DONE
2ØØØ ØA
```

afgedrukt. Dat hoeft u niet te doen. Sterker zelfs: màg u niet doen, want dan volgt de terugmelding "ILLEGAL KEY" en dan kunt u opnieuw beginnen.
Nou, u weet dat de gezochte instruktie voorkomt, want geheugenplaats ØØFA is de veel gebruikte displaybuffer POINTL. Het is dan ook meteen raak: op adres 1CØ8 komen we hem voor het eerst tegen. Het vervolgens acht keer indrukken van toets Y levert nog zes andere plaatsen op waar de instruktie zich bevindt. Merk op dat de diverse vondsten een toenemend adres hebben. Na de achtste Y zijn alle instrukties gevonden; er volgt de terugmelding DONE en er wordt vervolgens een instruktie op adres 2ØØØ afgedrukt. Dat laatste is niet verwonderlijk als men bedenkt dat bij de lauwe start CEND gelijk wordt gemaakt aan ENDAD. En die is 1FFF.
Vervolgens gaan we in tabel 1 op zoek naar de instruktie C8. Dat is INY. Die blijkt maar liefst negen keer voor te komen. U kunt overigens de adressen met de gevonden instrukties STAZ-POINTL respektievelijk INY kontroleren aan de hand van de listing van de standaard-EPROM op de pagina's 180 . . . 189 van boek 2.
Gaan we verder met tabel 1. Na het indrukken van toets T wordt het adres 1CØØ afgedrukt, met de eerste instruktie. En zo hóórt het ook. Over de uitwerking van het indrukken van toets P komen we nog te spreken (zie G, punt ⑨).
Vervolgens en tot slot van tabel 1 doen we een edukatief onderzoekje. We willen weten waar en hoe in de standaard-EPROM het richtingsregister PBDD wordt beïnvloed. Het adres van PBDD is $1A83. Dit register moet worden (zijn) beïnvloed door een store-instruktie. Er zijn drie mogelijkheden:
8D 83 1A STA-$1A83
8E 83 1A STX-$1A83
8C 83 1A STY-$1A83
Eerst gaan we na of er een STA-$1A83 voorkomt. Dat blijkt één keer het geval te zijn. De zoekakties naar STX-$1A83 en STY-$1A83 hebben een negatief resultaat. Belangrijke edukatieve konklusie: de enige plaats waar PBDD wordt beïnvloed is een plaats (lees: adres) die deel uitmaakt van de RESET-opstartroutine van de standaard-monitor.
We gaan verder met de volgende toetsfunktie van PME.

⑧ Toets L (List)

Het programma afdrukken

Ook deze is nieuw!

Van PM hebben we allerlei afdrukkommando's leren kennen. Denk aan de hex dump. Tot nog toe ging het bij PME om toetsfunkties die aanleiding geven tot het afdrukken van één instruktie, eventueel gekombineerd met een terugmelding. De toetsfunktie L en toetsfunktie P (zie G, punt ⑨) zorgen voor het achterelkaar afdrukken van meerdere instrukties.
Het indrukken van toets L heeft tot gevolg dat – te beginnen met de eerste instruktie of het eerste label op BEGAD – alle instrukties en labels van het door BEGAD en CEND bepaalde geheugengedeelte worden afgedrukt. En wel onder elkaar, op de linker kolom van het tv-scherm of printerpapier. Zodra alle instrukties zijn afgedrukt (de displaywijzer CURAD verplaatst zich naar een hoger adres in het geheugengedeelte na het afdrukken van een instruktie) volgt de terugmelding "DONE" en wordt er vervolgens een instruktie afgedrukt die in overeenstemming is met de waarde van CURAD na het "passeren" van CEND door CURAD. De toetsfunktie L is met name bedoeld voor het snel door een programma "fietsen". Wil men een bepaald gedeelte rustig kunnen bekijken dan drukt men op de BRK-toets (terugmelding "PM EDITOR") en gaat verder via het minstens één keer indrukken van de P-toets, aan de bespreking waarvan we nu toe zijn.

⑨ Toets P (Print)

Instruktieblokken afdrukken

Het gaat om een variant op de L-toetsfunktie. We gaan uit van een laatste, door PME afgedrukte instruktie. Hoe dat is gebeurd doet er niet toe. Dat hangt af van de voorgeschiedenis. Die laatste instruktie staat op de linker kolom. We drukken nu op de P-toets. Op dezelfde regel als de regel van de laatste instruktie, maar nu op de rechter kolom wordt de ingedrukte P als P afgedrukt. En vervolgens worden de daarop volgende 15 instrukties onder elkaar en op de linker kolom afgedrukt. In totaal worden er dus, de afgedrukte instruktie met op dezelfde regel de P meegerekend, zestien instrukties afgedrukt. In het geval van de Elekterminal is dat precies een heel tv-scherm vol.
We zeggen nou wel: zestien instrukties afgedrukt, oftewel een instruktieblok, maar dat is alleen dan het geval als er geen "opcode" 77 wordt vastgesteld. Zodra dit EOF-teken wordt vastgesteld wordt het afdrukken van het instruktieblok onderbroken. De "instruktie" 77 wordt als laatste afgedrukt.
N.B. Een praktijkvoorbeeld van toets P treft men aan in tabel 1.

⑩ Input-toetsfunktie

Toetsen Ø . . . 9 en A . . . F!

Het is al eerder in dit verhaal opgemerkt: Voor het realiseren van de INPUT-toetsfunktie is het niet nodig om, voordat de numerieke data wordt opgegeven, de een of andere funktietoets in te drukken. De enige andere toetsfunkties waarvan de uitvoering berust op de opgave, door de

gebruiker, van numerieke data, zijn Insert (toets I) en Search (toets S). Voordat bij die toetsfunkties numerieke data kan worden opgegeven moet eerst de toets I respektievelijk S worden ingedrukt.

Indien PME na de uitvoering van een toetsfunktie wacht op een nieuwe ingedrukte toets (dit kan men zien aan de stand van de "cursor" van de Elekterminal of de wagenpositie van een printer; deze staat gericht op de eerste positie van de rechter kolom), en indien er vervolgens numerieke toetsen worden ingedrukt, dan is er automatisch sprake van de Input-toetsfunktie.
Wat was ook weer de Input-toetsfunktie? Deze toetsfunktie houdt in dat de door de gebruiker opgegeven instruktie of het door de gebruiker opgegeven label nadat deze volledig is opgegeven in het geheugen op een plaats of plaatsen terecht komt die direkt volgen op de plaats of plaatsen waar de tot dan toe als laatste (op de rechter kolom) afgedrukte instruktie is gehuisvest.
De praktijk van het "Inputten" vertoont overeenkomsten en verschillen met de praktijk van de Insert-toetsfunktie.
Ook nu gaat de instruktie pas het geheugen in nadat deze volledig is opgegeven. Ook nu geeft het per ongeluk of bewust indrukken van een niet-numerieke toets aanleiding tot de foutmelding "ILLEGAL KEY". In dat geval moeten we opnieuw beginnen met de opgave van de instruktie. Ook nu wordt de instruktie afgedrukt op de linkerkolom op de volgende regel, nadat deze volledig is opgegeven. Ook nu zorgt PME voor spaties tussen de bytes van de instruktie; zowel op de rechter als op de linker kolom. Dus, wat de rechter kolom betreft, niet zèlf een spatie opgeven! Want dan ("ILLEGAL KEY") kunt u opnieuw beginnen met die instruktie.
Omdat er na het uitvoeren van de Input-toetsfunktie een instruktie bijgekomen is in het geheugen verplaatsen de 77 en CEND zich naar een hoger adres. Hoeveel hoger, dat hangt af van de lengte van de zojuist opgegeven instruktie.
Omdat ook hier de door BEGAD en CEND afgebakende geheugenruimte in omvang toeneemt bestaat ook hier de kans dat het beschikbare geheugen na verloop van tijd vol dreigt te raken. We verwijzen u daarvoor naar figuur 1 en het verhaal dat daarover is verteld bij de behandeling van de Inserttoetsfunktie. Is er geen plaats meer voor de zojuist opgegeven instruktie, dan volgt de foutmelding:

FULL
XXXX ZZ ZZ ZZ

Waarbij het adres XXXX en de instruktie ZZ ZZ ZZ in overeenstemming zijn met de verhoging van CURAD als gevolg van de zojuist opgegeven, maar niet in het geheugen toegelaten instruktie. Dit in tegenstelling tot de gang van zaken bij de Insert-toetsfunktie, waar XXXX en ZZ ZZ ZZ horen bij de instruktie die als laatste is afgedrukt voordat de geweigerde instruktie werd opgegeven.
Ook nu wordt ondanks de weigering van de instruktie door het geheugen de variabele eindadreswijzer CEND verhoogd in overeenstemming met de geweigerde instruktie. Ook nu, bij Input, blijft het EOF-teken 77 op zijn plaats!
In de vorm van de twee listings van tabel 2 geven we een praktijkvoorbeeld

Tabel 2. De opgave van een fantasie-programma met uitsluitend één-byte-instrukties via de Input-toetsen 0 . . . 9 (linker helft; de eerste instruktie moet altijd via Insert worden opgegeven) en via de Insert-toets I (rechter helft). Let op het verschil in "volmelding" en op het verschil in volgorde waarin de instrukties in het geheugen staan.

```
JUNIOR          ⓐ Input

1500
1500 20 R
BEGAD,ENDAD: 200,20F
PM EDITOR
0200 77                 ICA
0200 CA                 E8
0201 E8                 C8
0202 C8                 EA
0203 EA                 48
0204 48                 08
0205 08                 68
0206 68                 28
0207 28                 88
0208 88                 0A
0209 0A                 18
020A 18                 D8
020B D8                 58
FULL
020C 77                 T
0200 CA                 P
0201 E8
0202 C8
0203 EA
0204 48
0205 08
0206 68
0207 28
0208 88
0209 0A
020A 18
020B D8
020C 77
```

```
JUNIOR          ⓑ Insert

1500
1500 20 R
BEGAD,ENDAD: 200,20F
PM EDITOR
0200 77                 ICA
0200 CA                 IE8
0200 E8                 IC8
0200 C8                 IEA
0200 EA                 I48
0200 48                 1U
ILLEGAL KEY
0200 48                 I08
0200 08                 I68
0200 68                 I28
0200 28                 I88
0200 88                 I0A
0200 0A                 I18
0200 18                 ID8
0200 D8                 I58
FULL
0200 D8                 T
0200 D8                 P
0201 18
0202 0A
0203 88
0204 28
0205 68
0206 08
0207 48
0208 EA
0209 C8
020A E8
020B CA
020C 77
```

waarin het verschil tussen Insert en Input duidelijk tot uiting komt. In beide gevallen definiëren we een geheugenbereik (BEGAD, ENDAD) dat 17 geheugenplaatsen omvat (ENDAD meegerekend!). In beide gevallen gaat het om een fantasie-programma dat uitsluitend uit één-byte-instrukties bestaat. In het linker geval van tabel 2 worden de instrukties – afgezien van de verplichte eerste Insert – opgegeven via de Input-toetsfunktie. De rechter listing van tabel 2 betreft de opgave van dezelfde één-byte-instrukties in dezelfde volgorde, maar dan uitsluitend via de Inserttoetsfunktie.
In beide gevallen komt op adres Ø2ØB de laatste instruktie terecht. In beide gevallen is er geen plaats meer voor de instruktie met de opcode 58 (CLI). Dat Ø2ØB de laatste beschikbare geheugenplaats is stemt overeen met de regel dat die laatste plaats een adres heeft dat gelijk is aan ENDAD min vier. Immers: $ØF - $ØB is gelijk aan 15 min 11 en dat is vier. Op adres Ø2ØC staat de 77 (dat blijkt ook uit de in tabel 2 via toets P afgedrukte instrukties) en de drie geheugenplaatsen Ø2ØD, Ø2ØE en Ø2ØF zijn beschikbaar voor het opbergen van het eerste label tijdens de eerste fase van het assembleren.
In tabel 2 zien we verder heel duidelijk het verschil tussen Insert en Input ten aanzien van de volgorde waarin de instrukties in het geheugen worden gezet. Oftewel het verschil tussen toevoegen en tussenvoegen. De volgorde is omgekeerd.
U ziet in tabel 2 ook heel goed het verschil in afgedrukte instruktie na de melding "FULL".
Merk verder op dat in de rechter helft van tabel 2 bewust de foutmelding "ILLEGAL KEY" is uitgelokt. Bij het opgeven van Ø8 werd per ongeluk "1" ingetoetst in plaats van "I". Het indrukken van toets U bracht snel uitkomst.

⑪ Toets X (EXecute)

Assembleren via een druk op de (X-)toets

Het programma PME werkt net als de standaard-editor op basis van hexadecimale labels. Dat betekent dat de assembler die we al in boek 2 hebben leren kennen ongewijzigd kan worden gebruikt om programma's die met PME in elkaar zijn gezet te assembleren. Dat wil dus zeggen dat de labels stuk voor stuk uit het programma worden gehaald nadat de adresinformatie en het labelnummer zijn genoteerd in de labelgeheugenruimte, welke wordt opgebouwd op plaatsen vanaf ENDAD.
De assembler kan ook hierom worden gehandhaafd omdat het een "stil" programma is: tijdens de uitvoering van het assemblerprogramma wordt er geen gebruik gemaakt van het display en men hoeft ook geen toetsen in te drukken. Pas als de assemblage is voltooid laat men dat weten via een oplichtend display.
Het startadres van de assembler was en is $1F51. Vroeger moest je na het editen toets RST indrukken, dit startadres opgeven en op de GO-toets drukken. Een alternatief was om de NMI-sprongvektor te richten op het startadres van de assembler en, als het zo ver was, toets ST in te drukken. Dat hoeft nu, bij PME, allemaal niet meer.
Want we hebben de X-toetsfunktie.

Drukt men de toets X in, dan wordt er na een paar voorbereidende akties door PME naar de assembler gesprongen. Eén van de voorbereidende akties houdt in dat de NMI-sprongvektor wordt gericht op een adres binnen PME, waarnaar moet worden gesprongen na het assembleren. Na het indrukken van toets X zien we na verloop van tijd (de tijdsduur hangt af van de grootte van het te assembleren programma) het zeven-segment-display oplichten. Op dat moment is men klaar met het assembleren. Indien men vervolgens toets ST van het standaardtoetsenbord indrukt ontstaat er een NMI die, zoals gezegd, leidt tot de terugkeer naar PME.
En wat er dàn gebeurt is erg interessant.
Want wàt gebeurt er?
Alle labels worden op het tv-scherm of printerpapier afgedrukt! Dus zowel het labelnummer als het bijbehorende adres. Dat ziet er zò uit (een denkbeeldig programma met 5 labels):

LAB $1Ø: $ØØØØ LAB $12: $ØØØ8 LAB $14: $ØØØC LAB $13: $Ø213
LAB $15: $ØØ22B
PM EDITOR
XXXX ZZ ZZ ZZ

Dit is allemaal mogelijk omdat de labels na afloop van het assembleren nog steeds in het geheugen staan. Drie plaatsen voor elk label en de labelgeheugenruimte begint bij het adres ENDAD en strekt zich uit in de richting met afnemend adres.
We zien dat er maximaal vier labels op een regel worden afgedrukt. De labels worden in een volgorde afgedrukt die overeenkomt met de volgorde waarin ze tijdens de eerste fase van het assembleren worden ontdekt, genoteerd en verwijderd. De labels worden dus afgedrukt in de volgorde met toenemend adres. Waarom dat zo is kunt u natrekken in hoofdstuk 9 van boek 2, waar de assembler-software wordt besproken. In hoeverre de volgorde: toenemend adres overeenkomt met de volgorde: toenemend labelnummer hangt af van de gebruiker. Men is niet gebonden aan een bepaalde volgorde van labelnummers, hoewel het ter wille van de overzichtelijkheid aanbeveling verdient om dat wèl te doen. In het voorbeeld van daarnet is de labelnummervolgorde 1Ø, 12, 14, 13, 15. Het labelnummer 11 komt kennelijk niet voor, misschien wel omdat in het te assembleren programma een absoluut operandadres XX11 voorkomt (met XX hoogstwaarschijnlijk gelijk aan ØØ).
Nadat alle labels zijn afgedrukt (in dit uitzonderingsgeval is er geen sprake van een scheiding tussen de rechter aktiekolom en de linker reaktiekolom) volgt de terugmelding "PM EDITOR", op een nieuwe regel. Vervolgens wordt op wéér een nieuwe regel de eerste instruktie van het geassembleerde programma afgedrukt; het adres XXXX is dus het adres BEGAD. Na afloop van het assembleren en het vervolgens opsommen van alle labels is er dus naar de warme startingang van PME gesprongen.

Zo, nu zijn alle toetsfunkties van PME besproken. Het is een hele waslijst geworden. Nu een heleboel praktijk, waarbij we van de gelegenheid gebruik maken om verder te gaan met iets heel nuttigs, waarmee we in hoofdstuk 12 van boek 3 zijn begonnen. Namelijk: stoeien met PM-subroutines.

PME in de praktijk

De praktijk is the proof of the pudding — of iets in die geest

Eerst maar eens een warming up.

1. Acht-bits hex-decimaalomzetting

In hoofdstuk 5 van boek 2 (pagina's 18 . . . 25) is een praktijkvoorbeeld besproken ten behoeve van het leren omgaan met de standaard-editor. Het ging om een programma (DISPL plus de nodige subroutines) dat de decimale waarde van een acht-bits hexadecimaal getal op het display laat zien. Dat getal moet eerst worden opgegeven, via het indrukken van twee numerieke toetsen. Het programma zelf (lees: de aanpak, de oplossingsmetode oftewel het algoritme) is indertijd niet besproken en dat doen we ook nu niet. Temeer daar agendapunt 2 van dit praktijkgedeelte zich bezig houdt met een uitgebreide versie, die wèl zal worden besproken.
Nee, het gaat ons uitsluitend om de illustratie van het verschil tussen de PME-gang van zaken en het toetsengebeuren bij de standaard-editor.
Konkreet: vergelijk de listing op de pagina's 22 en 23 van boek 2 met tabel 3. In tabel 3 beginnen we met het opstarten van PM (terugmelding "JUNIOR"). Vervolgens starten we PME koud. De eerste "instruktie" is label 1Ø. Deze wordt via de Insert-toets I opgegeven. Dat moet omdat anders de 77 op adres ØØØØ blijft staan. Voor de variabele eindadreswijzer maakt het niet uit of je de eerste instruktie via Input of via Insert opgeeft; die CEND schuift hoedanook op naar een hoger adres. Maar een programma beginnen met de "opcode" 77 (ØØØØ is tevens het startadres!), dat is vràgen om moeilijkheden.
Alle volgende instrukties en labels worden via Input opgegeven: gewoon maar numerieke toetsen indrukken! Een stuk of wat regels verder in tabel 3 treft u een K aan en is de volgende instruktie via toets I opgegeven. Dat kwam omdat abusievelijk 85 F7 werd opgegeven, in plaats van 85 D7. Die fout is daarmee hersteld.
Binnen een paar minuten zit het hele programma in het geheugen. De kontrole of we het goed hebben gedaan gaat nu veel sneller dan vroeger. Immers, in het minimum-geval (Elekterminal) kun je zestien regels in één oogopslag bekijken. En heb je de beschikking over een geschikte printer, dan is dat "één oogopslag" hooguit niet van toepassing als er sprake is van een listing van 1 meter lang!
De volgende fase is het indrukken van toets X. Het programma wordt geassembleerd. Is dat gebeurd, dan licht het display op. We drukken daarna de toets ST ("STOP", "NMI") van het standaardtoetsenbord in. Met als gevolg dat PME alle labels ophoest en na deze hoestbui zich terugmeldt met "PM EDITOR" en het afdrukken van de eerste instruktie van het geassembleerde programma. Drie keer toets P indrukken volstaat om een volledige listing te maken van het geassembleerde programma.
En met een beetje handigheid en routine is die tabel 3 binnen vijf minuten gemaakt.
Alleen het van de band inlezen van kant en klare programma's gaat nòg sneller . . . *lees verder op pagina 32* ➔

Tabel 3. De opgave via PME van het programma DISPL en zijn subroutines. Zie pagina 18 . . . 23 van boek 2 en verder praktijkvoorbeeld 1 in de tekst.

①

```
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77                 IFF 10 00
0200 FF 10 00           A9 00
0203 A9 00              85 F9
0205 85 F9              85 FA
0207 85 FA              85 FB
0209 85 FB              FF 11 00
020B FF 11 00           20 6F 1D
020E 20 6F 1D           10 10
0211 10 10              85 F9
0213 85 F9              85 F7
0215 85 F7              K
0215 77                 I85 D7
0215 85 D7              20 12 00
0217 20 12 00           4C 11 00
021A 4C 11 00           FF 12 00
021D FF 12 00           20 14 00
0220 20 14 00           85 FA
0223 85 FA              84 D7
0225 84 D7              20 14 00
0227 20 14 00           A2 04
022A A2 04              FF 13 00
022C FF 13 00           0A
022F 0A                 CA
0230 CA                 D0 13
0231 D0 13              05 FA
0233 05 FA              85 FA
0235 85 FA              84 FB
0237 84 FB              60
0239 60                 FF 14 00
023A FF 14 00           A0 00
023D A0 00              84 D8
023F 84 D8              20 15 00
0241 20 15 00           18
0244 18                 A5 D7
0245 A5 D7              69 0A
0247 69 0A              60
0249 60                 FF 15 00
024A FF 15 00           38
024D 38                 A5 D7
024E A5 D7              E9 0A
0250 E9 0A              85 D7
0252 85 D7              A5 D8      →②
```

②

```
0254 A5 D8          E9 00
0256 E9 00          30 16
0258 30 16          C8
025A C8             4C 15 00
025B 4C 15 00       FF 16 00
025E FF 16 00       60
0261 60             X
LAB $10: $0200 LAB $11: $0208 LAB $12: $0217 LAB $13: $0223
LAB $14: $022E LAB $15: $023B LAB $16: $024C
PM EDITOR
0200 A9 00          P
0202 85 F9
0204 85 FA
0206 85 FB
0208 20 6F 1D
020B 10 F3
020D 85 F9
020F 85 D7
0211 20 17 02
0214 4C 08 02
0217 20 2E 02
021A 85 FA
021C 84 D7
021E 20 2E 02
0221 A2 04
0223 0A             P
0224 CA
0225 D0 FC
0227 05 FA
0229 85 FA
022B 84 FB
022D 60
022E A0 00
0230 84 D8
0232 20 3B 02
0235 18
0236 A5 D7
0238 69 0A
023A 60
023B 38
023C A5 D7          P
023E E9 0A
0240 85 D7
0242 A5 D8
0244 E9 00
0246 30 04
0248 C8
0249 4C 3B 02
024C 60
024D 77
```

2. Zestien-bits hex-decimaalomzetting

Waar gaat het om? We willen een programma maken dat een opgegeven zestien-bits hexadecimaal getal na de afsluiting van de opgave van dat getal omzet en afdrukt in zijn decimale vorm. Ieder om te zetten hexadecimaal getal moet op het begin van een nieuwe regel worden opgegeven en afgedrukt. De afsluiting van de opgave van dat getal moet aan de juniorcomputer kenbaar worden gemaakt via het indrukken van de toets ":". De decimale waarde moet achter de dubbele punt op dezelfde regel worden afgedrukt. Ook getallen van 4, 8 of 12 bits (1, 2 respektievelijk 3 numerieke toetsen) moeten kunnen worden verwerkt. Drukt men (per ongeluk?) meer dan vier numerieke toetsen achter elkaar in, dan moet het getal, gevormd door de laatste vier ingedrukte toetsen, als uitgangspunt dienen. Ziedaar de opdracht die we in konkrete software moeten vertalen.
Hoe pakken we dit aan? Eerst maar even een voorbeeld aan de hand van een doordeweeks decimaal getal. In het getal 1981 zit een bijdrage van één duizendtal. Trekt men van het getal twee duizend(tallen) af, dan houden we een negatief getal over. In het geval 1981 zitten verder bijdragen van negen honderdtallen, acht tientallen en één eenheid.
Het hoogste 16-bits hexadecimale getal is $FFFF. Dat komt overeen met een decimaal getal 65535. Met andere woorden: de hoogste bijdrage in de decimale versie van het getal wordt gevormd door de tienduizendtallen. Maximaal zes stuks. We stellen vast dat

10.000_{10} overeenkomt met $271Ø,
1.000_{10} overeenkomt met $Ø3E8,
100_{10} overeenkomt met $ØØ64, en dat
10_{10} overeenkomt met $ØØØA.

We gaan nu het volgende doen. Nadat een getal volledig is opgegeven trekken we er net zo vaak $271Ø (tienduizend) van af totdat we een negatief getal overhouden. Is dat laatste het geval, dan tellen we er bij dat negatieve getal weer tienduizend op. Het aantal keren dat van het opgegeven getal tienduizend is afgetrokken zonder dat het resultaat van dat aftrekken nog net niet negatief wordt is bijgehouden in een register (naar zal blijken: het Y-register) en wordt afgedrukt. Men kan beredeneren dat na afloop van de omschreven procedure de bijdrage van tienduizendtallen in het restgetal nul is.
Vervolgens herhalen we deze procedure, maar dan voor de duizendtallen en op basis van het restgetal. Van dat getal wordt net zo vaak $Ø3E8 afgetrokken totdat het resultaat negatief is. Het nieuwe restgetal ontstaat door bij dat negatieve resultaat – àls dat optreedt, want je kunt ook precies op nul uitkomen – $Ø3E8 op te tellen. Op dezelfde manier wordt het aantal honderdtallen en tientallen bepaald en afgedrukt. Na afloop daarvan is er geen sprake meer van een restgetal maar van een restcijfer; de mogelijkheden variëren van 0 tot en met 9. Dit restcijfer (het aantal eenheden) wordt direkt, zonder rekenpartijen afgedrukt.
Voordat we overgaan tot de bespreking van het eigenlijke programma waarmee een en ander zal worden gerealiseerd geven we eerst even een opgave van de in dat programma gebruikte PM-subroutines:

- **CRLF.** Adres $11E8. Twee opeenvolgende grafische kommando's, die tot gevolg hebben dat er op het begin van een nieuwe regel wordt

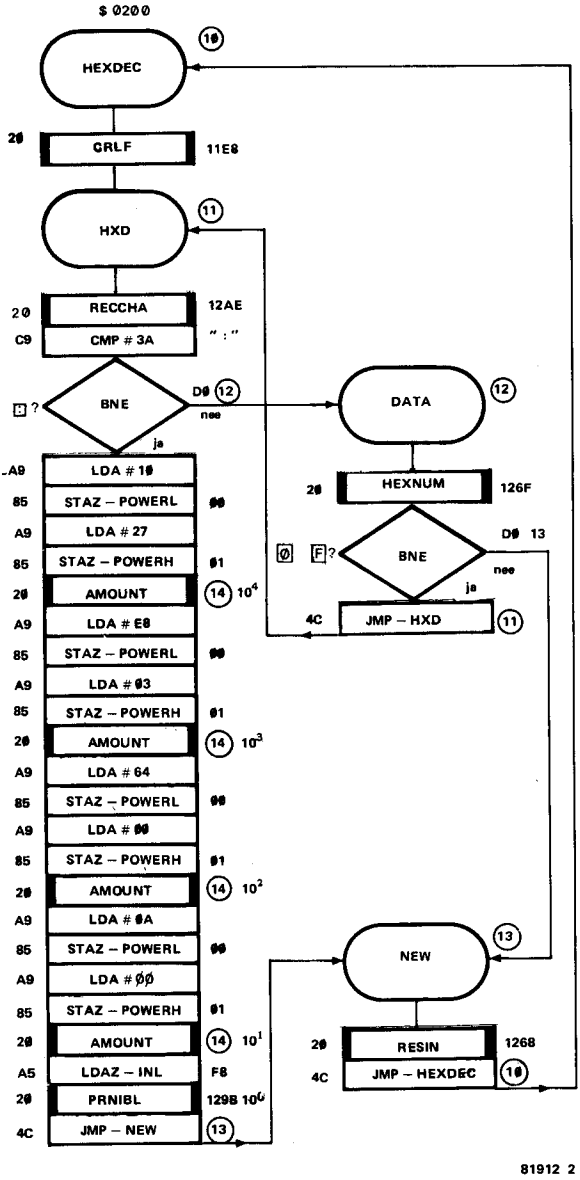
begonnen.

- **RECCHA.** Adres $12AE. Wacht op een ingedrukte toets en zet de ASCII-kode daarvan in de accu.
- **HEXNUM.** Adres $126F. Verwerkt binnengekomen numerieke data. Na omzetting van de ASCII-kode in een datanibble wordt dit datanibble van rechts uit de ingangsbuffer INL ingeschoven. De buffers INH en INL bevatten data die overeenkomt met de meest recente vier ingedrukte numerieke toetsen. Een linker nibble is altijd "ouder" dan een rechter nibble. Indien het om niet-numerieke data ≠ Ø . . . 9 en A . . . F gaat volgt de foutmelding "WHAT?".
- **RESIN.** Adres $1268. Maakt de inhoud van de buffers INL en INH nul.
- **PRNIBL.** Adres $129B. Is de tweede helft van de subroutine PRBYT. Zorgt voor het afdrukken van het rechter datanibble van de accu-inhoud.

Voor meer details verwijzen we naar hoofdstuk 14.

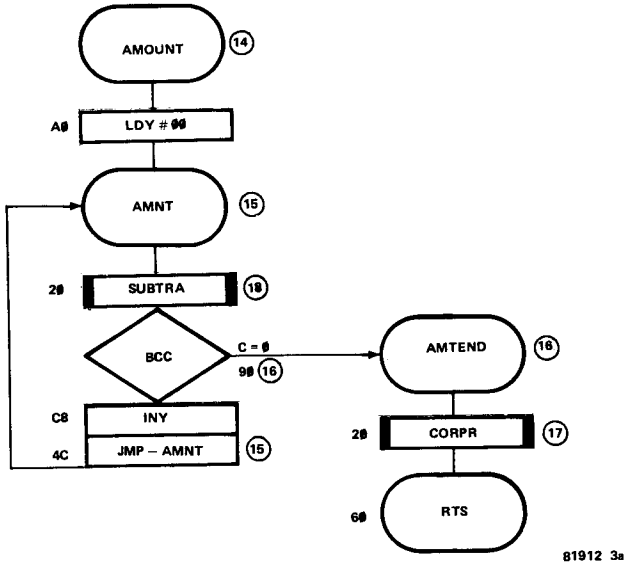
Dan nu het programma. De hoofdroutine heet – heel toepasselijk – HEXDEC en staat in figuur 2. Eerst zorgen we voor het begin op een nieuwe regel (CRLF) en het wachten op een ingedrukte toets. En dan moet er worden gekeken of die ingedrukte toets soms de dubbele-punt-toets was. Zoja, dan is de opgave van het om te zetten getal kennelijk voltooid en kan met de eerder beschreven omzettings- en afdrukprocedure worden begonnen.
Maar laten we eerst maar eens gaan kijken wat er gebeurt als er geen dubbele punt werd vastgesteld, maar bijvoorbeeld en hoogstwaarschijnlijk numerieke data. De BNE voert ons dan naar het label DATA, en naar de subroutine HEXNUM. Indien het geen numerieke toets was zorgt HEXNUM voor een foutmelding "WHAT?" en verder voor een Z-vlag nul. In dat geval leidt de op HEXNUM volgende BNE ons naar het label NEW: de inhoud van de buffers INL en INH wordt nul gemaakt en we gaan terug naar AF oftewel naar HEXDEC. We moeten dan opnieuw beginnen met de opgave van een getal.
Is het wèl een numerieke toets, dan zorgt HEXNUM ervoor dat het met de toets overeenkomende datanibble het rechter datanibble van INL wordt. De inhoud van INH en INL is altijd zodanig dat INH van de meest recente vier ingedrukte numerieke toetsen de eerste twee overeenkomende datanibbles bevat en INL de laatste twee. Verder is een linker datanibble altijd "ouder" dan een rechter datanibble. Niet gebruikte nibbles zijn ØØ.
Laten we eens gaan kijken wat er gebeurt als er een dubbele punt is vastgesteld: het programmadeel onder de linker BNE van figuur 2. Vier keer achter elkaar worden de geheugenplaatsen POWERL ($ØØØØ) en POWERH ($ØØØ1) met data gevuld die iets te maken heeft met de vraag of we bezig zijn met de bepaling van het aantal tienduizendtallen, duizendtallen, honderdtallen danwel het aantal tientallen. Vier keer achter elkaar wordt er een beroep gedaan op de subroutine AMOUNT, die zorgt voor de bepaling van het aantal -tallen, alsmede voor het afdrukken daarvan. Na vier keer AMOUNT zijn de aantallen -tallen bekend en afgedrukt en houden we het restcijfer, dus het aantal eenheden, over. Dat restcijfer staat in INL en wordt via de subroutine PRNIBL afgedrukt. Rest nog de sprong terug naar HEXDEC via NEW (buffers nul maken), voor de behandeling van een volgende getal.

★2

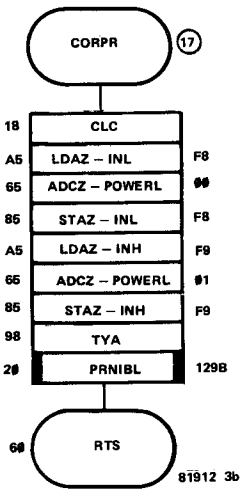


Figuur 2. De hoofdroutine HEXDEC van het programma waarmee het mogelijk is om een opgegeven hexadecimaal getal van maximaal vier cijfers om te zetten in het overeenkomstige decimale getal en dat vervolgens af te drukken.

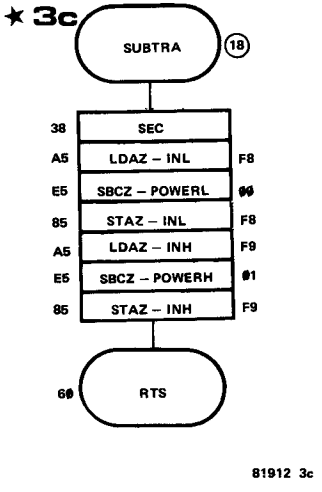
★ 3a



★ 3b



★ 3c



Figuur 3. De subroutine AMOUNT van figuur 3a bepaalt hoe vaak een bepaalde macht van tien voorkomt in het opgegeven hexadecimale getal en zorgt er vervolgens voor dat dat aantal wordt afgedrukt. Er wordt gebruik gemaakt van de subroutines CORPR (figuur 3b) en SUBTRA (figuur 3c).

Nu meer details over de subroutine AMOUNT. Deze staat in figuur 3a. Deze subroutine begint met het nul maken van Y. Daarna, na het label AMNT, wordt de subroutine SUBTRA aangeroepen. Die subroutine staat in figuur 3c. Van het getal (INH, INL) wordt het getal, gevormd door de inhoud van (POWERH, POWERL) afgetrokken. Is het resultaat daarvan positief of nul, dan is de carryvlag C één. Een negatief resultaat levert een carryvlag nul op. Die carry-geschiedenis is heel eenvoudig te onthouden. Er geldt: carry=borrow en dus borrow=carry. Als het resultaat negatief is moet er worden geleend. Dus is de borrow 1. Dus is de carry nul. Bij een resultaat groter dan of gelijk aan nul hoeft er niets te worden geleend.
Indien het resultaat niet negatief is volgt een verhoging van Y (INY) en gaan we opnieuw aftrekken: de sprong terug naar AMNT. Als het resultaat van het aftrekken negatief is volgt de sprong naar AMTEND. De waarde van Y komt overeen met het aantal keren dat het getal, gevormd door de inhoud van (INH, INL) is verlaagd zonder dat dit tot een negatief resultaat aanleiding gaf.
De rest van AMOUNT is gauw verteld. Na het label AMTEND wordt de subroutine CORPR van figuur 3b afgewerkt. Eerst wordt bij het getal (INH, INL) het getal (POWERH, POWERL) opgeteld. Met als gevolg dat de inhoud van INH en INL (het restgetal) in overeenstemming is met de situatie bij het berekenen van het volgende aantal -tallen. Rest nog het kopiëren van Y in A (TYA) en de aanroep van PRNIBL: het zojuist berekende aantal -tallen wordt afgedrukt.
We gaan de programma's van de figuren 2 en 3 aan de junior-computer opgeven via PME. In tabel 4 zijn al onze toetsakties en alle daarbij horende reakties van PME in de vorm van een "hard copy" zichtbaar gemaakt. Achtereenvolgens worden PM opgestart, PME koud gestart en alle instrukties opgegeven; de eerste instruktie netjes volgens de regels via toets I. Na het indrukken van X wordt er geassembleerd. Het vervolgens indrukken van toets ST van het standaardtoetsenbord levert levendig drukwerk op: de labels worden stuk voor stuk afgedrukt, met als slotakkoord "PM EDITOR" plus de eerste instruktie op het adres Ø2ØØ.
We willen het programma uitvoeren. En moeten dus eerst PME verlaten want na die labelgeschiedenis zijn we PME binnengekomen via de warme startingang. Dus wat doen we? We drukken achtereenvolgens de toetsen RST, 1, drie keer Ø, GO en RUB OUT (= RES) in: PM is opgestart. Dit is noodzakelijk omdat het programma HEXDEC gebruik maakt van PM-subroutines. De I/O moet in overeenstemming hiermee worden vastgelegd.
Dus het opstarten via de standaard-monitor is verboden!
Na het starten van HEXDEC is er een aantal hexadecimale getallen opgegeven en, na de afsluitende dubbele punt decimaal afgedrukt. U kunt ook zien dat er een getal ØØØØØ wordt afgedrukt indien we meteen de dubbele punttoets indrukken.
Kortom: het programma werkt; het doet wat het moest doen. Ter wille van de dokumentatie willen we wel graag weten hoe het programma er in geassembleerde vorm uitziet. Dus starten we PME warm. U ziet in tabel 4 dat als startadres 153D genomen is, in plaats van 1533. Dat mag omdat het programmagedeelte van 1533 tot en met 153C zich bezig houdt met het specificeren van de BRK-sprongvektor en het resetten van de stack pointer (zie hoofdstuk 15). De keuze 153D in plaats van 1533 is geoorloofd omdat
lees verder op pagina 40 ➔

Tabel 4. De opgave via PME van het programma HEXDEC en de bijbehorende subroutines. Zie verder de figuren 2 en 3 en praktijkvoorbeeld 2 in de tekst.

①

```
SIXTEEN BIT HEXADECIMAL TO DECIMAL CONVERSION
HEXDEC
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 10 00
0200 FF 10 00        20 E8 11
0203 20 E8 11        FF 11 00
0206 FF 11 00        20 AE 12
0209 20 AE 12        C9 3A
020C C9 3A           D0 12
020E D0 12           A9 10
0210 A9 10           85 00
0212 85 00           A9 27
0214 A9 27           85 01
0216 85 01           20 14 00
0218 20 14 00        A9 E8
021B A9 E8           85 00
021D 85 00           A9 03
021F A9 03           85 01
0221 85 01           20 14 00
0223 20 14 00        A9 64
0226 A9 64           85 00
0228 85 00           A9 00
022A A9 00           85 01
022C 85 01           20 14 00
022E 20 14 00        A9 0A
0231 A9 0A           85 00
0233 85 00           A9 00
0235 A9 00           85 01
0237 85 01           20 14 00
0239 20 14 00        A5 F8
023C A5 F8           20 9B 12
023E 20 9B 12        4C 13 00
0241 4C 13 00        FF 12 00
0244 FF 12 00        20 6F 12
0247 20 6F 12        D0 13
024A D0 13           4C 11 00
024C 4C 11 00        FF 13 00
024F FF 13 00        20 68 12
0252 20 68 12        4C 10 00
0255 4C 10 00        FF 14 00
0258 FF 14 00        A0 00
025B A0 00           FF 15 00
025D FF 15 00        20 18 00
0260 20 18 00        90 16        →②
```

```
                                        ②
0263 90 16          C8
0265 C8             4C 15 00
0266 4C 15 00       FF 16 00
0269 FF 16 00       20 17 00
026C 20 17 00       60
026F 60             FF 17 00
0270 FF 17 00       18
0273 18             A5 F8
0274 A5 F8          65 00
0276 65 00          85 F8
0278 85 F8          A5 F9
027A A5 F9          65 01
027C 65 01          85 F9
027E 85 F9          98
0280 98             20 9B 12
0281 20 9B 12       60
0284 60             FF 18 00
0285 FF 18 00       38
0288 38             A5 F8
0289 A5 F8          E5 00
028B E5 00          85 F8
028D 85 F8          A5 F9
028F A5 F9          E5 01
0291 E5 01          85 F9
0293 85 F9          60
0295 60
0296 77             X
LAB $10: $0200 LAB $11: $0203 LAB $12: $023E LAB $13: $0246
LAB $14: $024C LAB $15: $024E LAB $16: $0257 LAB $17: $025B
LAB $18: $026D
PM EDITOR
0200 20 E8 11
JUNIOR

200
0200 20 R
FFFF:65535
FFF:04095
FF:00255
F:00015
ABCD:43981
1000:04096
T
WHAT?

CF16:53014
14F8:05368
17FF:06143
1024:04132
2710:10000
03E8:01000
0064:00100          →③
```

③

```
000A:00010
:00000

JUNIOR

153D
153D A0 R
PM EDITOR
0200 20 E8 11        P
0203 20 AE 12
0206 C9 3A
0208 D0 34
020A A9 10
020C 85 00
020E A9 27
0210 85 01
0212 20 4C 02
0215 A9 E8
0217 85 00
0219 A9 03
021B 85 01
021D 20 4C 02
0220 A9 64
0222 85 00           P
0224 A9 00
0226 85 01
0228 20 4C 02
022B A9 0A
022D 85 00
022F A9 00
0231 85 01
0233 20 4C 02
0236 A5 F8
0238 20 9B 12
023B 4C 46 02
023E 20 6F 12
0241 D0 03                                    ④
0243 4C 03 02
0246 20 68 12        P              0264 65 01
0249 4C 00 02                       0266 85 F9          P
024C A0 00                          0268 98
024E 20 6D 02                       0269 20 9B 12
0251 90 04                          026C 60
0253 C8                             026D 38
0254 4C 4E 02                       026E A5 F8
0257 20 5B 02                       0270 E5 00
025A 60                             0272 85 F8
025B 18                             0274 A5 F9
025C A5 F8                          0276 E5 01
025E 65 00                          0278 85 F9
0260 85 F8                          027A 60
0262 A5 F9          →④              027B 77
```

we tòch niet van plan zijn om de BRK-toets te gebruiken (zouden we dat wèl hebben gedaan dan volgt "JUNIOR", oftewel terug naar PM). Zoals ook blijkt uit het laatste gedeelte van tabel 4, waar het gaat om een listing van HEXDEC plus subroutines in de vorm van drie volledige en één onvolledig afgedrukte instruktieblokken van 16 instrukties.

3. Decimaal optellen

Ons volgende praktijkvoorbeeld betreft een programma waarvan het praktisch nut afhangt van de vraag of u al een zakrekenmachientje ("japannertje") in huis hebt of niet. Want met zo'n ding gaat de door ons beoogde opteloperatie nog veel sneller in zijn werk en bovendien is het repertoire aan andere rekenkundige kunsten niet te verwaarlozen. Maar bij ons, hier en nu, ligt de nadruk op het leren omgaan met PM-subroutines en met PME.
Er moet een programma worden gemaakt dat het volgende doet: Ga uit van een decimaal getal van zes cijfers dat door de gebruiker is opgegeven, of van een decimaal getal van maximaal acht cijfers dat het resultaat is van vroegere optellingen (het zogenaamde cumulatieve getal). Bij één van die twee getallen (welke, dat wordt automatisch geregeld) wordt een derde decimaal getal opgeteld dat door de gebruiker wordt opgegeven. Met als gevolg dat er een (nieuw) cumulatief getal ontstaat waarbij men vervolgens wéér een getal kan optellen, enzovoorts.
Het eerste getal (van zes cijfers) wordt door de gebruiker opgegeven door het indrukken van maximaal zes numerieke toetsen 0 . . . 9, gevolgd door het indrukken van de punttoets ".". Vervolgens geeft de gebruiker een getal van maximaal zes cijfers op dat daarbij moet worden opgeteld. De gang van zaken is: eerst dat getal intoetsen en dan de toets "P" (van plus) indrukken. We hebben voor "P" in plaats van "+" gekozen omdat in het laatste geval telkens de Shifttoets moet worden ingedrukt. Dat vinden we lastig. Het indrukken van alle toetsen die afwijken van ".", "P" en 0 . . . 9, dus óók de hexadecimale numerieke toetsen A . . . F, heeft de bekende foutmelding "WHAT?" tot gevolg.
De praktijk ziet er zo uit:

```
123456.
    123456
654321P
+   654321
=00777777
1P
+   000001
=00777778
```

De reaktie van de junior-computer is vet gedrukt; onze toetsakties zijn in een normaal lettertype weergegeven. Ze zien onze toetsakties overigens niet in dezelfde vorm terug. Een regel die wordt afgesloten met "." of "P" wordt namelijk overschreven door de eerstvolgende "reaktieregel" van de computer. Dit houdt in dat dit programma niet of hooguit met konsessies op een papierprinter kan worden afgedraaid omdat er dan twee regels over elkaar worden afgedrukt. Bij een video terminal zoals de Elekterminal is dit geen probleem. Bij een papierprinter kan men de hardware-echo – die ervoor zorgt dat een door de gebruiker ingedrukte toets wordt afgedrukt –

uitschakelen. Dat gaat via het in de stand "full duplex" zetten van de "half/full duplex"-schakelaar die doorgaans op randapparatuur aanwezig is. Door het "overschrijven" van de gebruikersregels ontstaat een overzichtelijk geheel, met de diverse getallen netjes onder elkaar, zoals we dat vroeger op de lagere school moesten doen.
Het programma heet DECADD en staat in figuur 4. In dit programma en in de bijbehorende subroutines komen allerlei geheugenplaatsen voor, die we allereerst zullen bespreken.

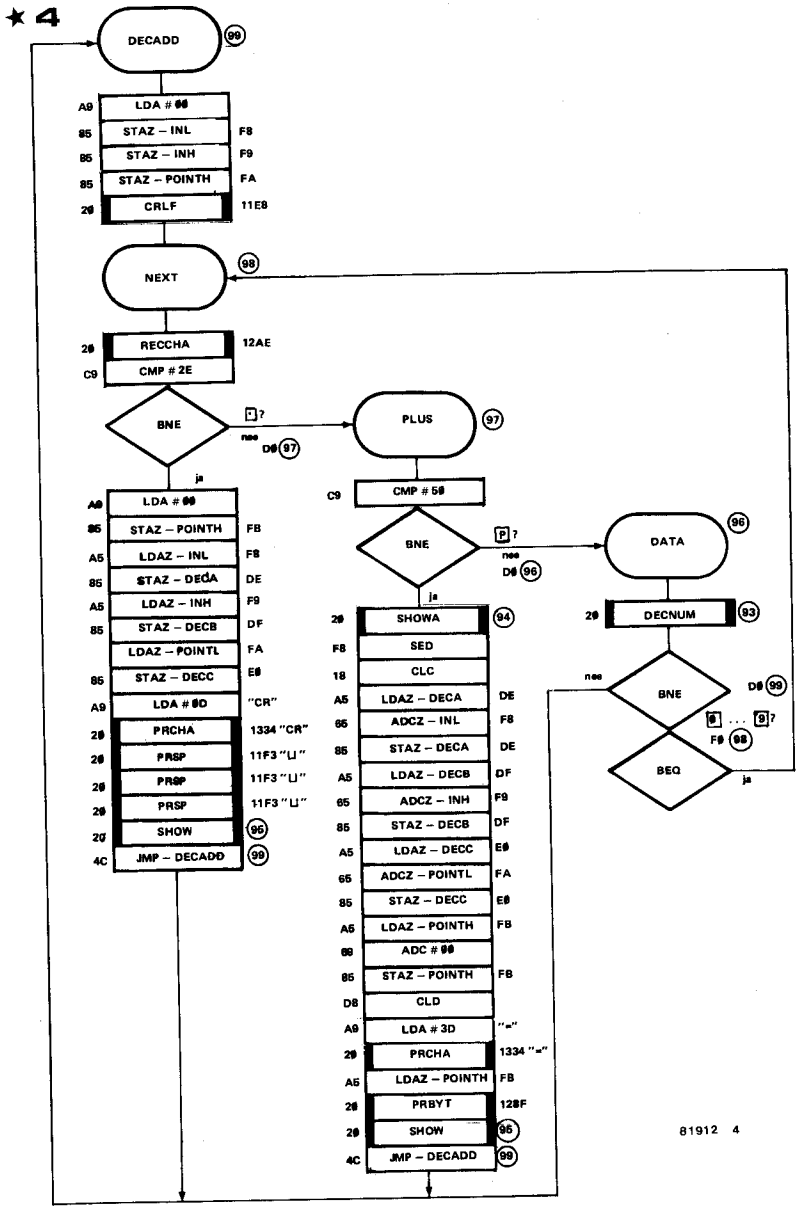
INL	adres $ØØF8	opgave getal 10^0 en 10^1
INH	adres $ØØF9	opgave getal 10^2 en 10^3
POINTL	adres $ØØFA	opgave getal 10^4 en 10^5
DECA	adres $ØØDE	cumulatief getal 10^0 en 10^1
DECB	adres $ØØDF	cumulatief getal 10^2 en 10^3
DECC	adres $ØØEØ	cumulatief getal 10^4 en 10^5
POINTH	adres $ØØFB	cumulatief getal 10^6 en 10^7

De maximale hoogte van het cumulatief getal, dus van de optelling, bedraagt derhalve 99.999.999.
In DECADD plus de bijbehorende subroutines worden de volgende PM-subroutines gebruikt:

- **CRLF, RECCHA**: zie praktijkvoorbeeld 2.
- **PRCHA.** Adres $1334. Drukt een karakter af of verzorgt een grafisch kommando waarvan de ASCII-kode in de accu staat.
- **PRSP.** Adres $11F3. Drukt een spatie af.
- **PRBYT.** Adres $128F. Drukt achtereenvolgens het linker en het rechter nibble van de accu-inhoud af. Dit na omzetting van de nibbles in de bijbehorende ASCII-kode.
- **MESSY.** Adres $11D6. Zorgt voor het afdrukken van tekst, die uit een look up-tabel afkomstig is. Welke tekst? Dat hangt af van de beginwaarde van Y en van de positie van het EOT-teken $Ø3 in de look up-tabel.

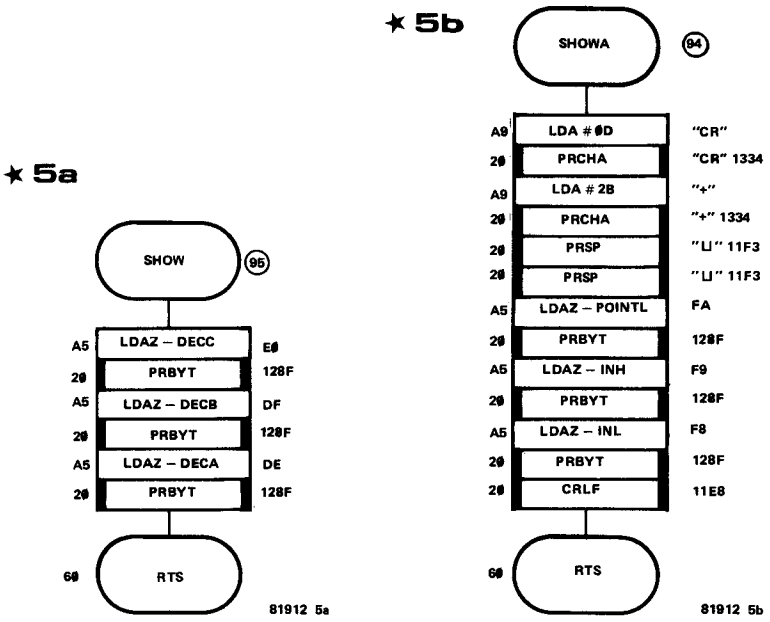
Ook nu verwijzen we u voor meer details naar hoofdstuk 14.

Het programma DECADD (figuur 4) begint met het nul maken van de drie buffers voor de opgave van een getal. Verder wordt er begonnen op een nieuwe regel (CRLF). En dan wacht het programma op een ingedrukte toets (RECCHA). De toetsen ".", P en 0 . . . 9 hebben iets te betekenen voor het programma; alle andere toetsen spelen geen enkele rol.
Stel dat er een toets is ingedrukt en dat het niet "." is en ook niet P. In DECADD komen we dan bij het label DATA terecht en daarmee bij de subroutine DECNUM. Dat is het decimale broertje van HEXNUM. Nu worden uitsluitend de toetsen 0 . . . 9 tot geldige data verwerkt. In alle andere gevallen volgt de terugmelding "WHAT?" en wordt de Z-vlag nul gemaakt. Een ingedrukte toets 0 . . . 9 wordt tot een nibble verwerkt dat van rechts uit buffer INL wordt ingeschoven, met als gevolg dat alle bestaande nibbles in de drie getalbuffers een plaatsje naar links opschuiven. De inhoud van POINTL, INH en INL bevat dus de data die overeenkomt met de zes meest recente ingedrukte toetsen. Een linker nibble is altijd "ouder" dan een rechter nibble. Het oorspronkelijke linker nibble van POINTL gaat verloren. Zijn er minder dan zes meest recente toetsen 0 . . . 9, dus is er een getal van minder dan zes cijfers opgegeven, dan zijn de nog niet gebruikte nibbles van de buffers nul. Na afloop van DECNUM



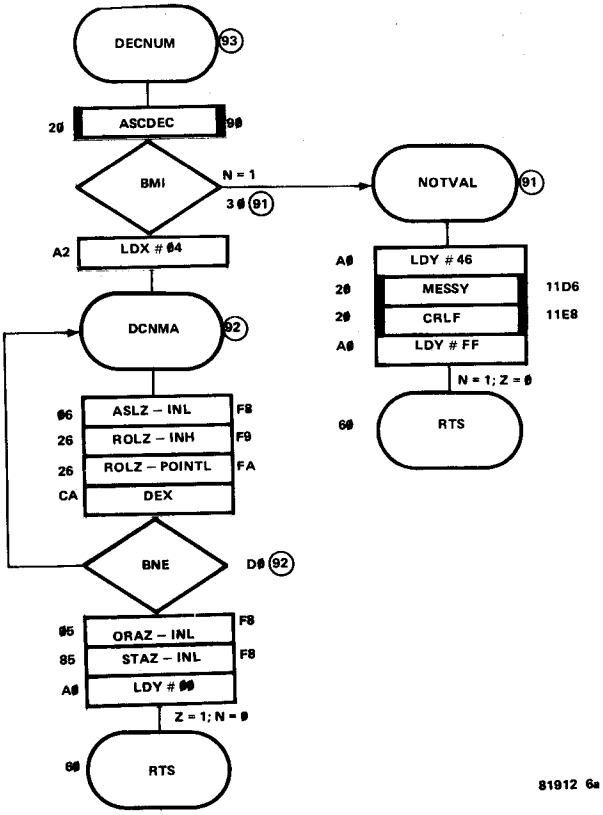
Figuur 4. De hoofdroutine DECADD van het programma waarmee men een decimaal getal van maximaal 6 cijfers kan optellen bij een ander decimaal getal van acht cijfers: het zogenaamde cumulatieve getal.

bepaalt de Z-vlag of we opnieuw beginnen (na een ontdekte foute toets), of dat we op een eventuele volgende numerieke toets moeten wachten; de sprong naar het label NEXT.
Wat gebeurt er nadat de plustoets is ingedrukt, dus nadat de opgave van een getal is voltooid? We beginnen dan met een schone lei: de buffer POINTH – bestemd voor het gevolg van het uitstijgen van de optelsom (= cumulatief getal) boven 999.999 – wordt nul gemaakt. En dan wordt de inhoud van de getalbuffers gekopieerd in de buffers die bestemd zijn voor het onthouden van het cumulatieve getal. Dat wil zeggen: in drie van de vier buffers, want POINT blijft buiten schot.
Dan volgt een aantal grafische aktiviteiten. Drukwerk! Eerst wordt het CR-kommando uitgevoerd: ga terug naar het begin van dezelfde regel. Dan volgen drie spaties (drie keer achter elkaar PRSP) en krijgt SHOW werk te doen: het zojuist opgegeven getal wordt afgedrukt. Daarna gaan we terug naar DECADD, met een schone lei en beginnend op een nog schone volgende regel.
De subroutine SHOW staat in figuur 5a en omvat drie laadoperaties en drie aanroepen van PRBYT. Heel simpel.
Nu gaan we eens kijken wat er gebeurt als toets P is ingedrukt. Dat wil zeggen: er is een getal opgegeven dat bij het nu te wijzigen cumulatieve getal moet worden opgeteld. Het begint met de subroutine SHOWA. Laten we die maar meteen meepakken. Hij is te vinden in figuur 5b. Ook deze



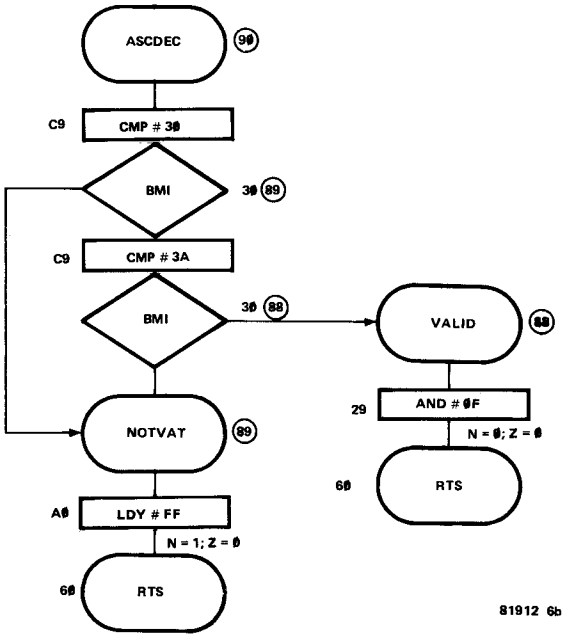
Figuur 5. De subroutine SHOW (figuur 5a) van DECADD (figuur 4) zorgt voor het afdrukken van de rechter zes cijfers van het cumulatieve getal. De subroutine SHOWA van DECADD staat in figuur 5b. Hij zorgt voor het afdrukken van "+ xxxxxx", waarbij "xxxxxx" het getal is dat bij het cumulatieve getal moet worden opgeteld, met als gevolg een nieuw cumulatief getal.

★ 6a



subroutine is één en al drukwerk. Vanaf het begin van dezelfde regel worden achtereenvolgens afgedrukt: een "+", twee spaties plus het getal dat men zojuist heeft opgegeven plus het begin op een nieuwe regel. Dit getal moet worden opgeteld bij het cumulatieve getal en het resultaat van de optelling doet vervolgens dienst als nieuw cumulatief getal. De toestand van de carryvlag bepaalt (ADC #ØØ) of de inhoud van de overloopbuffer POINTH zich wijzigt of niet. De optelling vindt decimaal plaats; aan het begin van de optelling zien we de instruktie SED en direkt na afloop van het optellen schakelen we terug (CLD). Dit is nodig omdat een aantal subroutines van PM niet korrekt wordt uitgevoerd als ze met een D-vlag 1 worden doorlopen. In hoofdstuk 12 van boek 3 hebben we het uitgebreid over deze kwestie gehad.
Na de optelling wordt op de volgende regel een "=" afgedrukt. Dan wordt de inhoud van POINTH afgedrukt en tenslotte zorgt SHOW voor het afdrukken van de inhoud van de overige drie buffers voor het cumulatieve getal. Rest nog de sprong naar DECADD en klaar is Kees.
Dan de subroutine DECNUM. Hij staat in figuur 6a. Het begint al meteen

✶ 6b



Figuur 6. De subroutines DECNUM (figuur 6a) en ASCDEC (figuur 6b) van DECADD (figuur 4) en van DECHEX (figuur 7) zorgen voor de verwerking van de geldige decimale numerieke data 0 . . . 9 (dus niet A . . . F!) in de getalbuffers POINTL, INH en INL.

goed met de aanroep van weer een andere subroutine, namelijk ASCDEC. Deze staat in figuur 6b. De ASCII-kode van de ingedrukte toets staat in de accu. Voor de toetsen 0 . . . 9 gelden de ASCII-kodes 30 . . . 39. Het resultaat van ASCDEC is dat voor een ongeldige toets de N-vlag 1 is en dus de Z-vlag nul. Voor een geldige toets 0 . . . 9 (label VALID) wordt via de operatie AND #0F de ASCII-kode 30 . . . 39 omgezet in een datanibble 00 . . . 09.
Terug naar DECNUM. De BMI na ASCDEC voert ons voor een ongeldige toets naar het label NOTVAL. Na het afdrukken van de boodschap "WHAT?" (MESSY, met Y=46, zie hoofdstuk 14) wordt de N-vlag 1 gemaakt. Dus de Z-vlag is dan nul. Een geldige toets wordt verwerkt in de inhoud van INL, INH en POINTL. Alla datanibbles schuiven een plaatsje naar links op en op de vrijgekomen plaats komt het datanibble dat overeenkomt met de ingedrukte toets (ORAZ-INL plus STAZ-INL). Tot slot wordt de Z-vlag één gemaakt. Daardoor is de N-vlag nul geworden. We gaan DECADD en zijn subroutines editen en vervolgens assembleren. Hoe dat precies in zijn werk gaat kunt u zien in tabel 5. Na het opstarten van
lees verder op pagina 50 ➔

Tabel 5. De opgave via PME van het programma DECADD en de bijbehorende subroutines. Zie verder de figuren 4, 5 en 6 en verder praktijkvoorbeeld 3 in de tekst.

①

```
SIX DIGIT DECIMAL ADDITION
UP TO 99,999,999
DECADD
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77              IFF 99 00
0200 FF 99 00        A9 00
0203 A9 00           85 F8
0205 85 F8           85 F9
0207 85 F9           85 FA
0209 85 FA           20 E8 11
020B 20 E8 11        FF 98 00
020E FF 98 00        20 AE 12
0211 20 AE 12        C9 2E
0214 C9 2E           D0 97
0216 D0 97           A9 00
0218 A9 00           85 FB
021A 85 FB           A5 F8
021C A5 F8           85 DE
021E 85 DE           A5 F9
0220 A5 F9           85 DF
0222 85 DF           A5 FA
0224 A5 FA           85 E0
0226 85 E0           A9 0D
0228 A9 0D           20 34 13
022A 20 34 13        20 F3 11
022D 20 F3 11        20 F3 11
0230 20 F3 11        20 F3 11
0233 20 F3 11        20 95 00
0236 20 95 00        4C 99 00
0239 4C 99 00        FF 97 00
023C FF 97 00        C9 50
023F C9 50           D0 96
0241 D0 96           20 94 00
0243 20 94 00        F8
0246 F8              18
0247 18              A5 DE
0248 A5 DE           65 F8
024A 65 F8           85 DE
024C 85 DE           A5 DF
024E A5 DF           65 F9
0250 65 F9           85 DF
0252 85 DF           A5 E0
0254 A5 E0           65 FA
0256 65 FA           85 E0      →②
```

```
0258 85 E0          A5 FB
025A A5 FB          69 00
025C 69 00          85 FB
025E 85 FB          D8
0260 D8             A9 3D
0261 A9 3D          20 34 13
0263 20 34 13       A5 FB
0266 A5 FB          20 8F 12
0268 20 8F 12       20 95 00
026B 20 95 00       4C 99 00
026E 4C 99 00       FF 96 00
0271 FF 96 00       20 93 00
0274 20 93 00       D0 99
0277 D0 99          F0 98
0279 F0 98          FF 95 00
027B FF 95 00       A5 E0
027E A5 E0          20 8F 12
0280 20 8F 12       A5 DF
0283 A5 DF          20 8F 12
0285 20 8F 12       A5 DE
0288 A5 DE          20 8F 12
028A 20 8F 12       60
028D 60             FF 94 00
028E FF 94 00       A9 0D
0291 A9 0D          20 34 13
0293 20 34 13       A9 2B
0296 A9 2B          20 34 13
0298 20 34 13       20 F3 11
029B 20 F3 11       20 F3 11
029E 20 F3 11       A5 FA
02A1 A5 FA          20 8F 12
02A3 20 8F 12       A5 F9
02A6 A5 F9          20 8F 12
02A8 20 8F 12       A5 F8
02AB A5 F8          20 8F 12
02AD 20 8F 12       20 E8 11
02B0 20 E8 11       60
02B3 60             FF 93 00
02B4 FF 93 00       20 90 00
02B7 20 90 00       30 91
02BA 30 91          A2 04
02BC A2 04          FF 92 00
02BE FF 92 00       06 F8
02C1 06 F8          26 F9
02C3 26 F9          26 FA
02C5 26 FA          CA
02C7 CA             D0 92
02C8 D0 92          05 F8
02CA 05 F8          85 F8
02CC 85 F8          A0 00
02CE A0 00          60
02D0 60             FF 91 00  →③
```

```
                                        ③
02D1 FF 91 00       A0 46
02D4 A0 46          20 D6 11
02D6 20 D6 11       20 E8 11
02D9 20 E8 11       A0 FF
02DC A0 FF          60
02DE 60             FF 90 00
02DF FF 90 00       C9 30
02E2 C9 30          30 89
02E4 30 89          C9 3A
02E6 C9 3A          30 88
02E8 30 88          FF 89 00
02EA FF 89 00       A0 FF
02ED A0 FF          60
02EF 60             FF 88 00
02F0 FF 88 00       29 0F
02F3 29 0F          60
02F5 60
02F6 77             X
LAB $99: $0200 LAB $98: $020B LAB $97: $0236 LAB $96: $0268
LAB $95: $026F LAB $94: $027F LAB $93: $02A2 LAB $92: $02A9
LAB $91: $02B9 LAB $90: $02C4 LAB $89: $02CC LAB $88: $02CF
PM EDITOR
0200 A9 00          P
0202 85 F8
0204 85 F9
0206 85 FA
0208 20 E8 11
020B 20 AE 12
020E C9 2E
0210 D0 24
0212 A9 00
0214 85 FB
0216 A5 F8
0218 85 DE
021A A5 F9
021C 85 DF
021E A5 FA
0220 85 E0          P
0222 A9 0D
0224 20 34 13
0227 20 F3 11
022A 20 F3 11
022D 20 F3 11
0230 20 6F 02
0233 4C 00 02
0236 C9 50
0238 D0 2E
023A 20 7F 02
023D F8
023E 18
023F A5 DE
0241 65 F8      →④
```

④

```
0243 85 DE          P
0245 A5 DF
0247 65 F9
0249 85 DF
024B A5 E0
024D 65 FA
024F 85 E0
0251 A5 FB
0253 69 00
0255 85 FB
0257 D8
0258 A9 3D
025A 20 34 13
025D A5 FB
025F 20 8F 12
0262 20 6F 02       P
0265 4C 00 02
0268 20 A2 02
026B D0 93
026D F0 9C
026F A5 E0
0271 20 8F 12
0274 A5 DF
0276 20 8F 12
0279 A5 DE
027B 20 8F 12
027E 60
027F A9 0D
0281 20 34 13
0284 A9 2B
0286 20 34 13       P
0289 20 F3 11
028C 20 F3 11
028F A5 FA
0291 20 8F 12
0294 A5 F9
0296 20 8F 12
0299 A5 F8
029B 20 8F 12
029E 20 E8 11
02A1 60
02A2 20 C4 02
02A5 30 12
02A7 A2 04
02A9 06 F8
02AB 26 F9          P
02AD 26 FA
02AF CA
02B0 D0 F7
02B2 05 F8
02B4 85 F8
02B6 A0 00      →⑤
```

⑤

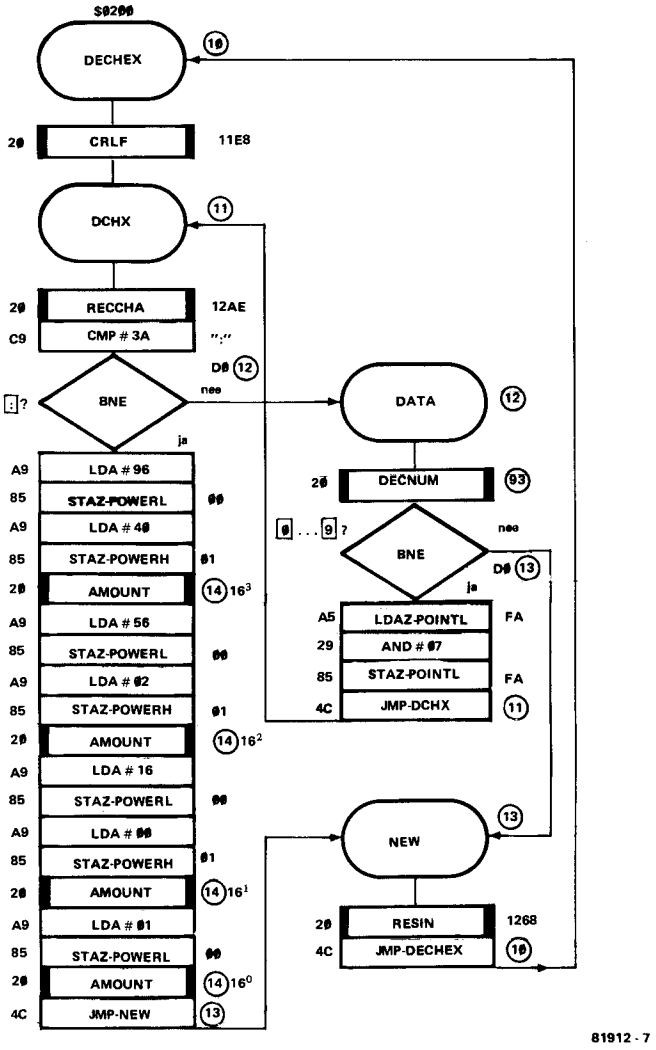
```
02B8 60
02B9 A0 46
02BB 20 D6 11
02BE 20 E8 11
02C1 A0 FF
02C3 60
02C4 C9 30
02C6 30 04
02C8 C9 3A          P
02CA 30 03
02CC A0 FF
02CE 60
02CF 29 0F
02D1 60
02D2 77
```

PM (terugmelding "JUNIOR") en na de koude start van PME worden alle instrukties opgegeven. De eerste instruktie via de I-toets. Na het indrukken van X wordt het programma geassembleerd. Na het indrukken van toets ST van het standaardtoetsenbord volgt het afdrukken van alle labels en de sprong naar de warme startingang van PME. Het programma in zijn geassembleerde vorm wordt gelist via het een aantal keren indrukken van toets P.
Omdat een gebruikersregel zoals eerder opgemerkt wordt overschreven door de eerste reaktieregel van de junior-computer hebben we in tabel 5 geen gebruiksvoorbeelden van DECADD gegeven. Het is overigens mogelijk om de gebruikersregels te laten staan door op twee plaatsen in het programma de instrukties die zorgen voor het "afdrukken" van een CR te vervangen door JSR-CRLF. Het is een goede oefening voor u om te weten waar dat is en om de wijzigingen aan te brengen via PME in het nog niet geassembleerde programma. U kunt dat doen via de in praktijkvoorbeeld 5 behandelde procedure. Zie verderop in dit hoofdstuk.

4. Van decimaal naar hexadecimaal

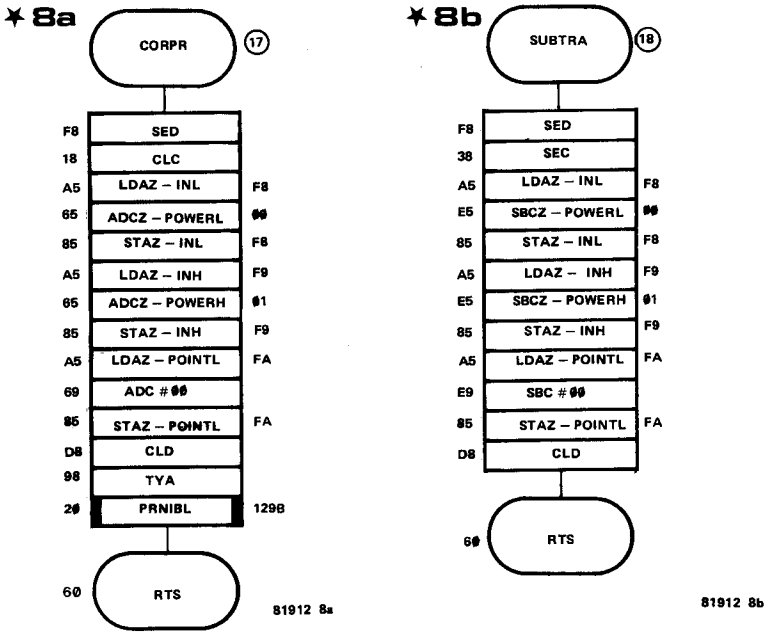
Als het mogelijk is om een programma HEXDEC te maken (zie figuur 2 en praktijkvoorbeeld 2), dan moet een programma DECHEX toch óók tot de mogelijkheden behoren. Een programma dus dat een opgegeven decimaal getal in het met dat getal overeenkomende hexadecimale getal omzet en vervolgens afdrukt. Dat is inderdaad mogelijk, getuige het programma DECHEX van figuur 7. Een snelle blik op die figuur 7 leert ons dat de overeenkomst met figuur 2 groot is. U ziet tevens dat voor de opgave van een getal (label DATA in figuur 7) gebruik wordt gemaakt van de subroutine DECNUM (zie ook figuur 6a). Dat betekent dat er drie getalbuffers beschikbaar zijn. Als we dat willen kunnen we dus een decimaal getal van maximaal zes cijfers opgeven.
Maar dat willen we niet. We zijn slechts geïnteresseerd in getallen van 65535 ($FFFF) en lager. Dit omdat hexadecimale getallen die langer zijn dan 16 bits (65536 en hoger) geen enkele praktische betekenis hebben in het microprocessor-gebeuren.
Zoals al opgemerkt lijkt DECHEX als twee druppels water op HEXDEC. We moeten achtereenvolgens vaststellen (subroutine AMOUNT) hoeveel keer het getal 4096 (16^3 oftewel $1000) in het opgegeven decimale getal voorkomt. En vervolgens hoeveel keer het getal 256 (16^2 oftewel $100) voorkomt. En vervolgens hoeveel keer het getal 16 (16^1 oftewel $10) voorkomt. En tenslotte moet het programma DECHEX nagaan hoeveel eenheden 16^0 ($1) er in het opgegeven decimaal getal zitten.
Voor opgegeven decimale getallen $\geqslant$65536 (16^4 = $10000) gaat het fout met DECHEX omdat er dan geen aantallen 65536 worden vastgesteld (dat zou hoogstens een aantal van één opleveren omdat er in principe een decimaal getal van maximaal 999.999 kan worden opgegeven); in plaats daarvan komt er een aantal bijdragen 4096 uit de bus dat groter is dan 15 ($F). Ondanks deze waarschuwing is er in DECHEX een maatregel genomen om de schade aan afgedrukte onzin (let wel: in het geval van een opgegeven getal $\geqslant$65536!) te beperken. Het linker nibble van de buffer POINTL bevat het aantal honderdduizendtallen van het opgegeven deci-

★7



Figuur 7. De hoofdroutine DECHEX van het programma waarmee een opgegeven decimaal getal dat kleiner is dan of gelijk is aan 65535 ($FFFF) wordt omgezet in het overeenkomstige hexadecimale getal, en vervolgens afgedrukt. De overeenkomst met figuur 2 (HEXDEC) is groot. Voor de subroutines DECNUM en ASCHEX: zie figuur 6a en 6b. De subroutine AMOUNT vindt men in figuur 3a. De subroutines CORPR en SUBTRA moeten echter worden aangepast; zie figuur 8.

male getal. Het rechter nibble van POINTL bevat het aantal tienduizendtallen. Door nu na het indrukken van een toets 0 . . . 9 (label DATA) de inhoud van POINTL te maskeren met Ø7 (AND #Ø7) wordt ervoor gezorgd dat het door DECHEX te verwerken decimale getal (na het indrukken van toest ":") geen honderdduizendtallen bevat en dat het aantal tienduizendtallen nooit hoger wordt dan 7.
De in DECHEX vier keer gebruikte subroutine AMOUNT kan hier ongewijzigd worden gebruikt. We verwijzen u dan ook voor AMOUNT naar figuur 3a. Echter, de in AMOUNT aangeroepen subroutines CORPR en SUBTRA moeten wèl worden aangepast aan hun taak in DECHEX. Vandaar dat u deze in gewijzigde vorm terugziet in de figuren 8a en 8b. Waarom die wijzigingen? Wel, er moet nu decimaal worden opgeteld (CORPR) en afgetrokken (SUBTRA). Verder moet de inhoud van POINTL ook worden aangepast, afhankelijk van de toestand van de carryvlag (ADC #ØØ in CORPR; SBC #ØØ in SUBTRA).
De praktijk van het editen en assembleren van DECHEX vindt u in tabel 6..
Lees verder op pagina 57 ➔



Figuur 8. Ten opzichte van figuur 3b respektievelijk figuur 3c zijn de subroutines CORPR en SUBTRA in die zin aangepast aan het gebruik bij DECHEX, dat er decimaal wordt gerekend en dat nu ook de inhoud van de getalbuffer POINTL moet worden aangepast.

Tabel 6. De opgave via PME van het programma DECHEX en de bijbehorende subroutines. Zie verder de figuren 8, 3a, 8a, 8b, 6a en 6b en verder praktijkvoorbeeld 4 in de tekst.

```
                                    ①
JUNIOR

1500
1500 20 R
BEGAD,ENDAD: 200,3FF
PM EDITOR
0200 77                 IFF 10 00
0200 FF 10 00           20 E8 11
0203 20 E8 11           FF 11 00
0206 FF 11 00           20 AE 12
0209 20 AE 12           C9 3A
020C C9 3A              D0 12
020E D0 12              A9 96
0210 A9 96              85 00
0212 85 00              A9 40
0214 A9 40              85 01
0216 85 01              20 14 00
0218 20 14 00           A9 56
021B A9 56              85 00
021D 85 00              A9 02
021F A9 02              85 01
0221 85 01              20 14 00
0223 20 14 00           A9 16
0226 A9 16              85 00
0228 85 00              A9 00
022A A9 00              85 01
022C 85 01              20 14 00
022E 20 14 00           A9 01
0231 A9 01              85 00
0233 85 00              20 14 00
0235 20 14 00           4C 13 00
0238 4C 13 00           FF 12 00
023B FF 12 00           20 93 00
023E 20 93 00           D0 13
0241 D0 13              A5 FA
0243 A5 FA              29 07
0245 29 07              85 FA
0247 85 FA              4C 11 00
0249 4C 11 00           FF 13 00
024C FF 13 00           20 68 12
024F 20 68 12           4C 10 00
0252 4C 10 00           FF 14 00
0255 FF 14 00           A0 00
0258 A0 00              FF 15 00
025A FF 15 00           20 18 00
025D 20 18 00           90 16
0260 90 16              C8
0262 C8                 4C 15 00    →②
```

```
                                    ②
0263 4C 15 00       FF 16 00
0266 FF 16 00       20 17 00
0269 20 17 00       60
026C 60             FF 17 00
026D FF 17 00       F8
0270 F8             18
0271 18             A5 F8
0272 A5 F8          65 00
0274 65 00          85 F8
0276 85 F8          A5 F9
0278 A5 F9          65 01
027A 65 01          85 F9
027C 85 F9          A5 FA
027E A5 FA          69 00
0280 69 00          85 FA
0282 85 FA          D8
0284 D8             98
0285 98             20 9B 12
0286 20 9B 12       60
0289 60             FF 18 00
028A FF 18 00       F8
028D F8             38
028E 38             A5 F8
028F A5 F8          E5 00
0291 E5 00          85 F8
0293 85 F8          A5 F9
0295 A5 F9          E5 01
0297 E5 01          85 F9
0299 85 F9          A5 FA
029B A5 FA          E9 00
029D E9 00          85 FA
029F 85 FA          D8
02A1 D8             60
02A2 60             FF 93 00
02A3 FF 93 00       20 90 00
02A6 20 90 00       30 91
02A9 30 91          A2 04
02AB A2 04          FF 92 00
02AD FF 92 00       06 F8
02B0 06 F8          26 F9
02B2 26 F9          26 FA
02B4 26 FA          CA
02B6 CA             D0 92
02B7 D0 92          05 F8
02B9 05 F8          85 F8
02BB 85 F8          A0 00
02BD A0 00          60
02BF 60             FF 91 00
02C0 FF 91 00       A0 46
02C3 A0 46          20 D6 11
02C5 20 D6 11       20 E8 11
02C8 20 E8 11       A0 FF       →③
```

```
                                  ③
02CB A0 FF          60
02CD 60             FF 90 00
02CE FF 90 00       C9 30
02D1 C9 30          30 89
02D3 30 89          C9 3A
02D5 C9 3A          30 88
02D7 30 88          FF 89 00
02D9 FF 89 00       A0 FF
02DC A0 FF          60
02DE 60             FF 88 00
02DF FF 88 00       29 0F
02E2 29 0F          60
02E4 60
02E5 77             X
LAB $10: $0200 LAB $11: $0203 LAB $12: $0235 LAB $13: $0243
LAB $14: $0249 LAB $15: $024B LAB $16: $0254 LAB $17: $0258
LAB $18: $0272 LAB $93: $0288 LAB $92: $028F LAB $91: $029F
LAB $90: $02AA LAB $89: $02B2 LAB $88: $02B5
PM EDITOR
0200 20 E8 11       P
0203 20 AE 12
0206 C9 3A
0208 D0 2B
020A A9 96
020C 85 00
020E A9 40
0210 85 01
0212 20 49 02
0215 A9 56
0217 85 00
0219 A9 02
021B 85 01
021D 20 49 02
0220 A9 16
0222 85 00          P
0224 A9 00
0226 85 01
0228 20 49 02
022B A9 01
022D 85 00
022F 20 49 02
0232 4C 43 02
0235 20 88 02
0238 D0 09
023A A5 FA
023C 29 07
023E 85 FA
0240 4C 03 02
0243 20 68 12
0246 4C 00 02       P
0249 A0 00
024B 20 72 02          →④
```

```
          ④
024E  90 04
0250  C8
0251  4C 4B 02
0254  20 58 02
0257  60
0258  F8
0259  18
025A  A5 F8
025C  65 00
025E  85 F8
0260  A5 F9
0262  65 01
0264  85 F9          P
0266  A5 FA
0268  69 00
026A  85 FA
026C  D8
026D  98
026E  20 9B 12
0271  60
0272  F8
0273  38
0274  A5 F8
0276  E5 00
0278  85 F8
027A  A5 F9
027C  E5 01
027E  85 F9          P
0280  A5 FA
0282  E9 00
0284  85 FA
0286  D8
0287  60
0288  20 AA 02
028B  30 12
028D  A2 04
028F  06 F8
0291  26 F9
0293  26 FA
0295  CA
0296  D0 F7
0298  05 F8
029A  85 F8          P
029C  A0 00
029E  60
029F  A0 46
02A1  20 D6 11
02A4  20 E8 11
02A7  A0 FF
02A9  60
02AA  C9 30
02AC  30 04  →⑤
```

```
          ⑤
02AE  C9 3A
02B0  30 03
02B2  A0 FF
02B4  60
02B5  29 0F
02B7  60                  P
02B8  77
JUNIOR

200
0200  20 R
4096:1000
256:0100
8192:2000
65535:FFFF
10000:2710
1000:03E8
100;
WHAT?

100:0064
4095:0FFF
255:00FF
15:000F
12345:3039
```

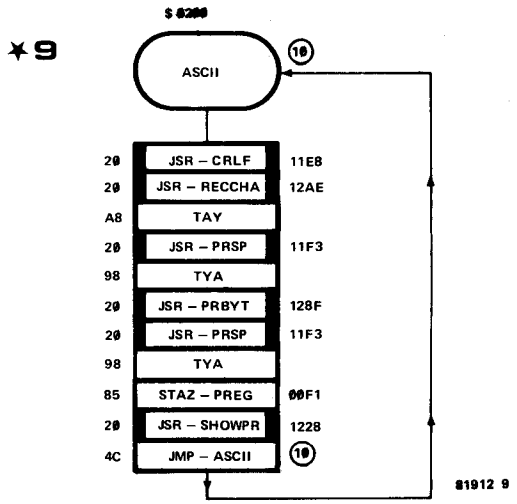
Achtereenvolgens worden opgegeven:
DECHEX ; labels 1Ø . . . 13 (figuur 7)
AMOUNT; labels 14 . . . 16 (figuur 3a)
CORPR ; label 17 (figuur 8a)
SUBTRA ; label 18 (figuur 8b)
DECNUM ; labels 93 . . . 91 (figuur 6a)
ASCDEC ; labels 9Ø . . . 88 (figuur 6b)
De verdere gang van zaken is u inmiddels vertrouwd: assembleren (X), terug naar PME (ST) via het afdrukken van de labels, het geassembleerde programma afdrukken (P's) en het programma starten na de terugkeer in PM. En tot slot het programma DECHEX uitvoeren. Zo lang als men wil. Genoeg geDECHEXt? Drukt dan RST in en kom eventueel terug via het starten van PM.

5. Warme CEND-start

PME en de cassette-band

In hoofdstuk 11 van boek 2 hebben we het gehad over de mogelijkheid om met behulp van TM een nog niet geassembleerd programma op de band te zetten alvorens het te assembleren en uit te proberen. Vooral bij grote programma's is het dan weinig werk — als blijkt dat er nog aan dat programma moet worden gesleuteld — het programma in zijn ongeassembleerde vorm van de band terug te lezen en om vervolgens via een warme CEND-start de toestand te herstellen zoals die was vlak voordat indertijd dat programma op de band werd gezet. Vervolgens kan men dan het eigenlijke sleutelwerk ("debuggen") uitvoeren en de beschreven procedure herhalen, net zo vaak als dat nodig is. Of juister: nodig blijkt.
We willen de te volgen procedures behandelen aan de hand van een piepklein voorbeeldprogrammaatje. Het is het programma ASCII van hoofdstuk 12 van boek 3. Zie figuur 9. Dit programma is zo klein dat het eigenlijk niet hoeft te worden geedit en ook niet hoeft te worden geassembleerd. Het label 1Ø is namelijk bekend; het is gelijk aan het startadres BEGAD! Desondanks hebben we dit korte programma genomen omdat het u en ons meterslange listings (hard copy) bespaart. En een goed overzicht is ook wat waard. En bovendien: het gaat erom dat u leert omgaan met een bepaalde "feature" van PME. U leert bij wijze van spreken omgaan met een hamer en dàt is belangrijker dan het feit dat het voorbeeld bestaat uit het in zachtboard aanbrengen van een punaise met die hamer!
Een listing van wat we allemaal met het programma ASCII gaan uitspoken vindt u in tabel 7. De gang van zaken zal punt voor punt worden behandeld.

① Het systeemprogramma PM wordt opgestart.
② Er volgt een koude start van PME. Het startadres van ASCII is $Ø2ØØ.
③ De instrukties worden opgegeven. We beginnen met het "inserten" van het label 1Ø. Zie verder figuur 9.
④ Alle instrukties zijn opgegeven. Het programma wordt via het indrukken van toets L kompleet opgehoest.
⑤ Het nog niet geassembleerde programma moet op de band worden gezet. Daartoe gaan we eerst terug naar PM (terugmelding "JUNIOR"), zetten de band startklaar in de opnamestand (pauzetoets!) en drukken



Figuur 9. Terwille van het leren omgaan met de warme CEND-ingang van PME is het programma ASCII uit hoofdstuk 12 van boek 3 hier nogmaals afgedrukt. Het startadres BEGAD is nu $0200. Waarom dit adres zo nodig het label 10 toegekend moest krijgen is in de tekst (praktijkvoorbeeld 5) verklaard.

achtereenvolgens de toetsen S, 8, 8, ", ", 2, Ø, Ø, "," 2, 1 en E in. Na het opnieuw indrukken van de pauzetoets drukken we de toets CR in; het programma wordt nu opgenomen en er volgt de terugmelding "READY". We stoppen de band. Als programmanummer is 88 genomen; SA is gelijk aan BEGAD en EA is gelijk aan CEND, dus een adres, één hoger dan het adres waarop de 77 staat (zie de listing, punt ④)

⑥ PME wordt warm gestart.

⑦ Het programma wordt geassembleerd (toetsen X en ST indrukken). Er is één label; dat wordt netjes afgedrukt.

⑧ Het geassembleerde programma wordt afgedrukt (toets P).

⑨ We gaan terug naar PM, teneinde het programma te starten en uit te proberen. Na de opgave van het startadres en na het starten willen we de ASCII-kode van toets A weten. Dat gaat prima. Dus het programma werkt naar behoren? Nee, dat doet het niet. Want vervolgens willen we de ASCII-kode van toets 6 weten. Die 6 wordt niet, zoals het hoort, op een nieuwe regel afgedrukt, laat staan dat er een ASCII-kode zichtbaar wordt. Potverdikkie, wat is hier aan de hand!? Misschien doen de toetsen B en C het wèl . . . Nee hoor: van hetzelfde laken een pak!
Kennelijk hebben we iets fout gedaan. Want in hoofdstuk 12 van boek 3 werkte het programma nog wèl! Er breekt een periode aan van grote ongerustheid. Tot overmaat van ramp valt er een kop koffie om over het nog zo maagdelijk boek 3 en men besluit om tòch maar
lees verder op pagina 61→

Tabel 7. De uitwerking op printerpapier van een "scenario", waarbij in een bepaalde fase gebruik wordt gemaakt van de warme CEND-startingang van PME. Zie verder figuur 9 en praktijkvoorbeeld 5 in de tekst.

```
                                   ①
① JUNIOR

② 1500
  1500 20 R
  BEGAD,ENDAD: 200,2FF
  PM EDITOR
③ 0200 77                    IFF 10 00
  0200 FF 10 00              20 E8 11
  0203 20 E8 11              20 AE 12
  0206 20 AE 12              A8
  0209 A8                    20 F3 11
  020A 20 F3 11              98
  020D 98                    20 8F 12
  020E 20 8F 12              20 F3 11
  0211 20 F3 11              98
  0214 98                    85 F1
  0215 85 F1                 20 28 12
  0217 20 28 12              4C 11 00
④ 021A 4C 11 00              L
  0200 FF 10 00
  0203 20 E8 11
  0206 20 AE 12
  0209 A8
  020A 20 F3 11
  020D 98
  020E 20 8F 12
  0211 20 F3 11
  0214 98
  0215 85 F1
  0217 20 28 12
  021A 4C 11 00
  021D 77
  DONE
  021E E8
⑤ JUNIOR

  S88,200,21E
  READY                    →②
```

```
                                    ②
⑥ 1533
  1533 A9 R
  PM EDITOR
⑦ 021E E8                 X
  LAB $10: $0200
  PM EDITOR
⑧ 0200 20 E8 11           P
  0203 20 AE 12
  0206 A8
  0207 20 F3 11
  020A 98
  020B 20 8F 12
  020E 20 F3 11
  0211 98
  0212 85 F1
  0214 20 28 12
  0217 4C 11 00
  021A 77
⑨ JUNIOR

  200
  0200 20 R
  A 41 0100000L6ABC
⑩ JUNIOR

  G88
  READY
⑪ 17C5
  17C5 20 R
  BEGAD,ENDAD: 200,2FF
  PM EDITOR
⑫ 021D 77                 S4C 11 00
  021A 4C 11 00           K
  DONE
  021A 4C 11 00           K
  021A 77                 I4C 10 00
  021A 4C 10 00
⑬ 021D 77                 X
  LAB $10: $0200
  PM EDITOR
⑭ 0200 20 E8 11           P
  0203 20 AE 12               →③
```

```
                    ③
0206  A8
0207  20  F3  11
020A  98
020B  20  8F  12
020E  20  F3  11
0211  98
0212  85  F1
0214  20  28  12
0217  4C  00  02
021A  77
⑮ JUNIOR

200
0200  20  R
A  41  01000001
B  42  01000010
C  43  01000011
D  44  01000100
1  31  00110001
2  32  00110010
3  33  00110011
4  34  00110100
K  4B  01001011
Z  5A  01011010
```

eerst naar het journaal van acht uur te kijken. Na afloop daarvan stelt men vast dat er ergere dingen in de wereld bestaan. Het gezonde denken wint het van de emotie. Toch de boel nog maar eens goed nakijken. En ja hoor, binnen een paar minuten klinkt er een "Eureka" door het huis. Gevónden. Met dat ene label dat er in ASCII zit is prompt iets fout gegaan. Er staat: "4C 11 ØØ" en dat moet natuurlijk "4C 1Ø ØØ zijn. Het label 11 is niet opgegeven en na het assembleren en nadat het met één toets (A) goed is gegaan wordt er naar het adres $ØØ11 gesprongen. Het programma ASCII is de mist ingegaan.
Remedie? Lees het programma terug van de band, breng een wijziging aan en assembleer het opnieuw. Enzovoorts.

⑩ Na het indrukken van toets RST wordt PM opgestart. We lezen het programma ASCII in zijn ongeassembleerde vorm van de band.

⑪ PME wordt gestart via de warme CEND-ingang. De adressen BEGAD en ENDAD blijven ongewijzigd. (Indien men vermoedt dat het sleutelen de toevoeging van een aantal instrukties inhoudt kan men eventueel ENDAD verhogen.) Nadat CEND weer dezelfde waarde (of juister: inhoud) heeft als destijds, vlak voor het op de band zetten, kunnen we verder gaan met het editen.

⑫ We zoeken de te wijzigen instruktie op via een searchkommando en drukken vervolgens op toets K, teneinde deze uit het geheugen te verwijderen (en later te vervangen door de korrekte instruktie). We dachten al met het "killen" bezig te zijn, maar eerst moeten we het searchen afbreken door het indrukken van een willekeurige toets ≠ toets Y. Dat was dus toets K. De korrekte waarde "4C ØØ ØØ" wordt opgegeven via de insert-toets L.
⑬ Het programma wordt opnieuw geassembleerd.
⑭ Het geassembleerde programma wordt afgedrukt.
⑮ We gaan terug naar PM en starten het programma ASCII.
Het werkt!

Nog wat losse praktijkopmerkingen

- Bij de koude, lauwe en warme CEND-start moeten we op het volgende letten: Tijdens de startfase, dus vóór het indrukken van de toets CR is de BRK-sprongvektor nog volgens PM vastgelegd. Drukt men tijdens het afdrukken van "BEGAD, ENDAD:" op de BRK-toets, dan volgt de terugmelding "JUNIOR" zodra men de BRK-toets heeft losgelaten; we zijn terug in PM. Maakt men een fout bij het opgeven van BEGAD en ENDAD, dan volgt de terugmelding "WHAT? " en wordt er vervolgens opnieuw "BEGAD, ENDAD:" afgedrukt. Waarom dat zo is kunt u nagaan in de hoofdstukken 14 en 15. Er volgt zoals opgemerkt geen foutmelding indien er een ENDAD wordt opgegeven die kwa adres lager is dan BEGAD.
- Nog even iets over het opgeven van adres-operanden die moeten worden geassembleerd. In hoofdstuk 5 van boek 2 hebben we verteld dat de opcode (2Ø voor een JSR,4C voor een JMP) moest worden gevolgd door een labelnummer (namelijk het nummer van het label waarnaar moet worden gesprongen) en door een tweede operand-byte, het begrenzer-byte (labelbegrenzer). Voor dit laatste byte is in hoofdstuk 5 en in de voorbeelden van dit hoofdstuk konsekwent het byte ØØ gekozen. Hetzelfde geldt voor het derde byte van een opgegeven label. Dit omdat "ØØ" lekker opvallend is. Men is echter niet verplicht om als derde byte ØØ te nemen. Dit volgt uit de gang van zaken tijdens de eerste fase van het assembleren. Zie hiertoe hoofdstuk 9 van boek 2. Belangrijk is dat er in het geheugen plaats wordt gereserveerd voor het tweede operand-byte van de JSR of JMP. Trouwens: een instruktie wordt pas in het geheugen gezet nadat een aantal bytes is opgegeven dat in overeenstemming is met de lengte van de instruktie. Bij PME krijgt men dat ondubbelzinnig in de gaten door de reaktie van PME op de nieuwe regel; op de linker kolom wordt de zojuist opgegeven instruktie afgedrukt.
- Kies nooit een labelnummer dat gelijk is aan het rechter byte ADL van een absoluut adres (dat dus niet hoeft te worden geassembleerd)! Ga dit na voordat u met editen begint en labelnummers moet kiezen.
- Vul bij spronginstrukties nooit alvast een absolute offset in als deze vooraf bekend is! De kans is groot dat die offset óók als labelnummer voorkomt.

- Gebruik elk labelnummer maar één keer. Hoewel het niet verplicht is verdient het aanbeveling om labelnummers toe te kennen in een volgorde met toe- of afnemend nummer (de eventuele verboden nummers gewoon overslaan) en in de volgorde waarin ze in het geheugen terecht komen tijdens het editen. Men kan dan vooraf al goed nagaan of het offsetbereik van een branch-instruktie niet wordt overschreden. Zonodig "brancht" men naar een label dat wordt gevolgd door een absolute sprong.

Tot zover nog wat praktische tips. Voor een deel ging het om herhalingsoefeningen van hoofdstuk 5 van boek 2. Maar uw oefeningen met PME zijn alles behalve herhalingsoefeningen. Het enige dat van de standaard-editor wordt herhaald is het elegante systeem van de hexadecimale labels en labelnummers. Wij wensen u een aangename tijd toe met PME – en met het bedenken van programma's.

14

1268 bytes PM-software

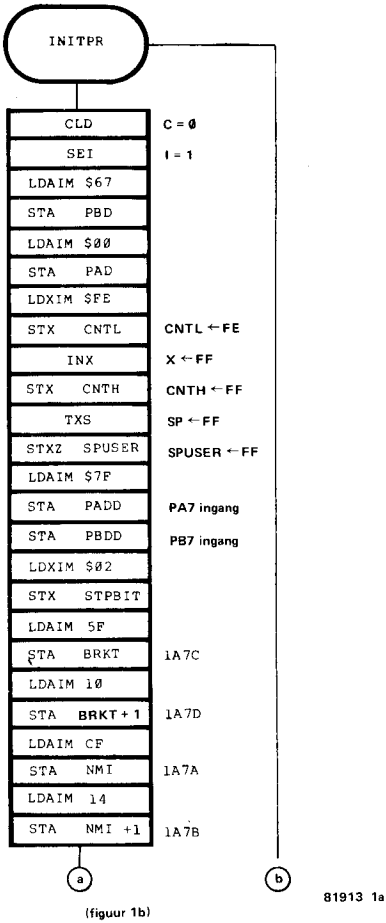
Het Junior-op-tv-programma

In hoofdstuk 12 van boek 3 heeft u het systeemprogramma PM leren kennen. U weet hoe u er mee om moet gaan en in dit hoofdstuk kunt u aan de weet komen hoe het in zijn werk gaat. Men kan zich natuurlijk afvragen of een gedetailleerd inzicht in de PM-software wel zo nuttig is en of men niet kan volstaan met de praktische omgangsvormen. Nog afgezien van het feit dat u niet verplicht bent om dit hoofdstuk te lezen zijn er twee argumenten aan te voeren om dat wèl te doen. Allereerst levert kennis van "hoe ze het gedaan hebben" bruikbare kennis op voor de ontwikkeling van eigen programma's. Ten tweede: allerlei PM-subroutines zijn in die ego-programma's toe te passen. Denk aan allerlei voorbeeldprogramma's uit de hoofdstukken 12 en 13.
Het nut van dit hoofdstuk is dus hetzelfde als het nut van vergelijkbare hoofdstukken van boek 2. Ten aanzien van de bespreking van de software moeten we ons echter de nodige beperkingen opleggen. Uit de titel van dit hoofdstuk kunt u al opmaken dat de hoeveelheid te bespreken software wel wat meer bytes inhoudt dan in boek 2 het geval was. Vandaar dat we de diverse onderdelen puntsgewijze bespreken en er hier en daar van uit gaan dat u bij het lezen van dit hoofdstuk óók al drie boeken achter de rug hebt. En dus de machinetaal van de 6502 al aardig onder de knie hebt.

1 Ieder systeemprogramma bestaat uit een hoofdroutine en één of meerdere voorroutines. En verder natuurlijk subroutines. De hoofdroutine heeft een bekende struktuur: er wordt gewacht op een ingedrukte toets en gekeken welke toets het was. Vervolgens wordt de bijbehorende

toetsfunktie uitgevoerd en gaan we terug naar een centraal punt en wachten we op een volgende toets: de kring is gesloten.
Een voorroutine wordt slechts incidenteel doorlopen. Bijvoorbeeld tijdens het starten van het programma. De voorroutine van PM staat in de figuren 1a en 1b. Hij is te vergelijken met de RESET-routine van de standaard-monitor en wordt doorlopen tijdens de eerste respektievelijk tweede start van PM; zie de pagina's 146 en 148 van boek 3. Het gedeelte na het label LABJUN (figuur 1b) wordt tevens doorlopen na het indrukken van de BRK-toets en na de foutmelding "JUNIOR". De routine STEP (figuur 1b) wordt doorlopen telkens nadat er een instruktie is uitgevoerd tijdens het stap voor stap uitvoeren van een programma en, in het algemeen, nadat een NMI is opgetreden.

★ 1a



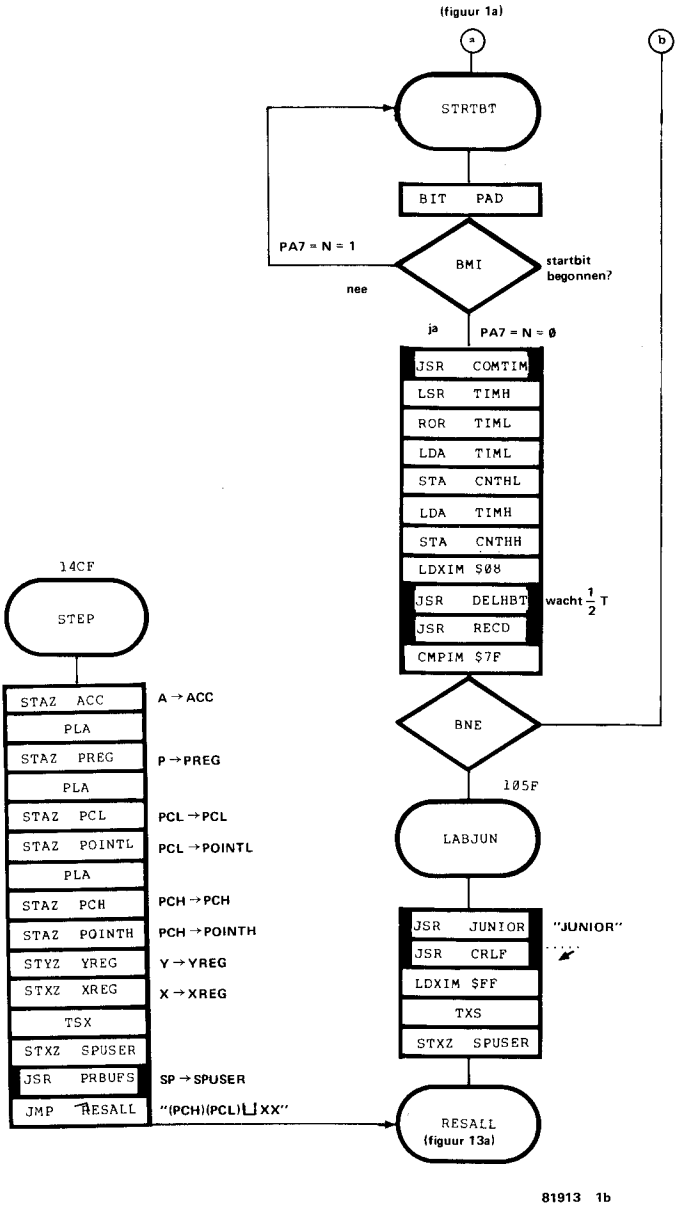
(figuur 1b)

Figuur 1a. Het eerste deel van de voorroutine van PM. En wel het gedeelte vanaf het label INITPR tot het label STRTBT.

De **voorroutine INITPR** (figuur 1a) begint met een aantal elementaire zaken. Er wordt overgeschakeld op binair rekenen en aan IRQ-interrupts hebben we geen boodschap (SEI). In figuur 1a wordt verder de I/O vastgelegd. Door het laden met 7F van PADD en PBDD zijn alle poortlijnen op PB7 en PA7 na tot uitgang verklaard. Poortlijn PB7 staat dus klaar om data van de band te ontvangen, PA7 is bereid om seriële data te ontvangen en PBØ staat klaar om data serieel te verzenden. Door PAD de waarde ØØ toe te kennen staat er op de uitgangen PAØ . . . PA6 een logische één. Dit zou kunnen leiden tot het oplichten van de zes displays (alle segmenten; zie hoofdstuk 7), ware het niet dat door het aan 67 gelijk maken van PBD de displayschakelaar in een neutrale stand staat. Kijk maar: met PBD = 67 komt overeen: PB1 = PB2 = 1 en PB3 = PB4 = Ø. Dan is de niet gebruikte uitgang "3" van de dekoder IC7 van de standaardkaart logisch nul; zie bijvoorbeeld figuur 10 op pagina 110 van boek 2. Door het logisch één maken van PB5 en PB6 worden de beide indikatie- en relais-stuurschakelingen die een rol spelen bij het cassette-gebeuren, uitgeschakeld (zie het schema van de interface-kaart in boek 3). En verder heeft het logisch één maken van de uitgang PBØ tot gevolg dat deze als seriële uitgang gebruikte uitgang het logische rustnivo aanneemt dat van toepassing is als er geen af te drukken karakters door de junior-computer worden verzonden (het logische rustnivo van de RS 232-lijn is dan nul).
Er valt nog veel meer voor te bereiden. De feitelijke en de gebruikers-stack pointer worden op FF gezet. De geheugenplaats STPBIT krijgt de waarde Ø2. Dat wil zeggen dat ieder door de computer verzonden ASCII-karakter wordt afgesloten met twee stopbits. Verder wordt de BRK-sprongvektor gespecificeerd op het startadres van LABJUN (figuur 1b; terugmelding "JUNIOR") en de NMI-sprongvektor idem dito op het startadres van STEP (eveneens figuur 1b). Rest nog één ding van figuur 1a: het laden van CNTL met FE en van CNTH met FF. Waarom? Dàt gaan we nu bespreken.

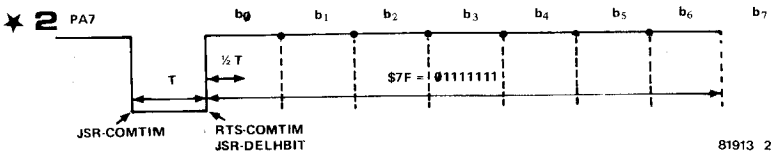
2 We zijn bij figuur 1b aangeland. Het programma van figuur 1a is helemaal doorlopen na de eerste start van PM. De vraag rijst wanneer we verder gaan met de tweede start. U weet van hoofdstuk 12 dat het voor die tweede start nodig is dat de als RUB OUT gebruikte toets RES wordt ingedrukt. Met als gevolg het verzenden van de bijbehorende ASCII-kode 7F. Te beginnen met de verzending van het startbit. En wat er zich dan vervolgens afspeelt is in figuur 2 geschetst. Het label STRTBT (figuur 1b) begint met het bekijken van de seriële ingang PA7 via een BIT-instruktie. Die BIT zorgt ervoor dat de N-vlag gelijk is aan bit 7 van de geheugeninhoud die wordt ge-BIT, en dat is PA7. Zodra het begin van een startbit is vastgesteld volgt de aanroep van de subroutine COMTIM. Deze staat in figuur 3. Gedurende de subroutine COMTIM wordt er gewacht totdat het startbit voorbij is. Dus gewacht totdat PA7 weer logisch één is geworden. Waarom dat "dus"? Omdat we weten dat de nu volgende seriële databits logisch één zijn. Er wordt immers gekeken of het ontvangen ASCII-karakter 7F is. Zie ook figuur 2.
Zolang het startbit nog aan de gang is wordt er periodiek bij het 16-bits getal, gevormd door de inhoud van de geheugenplaatsen CNTH en CNTL, één opgeteld. De uitvoering van die optelling kost een bepaalde, vaste tijd. De beginwaarde van (CNTH, CNTL) is FFFE. Naarmate de bittijd T hoger

★ 1b

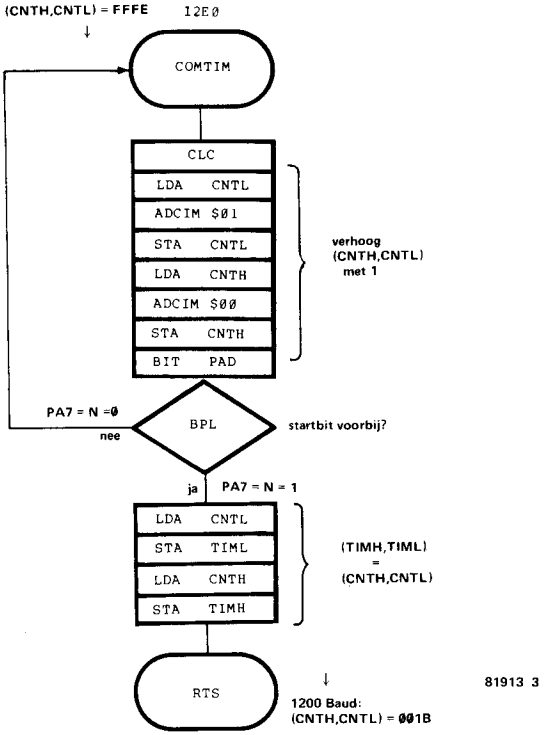


N.B. Voor de PM-hoofdroutine: zie de pagina's 85 . . . 87

Figuur 1b. Het vervolg van de voorroutine van figuur 1a (rechts in figuur 1b) en de STEP-voorroutine (links in figuur 1b).



Figuur 2. Het pulsdiagram van een serieel verzonden ASCII-karakter. Door één keer een halve bittijd ½T te wachten wordt het logische nivo van een databit altijd middenin de betreffende bittijd vastgesteld.



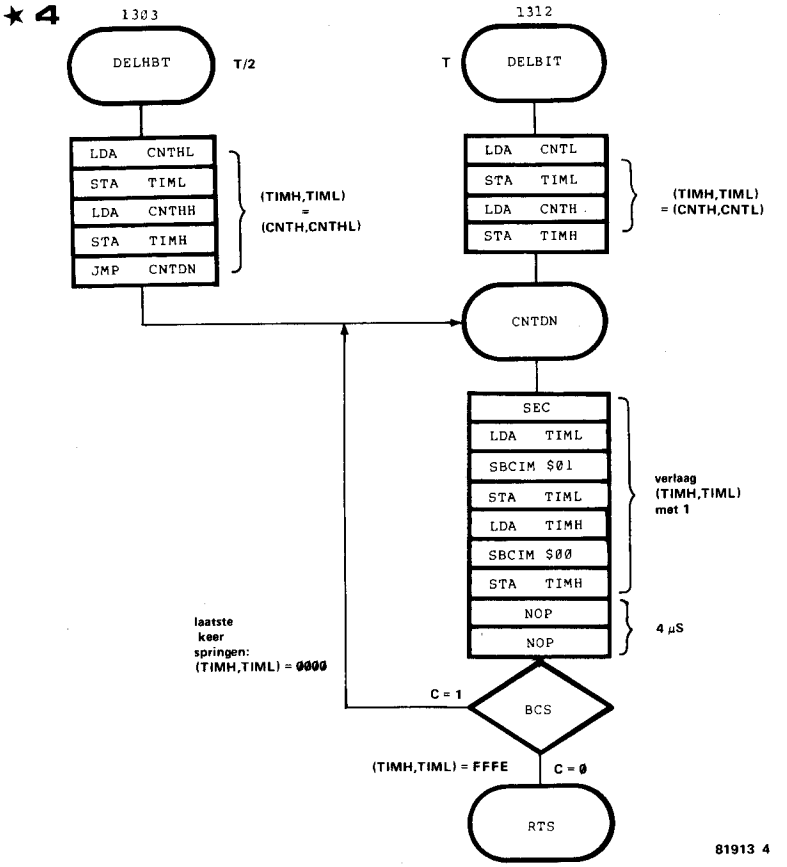
Figuur 3. De subroutine COMTIM bepaalt indirekt, namelijk via de inhoud van de geheugenplaatsen CNTH en CNTL, hoe lang het startbit van een verzonden ASCII-karakter 7F duurt. Daarmee ligt de bittijd T vast.

is zal ook de eindwaarde van (CNTH, CNTL) hoger zijn. De bittijd is bepaald via de data-verzendsnelheid, dus via Baudrate; deze wordt vastgelegd door de klokgenerator en de UART in het randapparaat. De bedoeling van die CNTH/CNTL-geschiedenis is ondermeer dat de junior-computer via PM ASCII-karakters verzendt met nagenoeg dezelfde snelheid als de snel-

heid waarmee ASCII-karakters naar de computer worden verzonden, door het randapparaat.
In de Elekterminal hebben we de baudrate-schakelaar laten vervallen. Zie hoofdstuk 12. Er is sprake van een vaste verzendsnelheid van 1200 Baud. Hiermee komt overeen een inhoud van CNTL van 1B en ØØ voor CNTH. Men kan dit kontroleren door na het opstarten van PM de geheugenplaatsen 1A5A (CNTL) en 1A5B (CNTH) te bekijken. Op deze manier, met twee geheugenplaatsen, is het voor de junior-computer mogelijk om zich in te stellen op een groot bereik van door de aangesloten randapparatuur bepaalde baudrates. Het gaat goed zolang de inhoud van CNTH niet hoger wordt dan FF, dus zolang de baudrate niet te laag is.
Aan het eind van COMTIM wordt de eindstand van (CNTH, CNTL) gekopieerd in de geheugenplaatsen TIMH en TIML.

3 Direkt na afloop van de subroutine COMTIM gebeurt er het een en ander met de inhoud van TIMH en TIML. Alle bits van het getal, gevormd door (TIMH, TIML) schuiven een plaatsje naar rechts op. Dat gaat via het naar rechts schuiven (LSR) van de bits van TIMH en het rechtsom draaien (ROR) van de bits van TIML. Dat heeft uiteindelijk tot gevolg dat het getal, gevormd door (CNTHH, CNTHL) ongeveer half zo groot is als het getal, gevormd door (CNTH, CNTL). Voor oneven getallen klopt het niet helemaal. De geheugenplaatsen CNTH en CNTL bevatten een getal dat samenhangt met de **bittijd T** en de geheugenplaatsen CNTHH en CNTHL een getal dat samenhangt met **de halve bittijd ½T**. Waarom? Zie het nu volgende punt 4.
(N.B. Wordt vervolgd in punt 6).

4 Voordat we verder gaan met de rest van de voorroutine van PM, dus met de rest van figuur 1b, volgt er nu eerst een intermezzo; er wordt een aantal elementaire subroutines besproken. Allereerst de subroutines **DELHBIT** en **DELBIT** van figuur 4. We gaan ervan uit dat de bittijd T vastligt via de inhoud van de geheugenplaatsen CNTH en CNTL en dat de halve bittijd ½T vastligt via de inhoud van de geheugenplaatsen CNTHH en CNTHL; zie hiertoe punt 3. De subroutines DELHBIT en DELBIT zorgen ervoor dat er een halve bittijd ½T respektievelijk een hele bittijd T wordt gewacht. De tijd ½T of T wordt als volgt besteed:
Zolang de BCS na het label CNTDN (Count down; aftellen) in figuur 4 op springen staat wordt het getal, gevormd door (TIMH, TIML), periodiek verlaagd. De beginwaarde van (TIMH, TIML) is gelijk aan de eindwaarde van het getal (CNTH, CNTL) na afloop van COMTIM (ingang DELBIT), òf gelijk aan de door twee gedeelde eindwaarde van (CNTH, CNTL), dus gelijk aan het getal (CNTHH, CNTHL) (ingang DELHBIT). Het periodiek met één verlagen gaat net zo lang door totdat de BCS niet meer op springen staat, dus totdat C nul wordt en dus de borrow = $\overline{C}$ één wordt. De inhoud van (TIMH, TIML) is dan negatief: het getal FFFE. **Dit getal is gelijk aan de beginwaarde van (CNTH, CNTL) aan het begin van de subroutine COMTIM** (zie punt 2). Met andere woorden: **indien we er voor zorgen dat het doorlopen van het programmadeel vanaf het label CNTDN tot en met de BCS net zo lang duurt als het doorlopen van het programmadeel vanaf het label COMTIM** (figuur 3) **tot en met de BPL, dan wordt er**



Figuur 4. De subroutines DELHBIT en DELBIT zorgen ervoor dat er een halve bittijd ½T respektievelijk een hele bittijd T wordt gewacht.

automatisch een halve bittijd ½T respektievelijk een hele bittijd T gewacht. In figuur 3, in COMTIM, duurt het 29 µs voordat het programma vanaf het label COMTIM tot en met de BPL (springen naar een adres op dezelfde pagina) één keer is "afgelegd". Zonder de twee NOPs duurt het in figuur 4 25 µs voordat het programma vanaf het label CNTDN tot en met de BCS (springen naar een adres op dezelfde pagina) één keer is doorlopen. Mèt die twee NOPs is dat óók 29 µs. Vandaar de toevoeging van die twee schijnbaar nutteloze instrukties.
De eerder (zie punt 2) genoemde eindstand (CNTH, CNTL) = ØØ1B betekent dat, vanuit de beginpositie FFFE, (CNTH, CNTL) 29 keer met één is verlaagd. Dat heeft 29 x 29 = 841 µs geduurd. Verder kost de aanroep van de subroutine COMTIM ook nog zo'n 6 µs en wordt het begin van een startbit hoogstwaarschijnlijk niet op exakt hetzelfde moment ontdekt

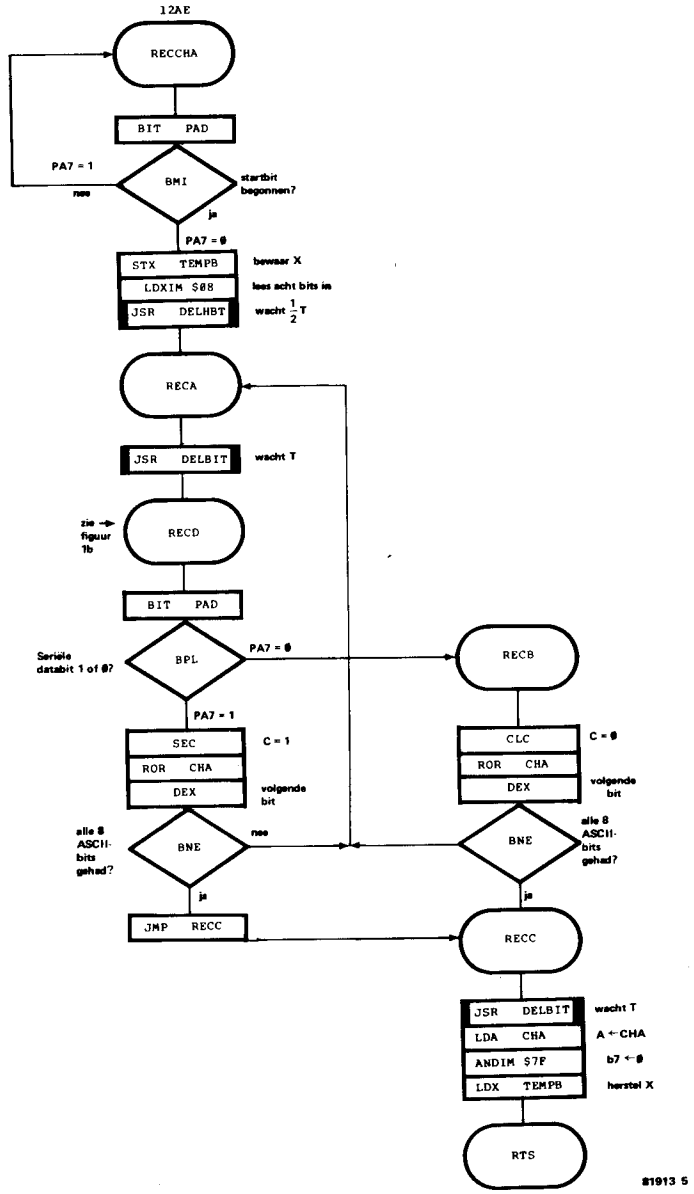
(onderbreking wachtlus met BIT-PAD, bovenaan in figuur 1b). We kunnen dus uitgaan van een vastgestelde bittijd van om en nabij de 850 μs. Aan de andere kant is er sprake van een baudrate van 1200. Dus 1200 bits per sekonde. Daarmee komt een bittijd overeen van 833 μs. We zitten er dus niet zo gek ver naast en gelukkig hoeft PM in dit opzicht nou ook weer niet zo'n Pietje Precies te zijn.

5 Het tweede intermezzo (na het eerste van punt 4) betreft de bespreking van de **subroutine RECCHA**. We treffen hem aan in figuur 5. Deze subroutine houdt zich bezig met het vaststellen van een naar de junior-computer verzonden ASCII-karakter en zet de bits van dat karakter in de juiste volgorde in de accu. Een naar de computer verzonden karakter komt overeen met het indrukken van een software-toets van het ASCII-toetsenbord. Omdat er sprake is van een hardware-echo (zie hoofdstuk 12) uit het verzonden karakter zich eveneens op het tv-scherm of printerpapier.
Het begint in figuur 5 met een wachtlus: BIT-PAD plus BMI. Pas nadat het begin van een startbit is vastgesteld (PA7 = Ø) gaan we verder. Eerst wordt de waarde van het X-register opgeslagen in TEMPB. Rechtsonder in figuur 5, vlak voor de RTS, zien we dat het X-register zijn oorspronkelijke inhoud terug krijgt. **Dus tijdens RECCHA gaat de inhoud van het X-register niet verloren.** Tijdens RECCHA doet X dienst als bitteller; hij krijgt de beginwaarde Ø8. Nadat dit is gebeurd wordt er een halve bittijd gewacht en vervolgens, na het label RECA, nog eens een hele bittijd. Dit heeft tot gevolgd dat we kwa tijd middenin de periode zitten tijdens welke het eerste ASCII-bit bØ wordt verzonden. Zie figuur 2. Via BIT-PAD en de aansluitende BPL wordt er gekeken of het bit nul of één is en afhankelijk daarvan gehandeld. De carryvlag wordt gelijk gemaakt aan de bitwaarde. Van links uit wordt die carry via een ROR de geheugenplaats CHA ingeschoven. Alle bits van CHA schuiven een plaatsje naar rechts op.
Voor CHA geldt de volgende toestand:

Na X = Ø8:	bØ	X	X	X	X	X	X	X
Na X = Ø7:	b1	bØ	X	X	X	X	X	X
Na X = Ø6:	b2	b1	bØ	X	X	X	X	X
Na X = Ø5:	b3	b2	b1	bØ	X	X	X	X
Na X = Ø4:	b4	b3	b2	b1	bØ	X	X	X
Na X = Ø3:	b5	b4	b3	b2	b1	bØ	X	X
Na X = Ø2:	b6	b5	b4	b3	b2	b1	bØ	X
Na X = Ø1:	b7	b6	b5	b4	b3	b2	b1	bØ

En alle bits staan netjes op hun plaats. In CHA. Rest nog de afwerking. Na het label RECC wordt nog een bittijd T gewacht. Dan wordt het eerste stopbit verzonden. De inhoud van CHA wordt gekopieerd in de accu en met behulp van de instruktie AND #7F wordt het bit b7 nul gemaakt. Dat bit was het door het toetsenbord, of juister: het door de UART verzonden pariteitsbit. In hoofdstuk 12 heeft u kunnen lezen dat we daarin niet zijn geïnteresseerd.
Het label RECD in figuur 5 is schijnbaar voor niets aanwezig. Schijnbaar, omdat er naar wordt gesprongen vanuit de voorroutine (zie figuur 1b en punt 6). In deze fase is het startbit al vastgesteld. Verder moet X vóór de sprong naar RECD de waarde Ø8 toegekend krijgen.

★ 5



Figuur 5. De subroutine RECCHA houdt zich bezig met het vaststellen van een aan de junior-computer verzonden ASCII-karakter, dus met het vaststellen van een ingedrukte toets op het ASCII-toetsenbord.

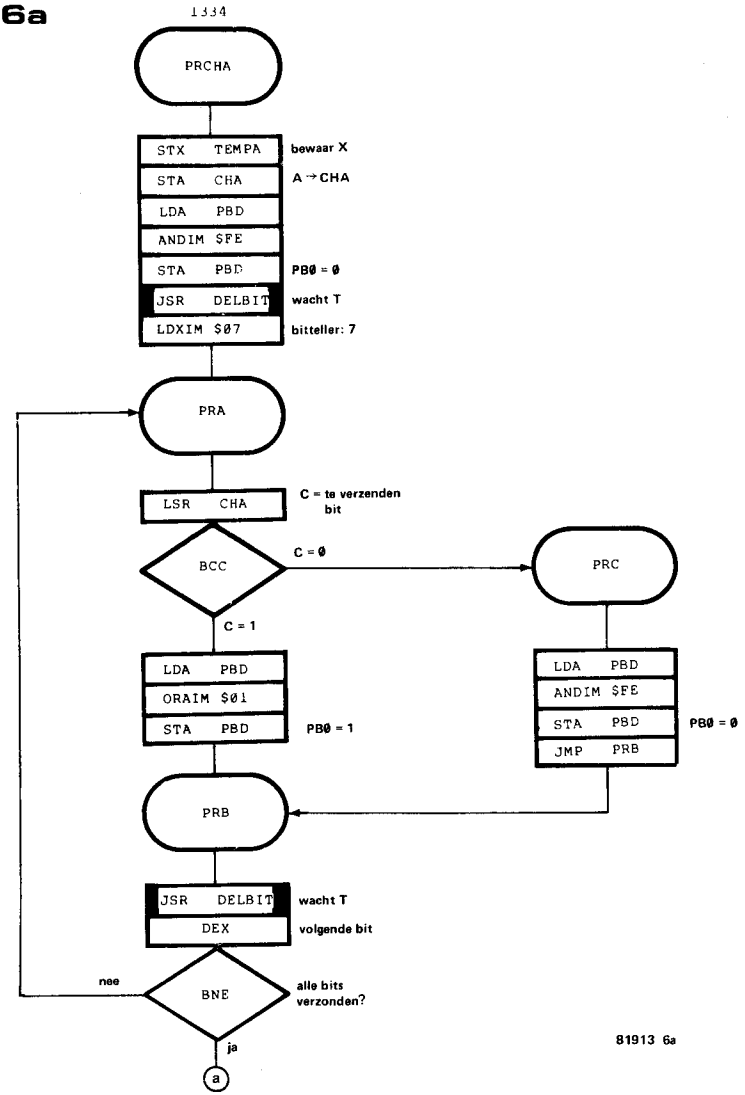
6 (vervolg van punt 3). We bespreken nu de rest van de tweede helft van de voorroutine van PM. Zie figuur 1b. Nadat het startbit achter de rug is en nadat de diverse bittijd-geheugenplaatsen de juiste inhoud hebben gekregen krijgt X de waarde Ø8, wachten we een halve bittijd ½T en springen we naar dat gedeelte van de subroutine RECCHA dat zich bezig houdt met het inlezen van alles van het ASCII-karakter na het startbit (zie figuur 5 en punt 5). Is dat allemaal achter de rug, dat wordt er gekeken of dat ASCII-karakter soms de kode 7F had. Dus of de toets RES = RUB OUT is ingedrukt, ten teken van het afwerken van de tweede startfase van PM. Is dat het geval, dan gaan we verder met het label LABJUN. Zoniet: terug naar AF, naar INITPR (figuur 1a).
Het afwerken van alles na het label LABJUN houdt in dat er wordt gezorgd voor het afdrukken van de boodschap "JUNIOR" en dat al het toekomstige, door PM verzorgde drukwerk plaatsvindt vanaf het begin van een nieuwe regel (CRLF). De beide subroutines JUNIOR en CRLF worden later besproken (zie de punten 11 en 12).
Indien we bedenken dat er óók naar LABJUN wordt gesprongen na het indrukken van die grafische noodrem: de BRK-toets, dan kan men zich voorstellen dat er situaties optreden waarbij de stapel "overloopt", met alle nare gevolgen van dien. Vandaar dat de stack pointer en SPUSER beide op hun nummer (FF) worden gezet. Nadat dit is gedaan is de voorroutine van PM doorlopen; we zijn aangekomen bij het label RESALL. Dat is het centrale punt van de hoofdroutine van PM.

7 Een tweede voorroutine van PM is de voorroutine STEP van figuur 1b. Deze wordt doorlopen tijdens de terugkeer naar PM na de uitvoering van één instruktie in het geval van het stap voor stap doorlopen van een programma. Dat is óók in het algemeen het geval nadat er een NMI heeft plaatsgevonden. De NMI-sprongvektor is tijdens de voorroutine INITPR gespecificeerd op het adres van het label STEP; zie figuur 1a.
De voorroutine STEP lijkt als twee druppels water op de SAVE-routine van de standaard-monitor. Alle registers worden bewaard in de bekende geheugenplaatsen (zie de hoofdstukken 3 en 7). Ter afsluiting volgt de sprong naar de subroutine PRBUFS. Deze wordt later (zie punt 10) in detail besproken. Nu is het voldoende om te weten dat het adres (PCH, PCL) op het begin van een nieuwe regel wordt afgedrukt, gevolgd door een spatie en de inhoud van de bij dat adres horende geheugenplaats. Zoals zal blijken: werkadres plus werkdata. In het stap-voor-stapgeval is de inhoud van die geheugenplaats de opcode van de volgende uit te voeren instruktie. (Deze instruktie wordt uitgevoerd via het indrukken van toets R).
Men kan de STEP-ingang van PM eveneens gebruiken om naar te springen na de instruktie BRK of na een hardware-IRQ. In dat geval moet men de IRQ-sprongvektor zelf specificeren.

De voorroutines van PM zijn nu behandeld. Verder zijn er al enige PM-subroutines de revue gepasseerd. Voordat we overgaan tot de bespreking van de hoofdroutine van PM komen er eerst nog enige andere PM-subroutines aan de orde. Ze hebben allemaal iets te maken met het door de junior-computer af te leveren drukwerk.

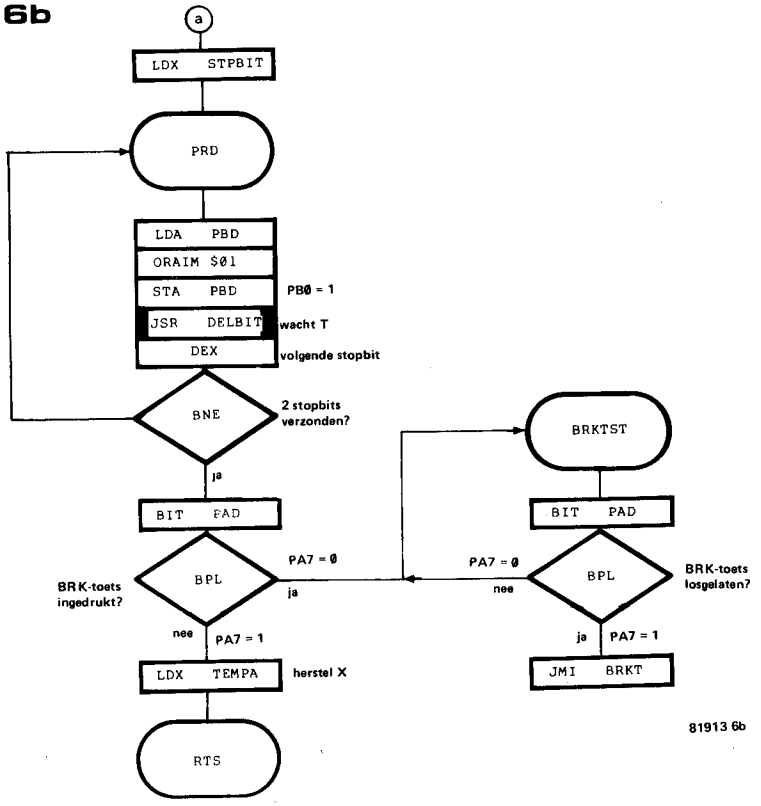
8 Zoals RECCHA de elementaire ontvang-subroutine is, zo is **PRCHA de elementaire verzend-subroutine.** Het stroomdiagram staat in de figuren 6a en 6b. Deze subroutine zorgt ervoor dat een ASCII-karakter, waarvan de kode in de accu staat, wordt afgedrukt.
Het begint met het bewaren van de inhoud van het X-register in de geheugenplaats TEMPA. Vlak voor de RTS (figuur 6b) krijgt X de oude

★ 6a



inhoud terug. **Dus uiteindelijk blijft het X-register tijdens PRCHA onaangetast.** De geheugenplaats CHA speelt net zoals bij RECCHA ook nu een rol bij de behandeling van de bits; de inhoud van de accu wordt gekopieerd in CHA. Vervolgens maken we via het maskeren van bit nul (de instruktie AND #FE, dus AND #1111.111Ø) PBØ nul en wachten een bittijdje T: het startbit is nu verzonden. Daarna maken we X, die als bitteller dienst doet, gelijk aan Ø7.
Nu krijgt de carryvlag de waarde van het te verzenden bit (direkt na het label PRA in figuur 6a). Dat gebeurt door zeven keer de instruktie LSR-CHA uit te voeren. De poortlijn PBØ krijgt een met de bitwaarde (= C) overeenkomende waarde; de andere poortlijnen PB1 . . . PB7 blijven ongemoeid.

★ 6b



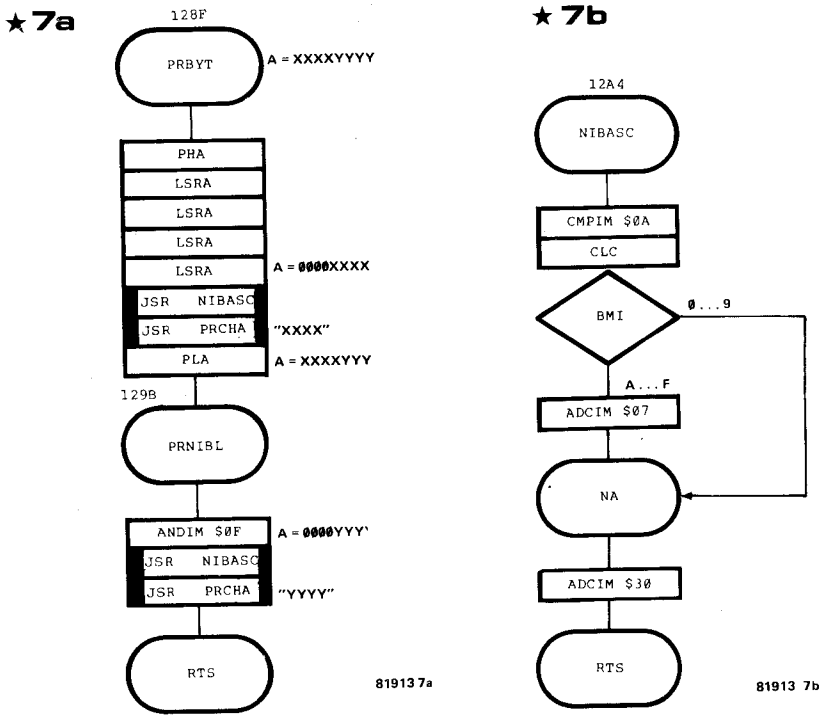
Figuur 6. De elementaire drukwerk-subroutine PRCHA is er voor het afdrukken van een ASCII-karakter (waarvan de kode in de accu staat) op het tv-scherm of printerpapier. Het stroomdiagram omvat de figuren 6a en 6b.

De gang van zaken is als volgt:

Begin PRCHA:	Ø	b6	b5	b4	b3	b2	b1	bØ	C = X
Na X = Ø7:		Ø	b6	b5	b4	b3	b2	b1	C = bØ
Na X = Ø6:			Ø	b6	b5	b4	b3	b2	C = b1
Na X = Ø5:				Ø	b6	b5	b4	b3	C = b2
Na X = Ø4:					Ø	b6	b5	b4	C = b3
Na X = Ø3:						Ø	b6	b5	C = b4
Na X = Ø2:							Ø	b6	C = b5
Na X = Ø1:								Ø	C = b6

We zien dat bit zeven vóór de seriële dataverzending nul is en niet wordt verzonden. Dat klopt, want in PM doen we niet aan een pariteitsbit; alleen de zeven ASCII-bits worden verzonden.
We zijn toe aan figuur 6b, en aan de verzending van twee afsluitende stopbits. Dat aantal, 2, staat in de geheugenplaats STPBIT (zie figuur 1a) en wordt nu in X gezet. Vervolgens wordt er twee keer achter elkaar een bittijd T gewacht, waarbij PBØ de waarde nul heeft.
De eigenlijke verzending van het ASCII-karakter is nu voltooid en wat er vervolgens in figuur 6b gebeurt heeft te maken met de BRK-toets. U weet van hoofdstuk 12 dat we het afdrukken van ASCII-karakters door de junior-computer konden onderbreken door de BRK-toets in te drukken. Nadat het aan de beurt zijnde ASCII-karakter volledig is verzonden gaan we dan ook kijken of die BRK-toets is ingedrukt. Is dat niet het geval, dan zijn we op het herstellen van X na klaar. Is dat wèl het geval, dan wordt er net zolang gewacht totdat de BRK-toets weer is losgelaten. Als het zo ver is dan volgt een Jump-indirekt (in figuur 6b aangegeven met de niet-officiële opcode JMI). Het effektieve sprongadres (BRK-sprongvektor) is gespecificeerd in de geheugenplaatsen BRKT ($1A7C) en BRKT+1 ($1A7D). Dit effektieve sprongadres is gelijk aan het adres van het label LABJUN. Zie figuur 1a. Na het indrukken – èn het loslaten! – van de BRK-toets volgt de terugmelding "JUNIOR".

9 De **subroutine PRBYT** houdt zich bezig met het **afdrukken van achtbits data, die in de accu staat.** In figuur 7a is-ie te zien. De data wordt verzonden in de vorm van twee ASCII-karakters; eentje voor het linker nibble en eentje voor het rechter nibble. Dat betekent dat elk nibble vóór de verzending eerst in de bijbehorende ASCII-kode moet worden vertaald. (Data 2Ø bijvoorbeeld zou zonder omzetting in de ASCII-kodes 32 resp. 3Ø worden behandeld als de ASCII-kode van een spatie). Daarom komt nu eerst de **subroutine NIBASC** van figuur 7b aan de orde.
Voor datanibbles ØØ . . . Ø9 (hexadecimale en decimale numerieke toetsen) is de ASCII-kode gelijk aan $3Ø . . . $39. Voor datanibbles ØA . . . ØF (hexadecimale numerieke toetsen) is de ASCII-kode gelijk aan $41 . . . $46. Zie Aanhangsel 7 op pagina 215 van boek 3. Gaan we ervan uit dat aan het begin van NIBASC de inhoud van de accu gelijk is aan ØX, met X = Ø . . . F, dan moet in het ene geval $3Ø bij de accu-inhoud worden opgeteld om de ASCII-kode te krijgen (Ø . . . 9), en in het andere geval (A . . . F) moet er $37 bij de accu-inhoud worden opgeteld om de juiste ASCII-kode te realiseren. Dat is precies wat er gebeurt tijdens NIBASC. Via een CMP-immediate en een BMI worden de Ø . . . 9-bokken van de A . . . F-schapen



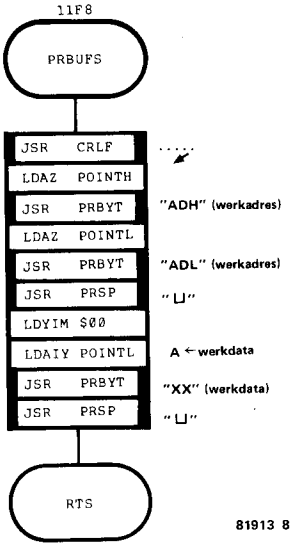
Figuur 7. De subroutine PRBYT (figuur 7a) verzorgt het afdrukken van een databyte in de vorm van twee datanibbles. Elk datanibble moet eerst worden omgezet in de overeenkomende ASCII-kode, alvorens te worden verzonden. Daartoe dient de subroutine NIBASC van figuur 7b.

gescheiden. Zowel bij de bokken als bij de schapen wordt $3Ø opgeteld; bij de schapen wordt vooraf eerst nog $Ø7 opgeteld.
Terug naar de subroutine PRBYT. Dus naar figuur 7a. Die begint met het in de ijskast zetten van de accu-inhoud: PHA. Vervolgens gaan we vier keer achter elkaar de bits van A naar rechts schuiven. Het resultaat is dat het rechter nibble gelijk is aan het oorspronkelijke linker nibble en dat het linker nibble nu nul is.
Nu volgt de sprong naar NIBASC (figuur 7b); het linker nibble is vertaald in de bijpassende ASCII-kode en gaat vervolgens naar het tv-scherm of printerpapier: de subroutine PRCHA van punt 8 en van figuur 6a/6b. Daarna wordt de oorspronkelijke accu-inhoud weer uit de ijskast gehaald (PLA) en zijn we toe aan het label PRNIBL, teneinde het rechter accu-nibble af te drukken. Het label PRNIBL is vaak heel goed te gebruiken als het startadres voor een subroutine die slechts één nibble afdrukt (namelijk het rechter nibble). Door het maskeren van het linker nibble is de accu-inhoud voorbereid voor de nu volgende gang door de sub-

routines NIBASC en PRCHA: het rechter nibble wordt afgedrukt en we zijn klaar. Vandaar de RTS.

10 **De subroutine PRBUFS** van figuur 8 houdt zich bezig met het afdrukken van wat we het **werkadres** hebben genoemd. De adresinformatie staat in de buffers POINTH (linker adresbyte) en POINTL (rechter adresbyte). Dàt zijn oude bekenden! Tevens wordt de inhoud van de geheugenplaats (POINTH, POINTL) afgedrukt. Dus de **werkdata**. Het verhaal van en over figuur 8 is gauw verteld. Er wordt altijd op het begin van een nieuwe regel begonnen (subroutine CRLF, zie punt 12). Daarna wordt het linker byte ADH van het werkadres afgedrukt via de subroutine PRBYT, die we zojuist, in punt 9, hebben leren kennen. Vervolgens van hetzelfde laken een pak, maar dan voor het rechter byte ADL van het werkadres. Dan een spatie, ter wille van de grafische vormgeving (subroutine PRSP, zie punt 12). Rest nog het naar de accu halen van de geheugeninhoud van het adres (POINTH, POINTL) via LDA-(POINTL),Y, met Y gelijk aan nul. En tenslotte het zichtbaar maken van de twee datanibbles (PRBYT) van deze **werkdata** en het nogmaals "afdrukken" van een spatie.

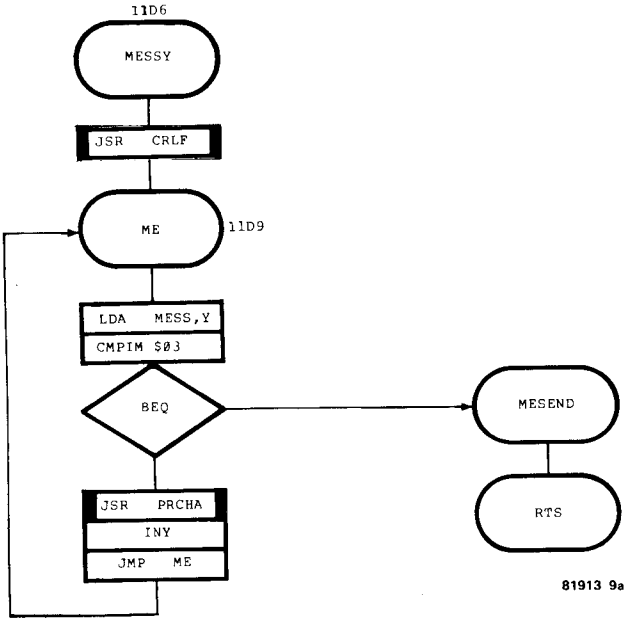
★ 8



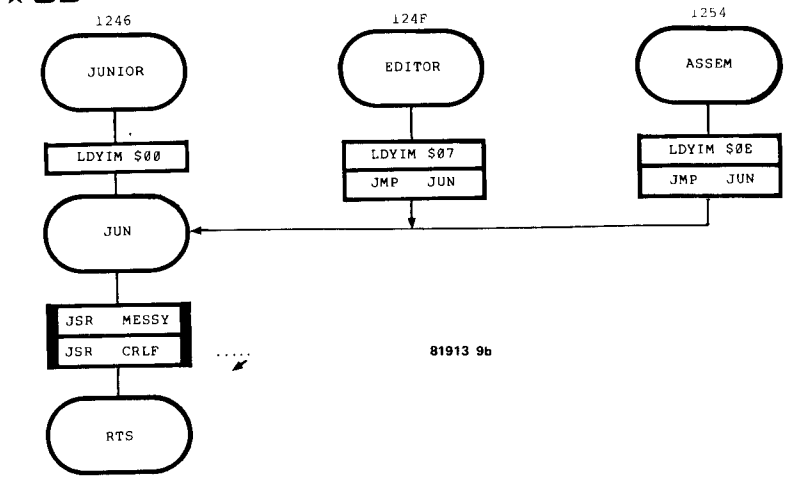
Figuur 8. De subroutine PRBUFS, voor het afdrukken van een werkadres met de bijbehorende werkdata.

11 **De subroutine MESSY** van figuur 9a is er voor het afdrukken van een stukje tekst ("text string"). Van een boodschap dus. Bijvoorbeeld een terugmelding of een foutmelding. Denk aan "JUNIOR" of "WHAT?", maar ook aan "SA:". De af te drukken tekst is, in de ASCII-kode vertaald, aanwezig in een aantal opeenvolgende geheugenplaatsen van

★9a



★9b



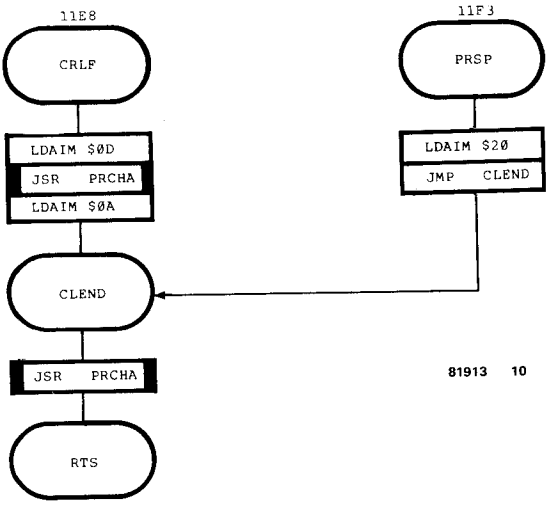
Figuur 9. De subroutine MESSY (figuur 9a) verzorgt het afdrukken door de junior-computer van een terugmelding; in sommige gevallen heeft die terugmelding het karakter van een foutmelding. De subroutines JUNIOR, EDITOR en ASSEM van figuur 9b leveren, onder gebruikmaking van MESSY, de terugmelding "JUNIOR", "EDITOR" of "ASSEMBLER" op. De laatste twee terugmeldingen worden bij PM niet toegepast.

de opzoektabel MESS, die de adressen 13BD tot en met 141D omvat (zie de listing achter in dit boek). Voor elke ASCII-kode één geheugenplaats. Het aantal plaatsen dat een bepaalde geheugenplaats van de opzoektabel van het eerste adres 13BD verwijderd is leggen we vast via de waarde van de index Y. Een tekst is afgesloten met een EOT-teken (ASCII-kode $Ø3). De gang van zaken is dat men met een Y begint die overeenkomt met het eerste tekstteken van de boodschap en dat men net zo lang doorgaat (via het verhogen van Y) totdat een einde-tekstteken EOT is vastgesteld.
De subroutine MESSY (figuur 9a) begint met een andere subroutine: CRLF (zie punt 12). Er wordt begonnen op het begin van een nieuwe regel. Daarna (label ME) wordt via de instruktie LDA-MESS,Y een af te drukken tekstteken de accu binnengehaald. De beginwaarde van Y moet vóór de sprong naar MESSY zijn vastgelegd. Er wordt vervolgens gekeken of er geen EOT-teken is binnengehaald. Want dan zijn we klaar (RTS). Zoniet, dan volgt het afdrukken van het zojuist opgehaalde tekstteken (PRCHA) en verhogen we Y met één; het spel kan opnieuw beginnen (JMP-ME). In sommige gevallen wordt alles via MESSY vanaf het label ME als subroutine gebruikt. Er wordt dan niet op een nieuwe regel begonnen.
Alvast een voorbeeld van een toepassing van MESSY. **De subroutines JUNIOR/EDITOR/ASSEM** van figuur 9b. Er zijn nog veel meer toepassingen; zie verderop in dit hoofdstuk. U ziet dat eerst Y een bepaalde waarde krijgt toegekend. Dan volgt het beroep op MESSY ("JUNIOR", "EDITOR" of "ASSEMBLER") en tot slot zorgt een CRLF ervoor dat er na een boodschap altijd (te zijner tijd) nieuw drukwerk volgt vanaf het begin van de volgende regel. N.B. De subroutine EDITOR en ASSEM worden niet gebruikt bij PM.

12 **De subroutines CRLF en PRSP** van figuur 10 worden buitengewoon vaak toegepast. Dat is niet verwonderlijk, want het gaat om grafische subroutines, die iets te maken hebben met de vormgeving en de overzichtelijkheid van het af te lezen drukwerk. De bespreking van de subroutines is een fluitje van een cent.
Het is een kwestie van het laden van de accu met de ASCII-kode van het "af te drukken" grafische kommando en vervolgens het "afdrukken" van dat kommando (PRCHA). Voor het kommando CR (Carriage Return; terug naar het begin van **dezelfde** regel) geldt de ASCII-kode ØA en voor het kommando LF (Line Feed; ga door naar **dezelfde** positie op de volgende regel) geldt een ASCII-kode ØA. Het afdrukken van een spatie (PRSP) gaat ook al heel eenvoudig: laad de accu met de ASCII-kode, 2Ø, en spring naar het label CLEND, dus druk die spatie af.

13 **De subroutine HEXNUM** (figuur 11) speelt een sleutelrol bij het vaststellen of een naar de junior-computer verzonden ASCII-karakter hoort bij een ingedrukte numerieke toets of niet. Gaat het inderdaad om de opgave, via het toetsenbord, van numerieke data, dan wordt deze data na het vertalen uit de ASCII-kode verwerkt in de inhoud van de **databuffers INL en INH**. Is dat echter niet het geval, dan volgt de terugmelding "WHAT?" (vrij vertaald: "wat krijgen we nou!?). Voordat we HEXNUM in detail gaan bespreken volgt het verhaal over **de subroutine**

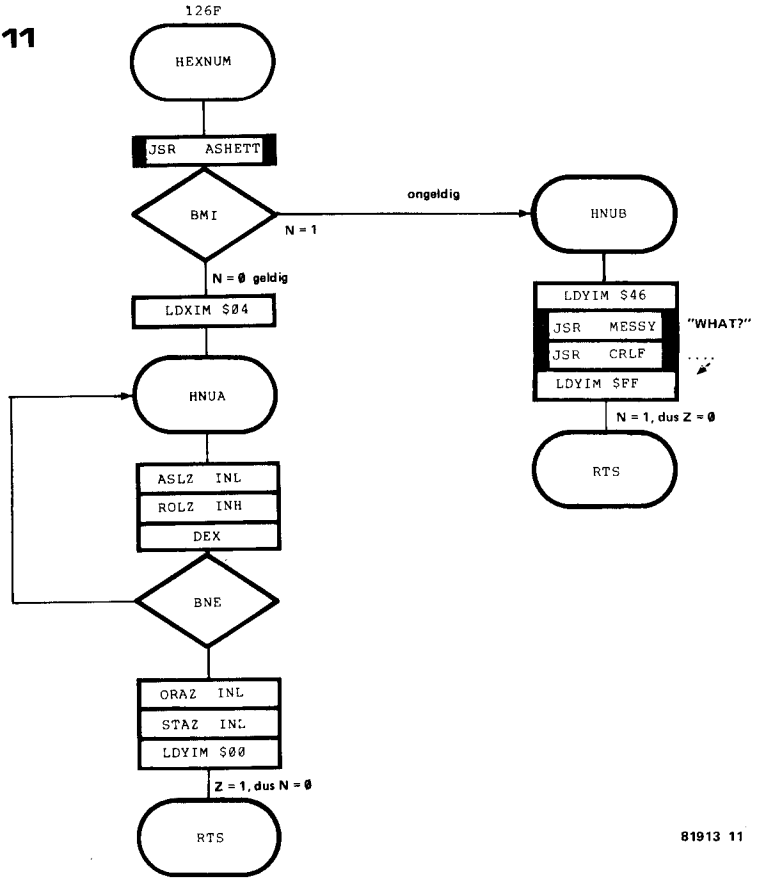
★10



Figuur 10. De subroutines CRLF (toekomstig drukwerk vanaf het begin van de volgende regel) en PRSP (druk een spatie, dus niets af). Het effekt van PRSP is de verplaatsing, één positie naar rechts of naar het begin van de volgende regel, van de "schrijfwijzer" ("cursor"), respektievelijk de wagen ("carriage").

ASHETT van figuur 12, die vaststelt of we wel met numerieke data hebben te doen en zo ja uit de ASCII-kode de data destilleert.
Dus nu eerst figuur 12. We gaan ervan uit dat de ASCII-kode in de accu staat. Er geldt dat ASCII-kodes 3Ø . . . 39 (toetsen Ø . . . 9) en ASCII-kodes 41 . . . 46 (toetsen A . . . F) geldig zijn; alle andere ASCII-kodes horen bij iets dat als een ongeldig karakter zal worden beschouwd. In ASHETT vindt de splitsing tussen geldig en ongeldig plaats via vier, elkaar opeenvolgende compare-instrukties en vier daarop aansluitende instrukties BMI. Er ontstaat een soort filterwerking. Nadat een ongeldig karakter (dus een niet-numeriek karakter) is vastgesteld volgt een RTS, waarbij de N-vlag één is en dus de Z-vlag nul. De ASCII-kode van een geldig karakter moet eerst nog worden vertaald in het overeenkomende datanibble. Dat gebeurt in ASHETT na het label VALIT.
Voor data ØØ . . . Ø9 kan men het vertalen uitvoeren door domweg het linker nibble van de accu-inhoud nul te maken (AND #ØF). Voor data ØA . . . ØF moet men echter eerst nog Ø9 bij de accu-inhoud optellen. Omdat na de terugkeer uit ASHETT bij een geldig karakter het linker nibble van de accu-inhoud nul is en het rechter nibble altijd ongelijk aan nul, zijn zowel de N-vlag als de Z-vlag tijdens de afsluitende RTS nul.
En dan nu HEXNUM. Figuur 11 dus. Die begint met ASHETT (figuur 12) en vervolgens een BMI. Een ongeldig, niet numeriek ASCII-karakter voert ons naar het label HNUM. Er gebeuren dan allerlei dingen die te maken hebben met de foutmelding "WHAT?". Eerst krijgt Y de passende waarde 46. Vervolgens wordt er een beroep gedaan op de subroutine MESSY (zie

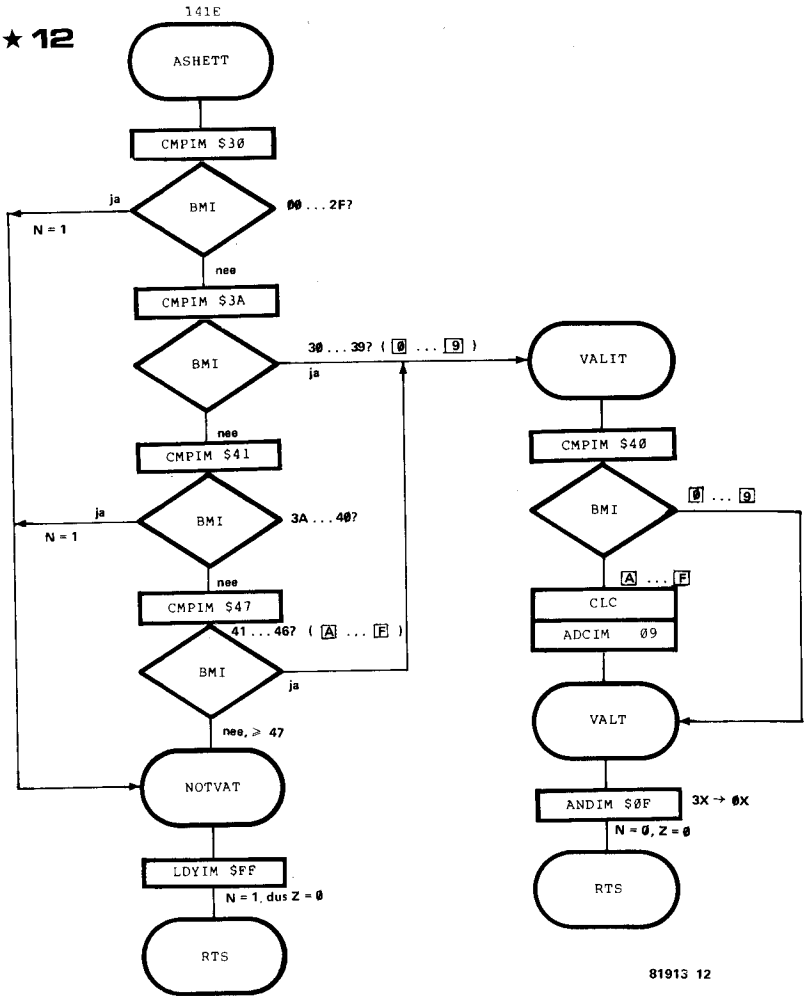
★ 11



Figuur 11. De subroutine HEXNUM, voor het verwerken van een met een ingedrukte numerieke toets overeenkomend datanibble in de inhoud van de databuffers INH en INL.

punt 11 en figuur 9a). Na "WHAT?" komt nieuw drukwerk, te zijner tijd, te staan op het begin van de volgende regel (CRLF). Vóór de RTS krijgt Y de waarde FF. De N-vlag is dus één en de Z-vlag nul.
Als er een geldig, dus numeriek ASCII-karakter is vastgesteld wordt het overeenkomstige datanibble "bijgezet" in de databuffers INL en INH. Dat gaat via het vier keer achter elkaar naar links schuiven en linksom draaien van de bits van INL en INH. De manier waarop dat gebeurt is identiek aan soortgelijke operaties in de software van de standaard-EPROM. Zie boek 2. Daarom volstaan we hier en nu met een korte samenvatting:

- Het rechter nibble van INL is gelijk aan het zojuist ontvangen datanibble.
- Het linker nibble van INL is gelijk aan het voormalige rechter nibble van INL.



Figuur 12. De subroutine ASHETT wordt in HEXNUM (figuur 11) aangeroepen. Deze gaat na of er sprake is van een ingedrukte toets 0 . . . F of niet. In het laatste geval leidt dit in HEXNUM tot de terugmelding "WHAT?".

- Het rechter nibble van INH is gelijk aan het voormalige linker nibble van INL.
- Het linker nibble van INH is gelijk aan het voormalige rechter nibble van INH.
- Het voormalige linker nibble van INH is weg (in groenteveilingtermen: is "doorgedraaid").

Met andere woorden: de inhoud van INH en INL is altijd in overeenstemming met de laatste vier ingedrukte numerieke toetsen, tenzij de buffers tussentijds zijn gereset (subroutine RESIN, zie punt 14). Indien HEXNUM minder dan vier keer achter elkaar wordt aangeroepen, dus bijvoorbeeld x keer, dan zijn de linker (4-x) nibbles van (INH, INL) nul.
Vlak voor de afsluitende RTS (linksonder in figuur 11) krijgt Y de waarde ØØ toegekend. Dit heeft tot gevolg dat de Z-vlag één wordt en de N-vlag nul; precies het tegenovergestelde van de situatie na een ongeldig ASCII-karakter (de rechter RTS in figuur 11).

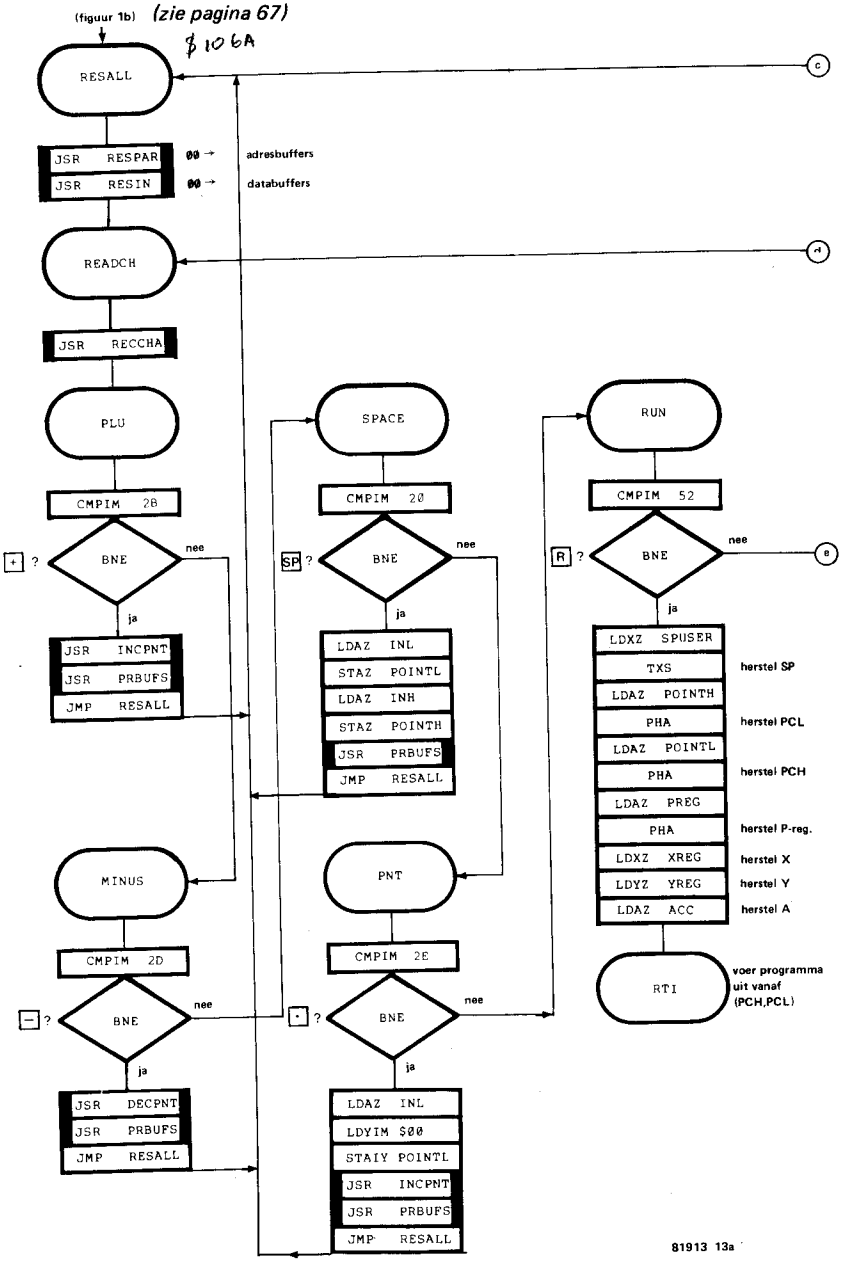
14 **De hoofdroutine van PM** en de resterende PM-subroutines zullen nu worden besproken. Het gedetailleerde stroomdiagram van de PM-hoofdroutine staat in de figuren 13a, 13b en 13c. Het centrale punt is het label RESALL, linksboven in figuur 13a. Na dit label worden **de subroutines RESPAR** (figuur 14a) **en RESIN** (figuur 14b) uitgevoerd. In RESPAR worden de twee **adresbuffers PARA en PARB** (beide 16 bits) nul gemaakt. In RESIN gebeurt hetzelfde, maar dan voor de **databuffers INH en INL**. Dit nul maken vindt plaats zodra de inhoud van deze buffers niet meer nodig is, na het afwerken van een bepaalde toetsaktie. Het nu volgende label READCH is óók een soort centraal punt van de PM-hoofdroutine, namelijk in het geval van dataverwerking (bijvoorbeeld de opgave van een werkadres of werkdata). Direkt na het label READCH springt PM naar de subroutine RECCHA: er wordt (opnieuw) gewacht op een ingedrukte toets. Vervolgens houdt de PM-hoofdroutine zich bezig met het afchecken van de mogelijkheden: Is het toets X? Zo ja: voer de X-toetsroutine uit. Zo nee: is het soms toets Y? Zo ja: voer de Y-toetsroutine uit. Zo nee: Ga na of het soms toets Z is, enzovoorts.
N.B. De centrale labels RESALL (buffers nul maken) en READCH (vaststellen ingedrukte toets) worden niet tussentijds "bezocht" tijdens de uitvoering van een toetsroutine die hoort bij de toetsfunkties M, G en S. Bij deze toetsroutines maakt het wachten op een ingedrukte toets (opgave parameters) deel uit van die toetsroutine. Met andere woorden: de subroutine RECCHA wordt ook elders in PM gebruikt en dus niet uitsluitend na het label READCH.

15 Eerst maar eens de **plustoetsroutine.** Dus de instrukties van figuur 13a direkt na het label PLU. U weet van hoofdstuk 12 dat het werkadres met één moet worden verhoogd en dat dit nieuwe werkadres mèt data op een nieuwe regel moet worden afgedrukt. En dat gebéurt er ook. Eerst volgt de sprong naar de **subroutine INCPNT** van figuur 15a. Het werkadres wordt aangewezen door de adreswijzer (POINTH, POINTL). Het met één ophogen van deze adreswijzer (INCPNT, figuur 15a) gebeurt via een inmiddels wèlbekende metode. Na de verhoging van de adreswijzer volgt het afdrukken van het nieuwe werkadres plus data (subroutine PRBUFS; zie punt 10 en figuur 8). Rest de gang terug naar RESALL.

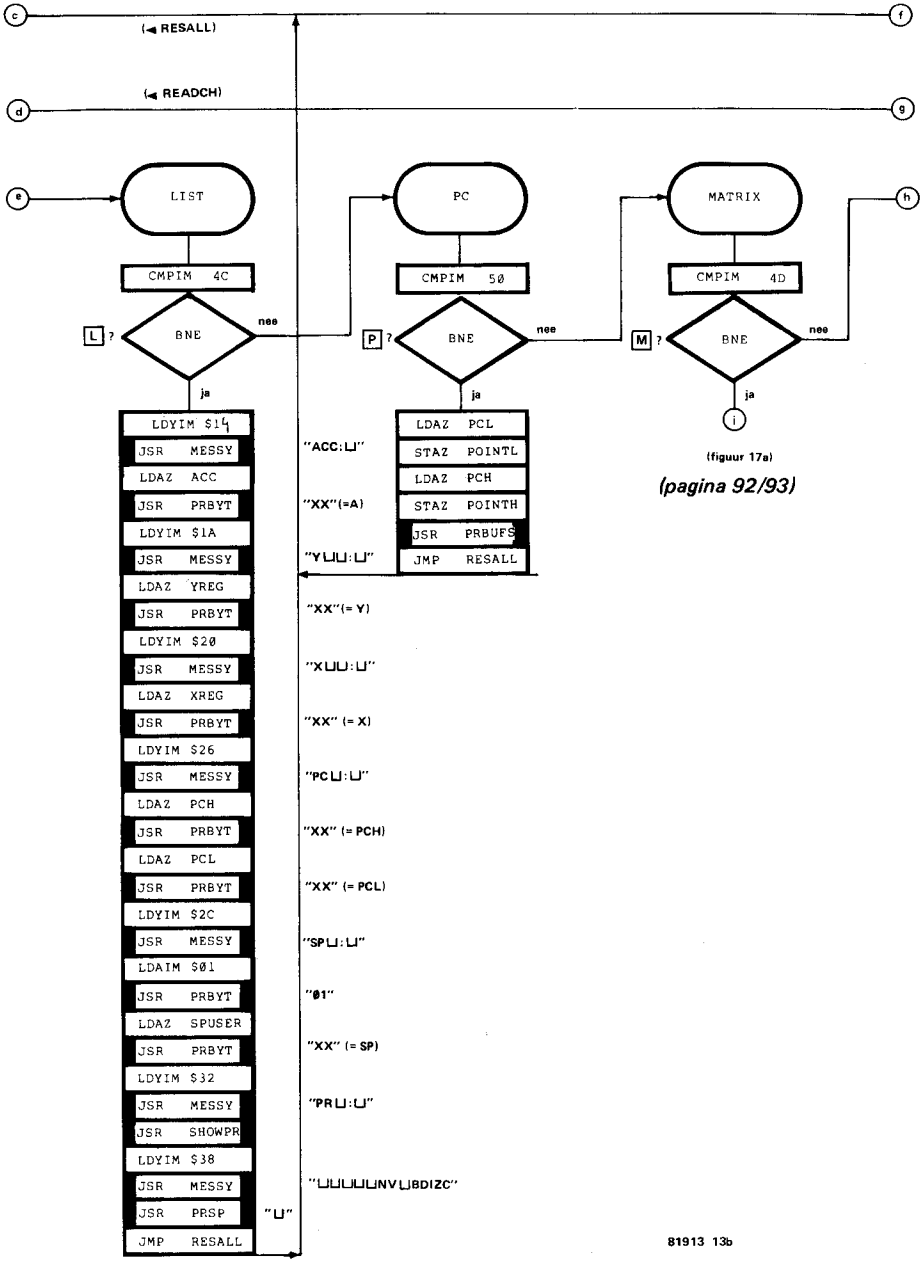
16 **Dan de mintoetsroutine.** Dat zijn de instrukties van figuur13a die direkt volgen op het label MINUS. Het werkadres moet met één worden verlaagd; het nieuwe werkadres moet mèt data op een nieuwe regel worden afgedrukt. Het gaat bijna net zo als bij de plustoetsroutine
lees verder op pagina 89 ➔

★ 13a

(figuur 1b) *(zie pagina 67)*



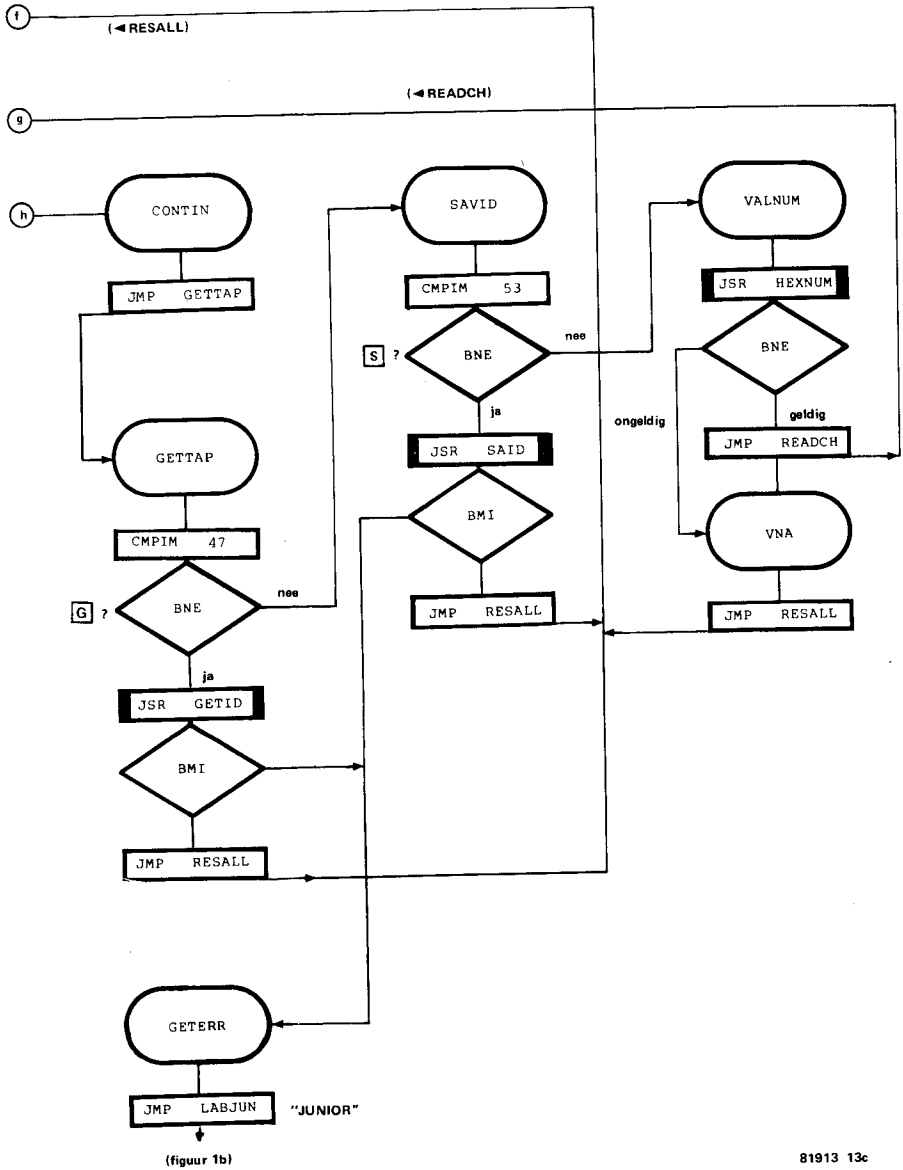
★ 13b



(figuur 17a)

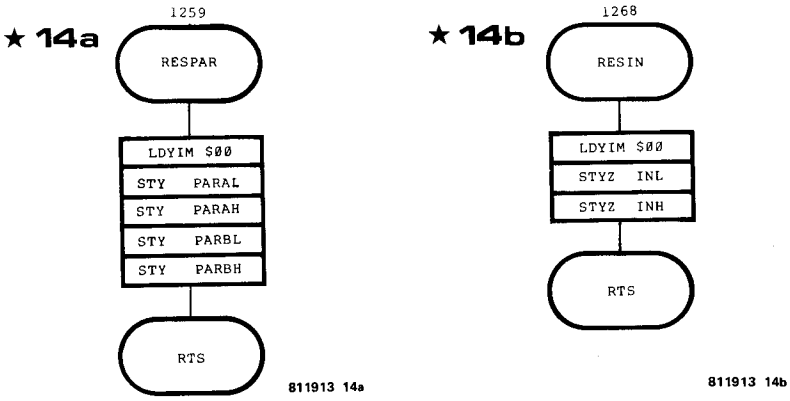
(pagina 92/93)

★ 13c

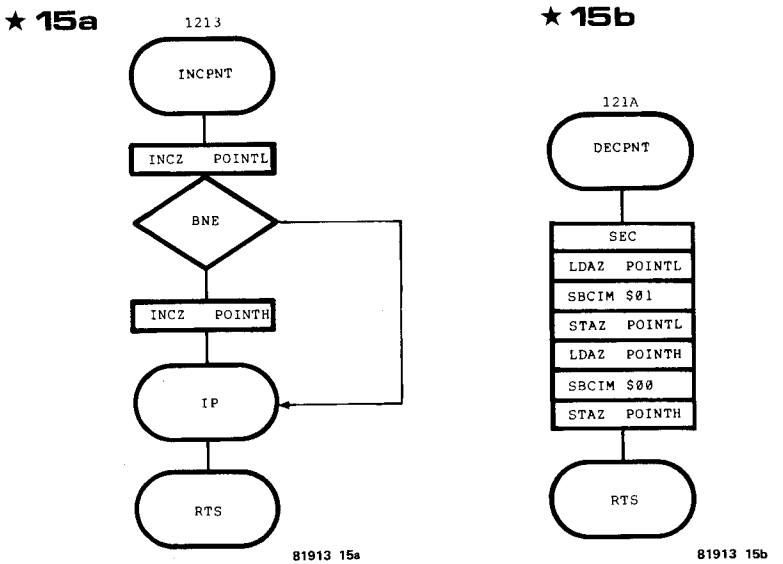


(pagina 67)

Figuur 13. De PM-hoofdroutine, verdeeld over de figuren 13a, 13b en 13c. In verband met het aantal instrukties is de M-toetsroutine in een aparte figuur, figuur 17a/17b, getekend.



Figuur 14. De subroutine RESPAR (figuur 14a) zorgt ervoor dat de inhoud van de adresbuffers PARAH en PARAL voor het eerste adres nul wordt, evenals de inhoud van de adresbuffers PARBH en PARBL voor het laatste adres. De subroutine RESIN (figuur 14b) levert als eindresultaat een inhoud nul op van de databuffers INH en INL.



Figuur 15. De subroutine INCPNT (figuur 15a) levert een verhoging met één op van de adreswijzer POINT; de subroutine DECPNT (figuur 15b) levert daarentegen een verlaging met één op.

van punt 15. Bijna, omdat nu niet INCPNT, maar de **subroutine DECPNT** van figuur 15b wordt aangeroepen: de adreswijzer (POINTH, POINTL) wordt nu met één verlaagd. Verder alles van hetzelfde laken een pak.

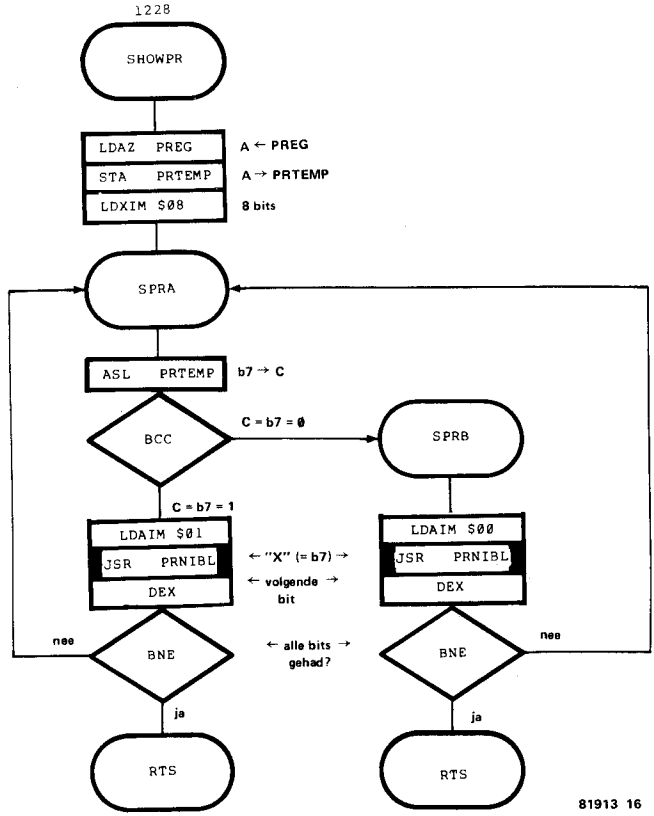
17 **De spatietoetsroutine** bestaat uit de instrukties van figuur 13a die direkt volgen op het label SPACE. Het gaat om de opgave van een werkadres, dat wordt afgedrukt. De adresdata is vooraf via het indrukken van één à vier numerieke toetsen opgegeven en staat in de databuffers INL en INH. Er wordt begonnen met het kopiëren van INL in POINTL (rechter werkadresbyte) en van INH in POINTH (linker werkadresbyte). Het werkadres ligt vast en kan worden afgedrukt; dus moet er een beroep worden gedaan op de subroutine PRBUFS van figuur 8 en punt 10. En dan zijn we klaar: terug naar RESALL.

18 **De punttoetsroutine** bestaat uit de instrukties van figuur 13a die direkt volgen op het label PNT. Er moet data worden geladen op een werkadres dat vooraf vastligt; hetzij via de spatietoets, hetzij via de plus- of mintoets. Het begint met het laden van de accu met de inhoud van de databuffer INL; INH is nu niet nodig omdat het om één databyte gaat, dus om nul (data ØØ!), één of twee achter elkaar ingedrukte numerieke toetsen. Drukt men achter elkaar meer dan twee numerieke toetsen in, dan geldt als werkdata de data die hoort bij de twee meest recente numerieke toetsen. Nadat de werkdata in de accu is gezet gaat de accuinhoud via de instruktie STA-(POINTL),Y naar de bij het werkadres horende geheugenplaats. Daarna verhogen we het werkadres met één (INCPNT) en drukken dit werkadres met data af (PRBUFS). Dit als voorbereiding voor de volgende opgave van werkdata. En dan zijn we klaar.

19 **De R-toetsroutine** (alle instrukties direkt na het label RUN in figuur 13a) is exakt gelijk aan de toetsroutine die hoort bij de GO-toets van de standaard-monitor. Zie figuur 4a op pagina 99 van boek 2. Alle registers krijgen de inhoud terug die ze hadden tijdens de meest recente sprong naar PM (STEP-ingang, zie figuur 1b). De afsluitende RTI van de R-toetsroutine zorgt ervoor dat er een programma wordt uitgevoerd waarvan het startadres gelijk is aan (PCH, PCL), óf dat, in het stap-voor-stapgeval, de volgende instruktie wordt uitgevoerd. De opcode van die instruktie bevindt zich op het adres (PCH, PCL). (PC: Program Counter: wàt staat er op het programma?)

20 Voordat het zover is dat de L-toetsroutine kan worden besproken gaan we het eerst hebben over een daarbij gebruikte speciale subroutine, namelijk de **subroutine SHOWPR** van figuur 16. Hij zorgt ervoor dat de inhoud van het P-register bit voor bit wordt afgedrukt.
Figuur 16 begint met het kopiëren van de geheugenplaats PREG (dat is het P-register: de vlaggenbewaarplaats) in de geheugenplaats PRTEMP. Verder krijgt het als bitteller gebruikte X-register de beginwaarde Ø8. De positie van de diverse vlaggen in PREG (èn in PRTEMP, althans in het begin) kan men vinden in het plaatje van het software-dashboard van de junior-computer; dat is figuur 1b op pagina 61 van boek 1.
Na het label SPRA van SHOWPR wordt er acht keer achter elkaar via de

★ 16



Figuur 16. De subroutine SHOWPR wordt in een bepaalde fase van de L-toetsroutine aangeroepen en zorgt ervoor dat de inhoud van het P-register bit voor bit, dus dat alle vlaggen stuk voor stuk en in de juiste volgorde, worden afgedrukt.

instruktie ASL-PRTEMP het zevende bit van PRTEMP de carry ingeschoven. Afhankelijk van de vraag of die carry één of nul is wordt er een 1 of een Ø afgedrukt (PRNIBL; zie punt 9 en figuur 7a, onderste helft). Daarna volgt de voorbereiding voor het volgende bit. Tenminste: als er nog een volgend bit is (BNE staat op springen).
Een overzicht:
X = Ø8: N-vlag afgedrukt;
X = Ø7: V-vlag afgedrukt;
X = Ø6: willekeurig bit afgedrukt;
X = Ø5: BRK-vlag afgedrukt;
X = Ø4: D-vlag afgedrukt;
X = Ø3: I-vlag afgedrukt;
X = Ø2: Z-vlag afgedrukt;
X = Ø1: C-vlag afgedrukt.

21 Het is de taak van de **L-toetsroutine** om de inhoud van het software-dashboard van de junior-computer in overzichtelijke vorm af te drukken. De L-toetsroutine omvat een hele reeks instrukties, waaronder zeer veel sprongen naar grafische subroutines, die in de linker helft van figuur 13b direkt volgen op het label LIST. De bespreking van deze subroutine is vrij gauw gebeurd indien we er een praktijkvoorbeeld van de L-toetsfunktie (hoofdstuk 12 van boek 3) bijnemen. De in een bepaalde fase afgedrukte tekst staat tussen aanhalingstekens. De aanhalingstekens zèlf worden niet afgedrukt! Ook het afdrukken van een spatie is in figuur 13b aangegeven. Alle in de L-toetsroutine gebruikte subroutines zijn al eerder besproken: MESSY in punt 11, PRBYT in punt 9, SHOWPR zojuist, in punt 20 en PRSP in punt 12.

22 **De P-toetsroutine** bestaat uit de zes instrukties direkt na het label PC in figuur 13b. Het indrukken van de P-toets heeft tot gevolg dat het werkadres gelijk wordt gemaakt aan de inhoud van de programmateller, dus (PCH, PCL) en vervolgens wordt afgedrukt. Het betreft de inhoud van de programmateller PC vlak voor de meest recente sprong naar PM via de STEP-ingang (zie punt 7). De funktie van de P-toets van PM is exakt gelijk aan de funktie van de PC-toets van de standaard-monitor. De toetsroutine is gauw verteld: de inhoud van de **geheugenplaatsen** PCL en PCH wordt gekopieerd in POINTL respektievelijk POINTH. Het aldus ontstane nieuwe werkadres wordt afgedrukt (PRBUFS).

23 **De M-toetsroutine** verzorgt de uitvoering van een hex dump. Deze toetsroutine bestaat uit een relatief groot aantal instrukties, vanaf het label MATRIX van figuur 13b. Zoveel zelfs dat er in twee helften een aparte figuur voor is uitgetrokken: figuur 17a/17b. Het gemeenschappelijke punt van figuur 13b en figuur 17a is het koppelpunt ⓘ.
Linksboven in figuur 17a begint de hex dump met de terugmelding, via MESSY, met de tekst "HEXDUMP:". En dan zijn we toe aan de subroutine INPAR, die we nu eerst zullen bespreken. (Vervolg M-toetsroutine in punt 25.)

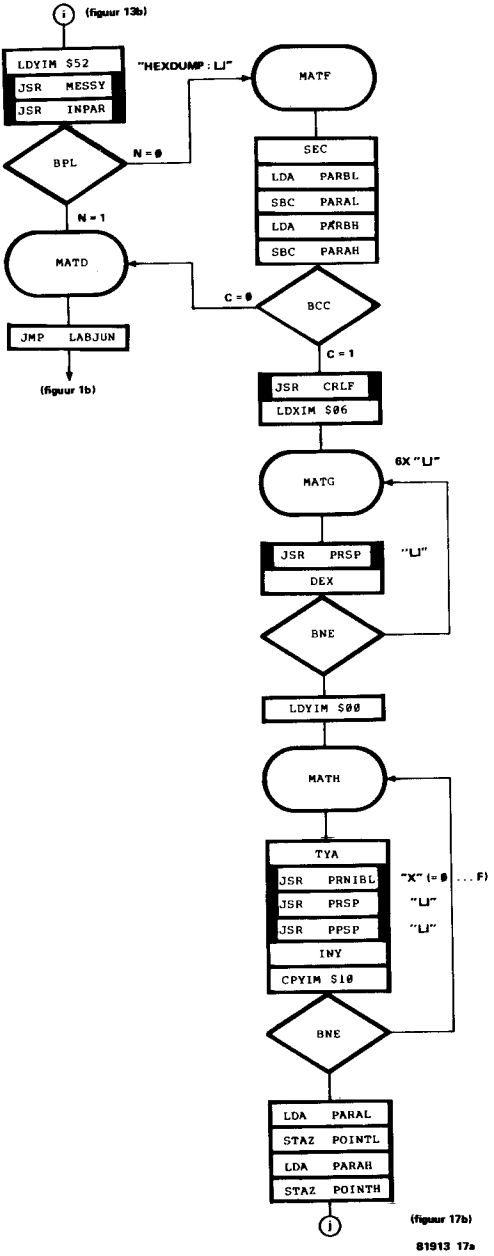
24 (intermezzo)
De subroutine INPAR van figuur 18 speelt geheel danwel gedeeltelijk (alles na het label IPB) een rol bij de toetsroutines van de toetsen M, G en S. Dat zijn de toetsen met een geparametriseerde toetsfunktie: U weet nog wel van hoofdstuk 12: die toestanden met één of twee komma's, data en de CR-toets. De subroutine INPAR regelt de gang van zaken bij het opgeven van de parameters:

opgave data-kommatoets — opgave data-toets CR.
↑ ↑
eerste adres laatste adres of ID

Figuur 18 begint met het wachten op een ingedrukte toets: de subroutine RECCHA. Dan volgt de vraag: is het een kommatoets? Zo nee, dan moet het om de opgave van geldige (numerieke toetsen) of ongeldige data gaan (alle andere toetsen met uitzondering van ","). De subroutine HEXNUM (zie punt 13 en figuur 11) zorgt voor de verwerking van geldige data Ø . . . F in de databuffers INL en INH.

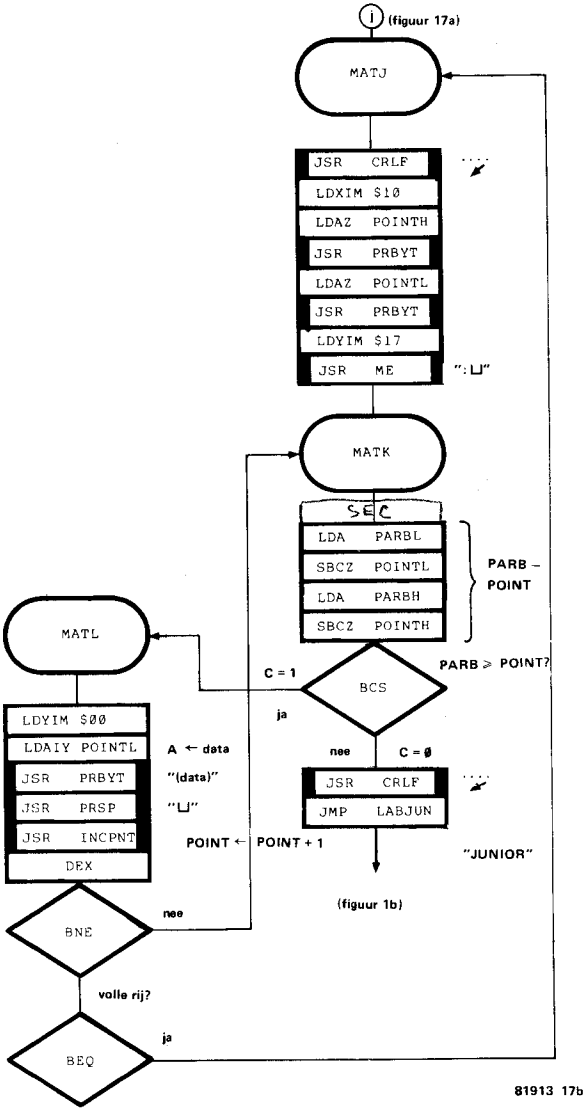
★ 17a

$ 1135



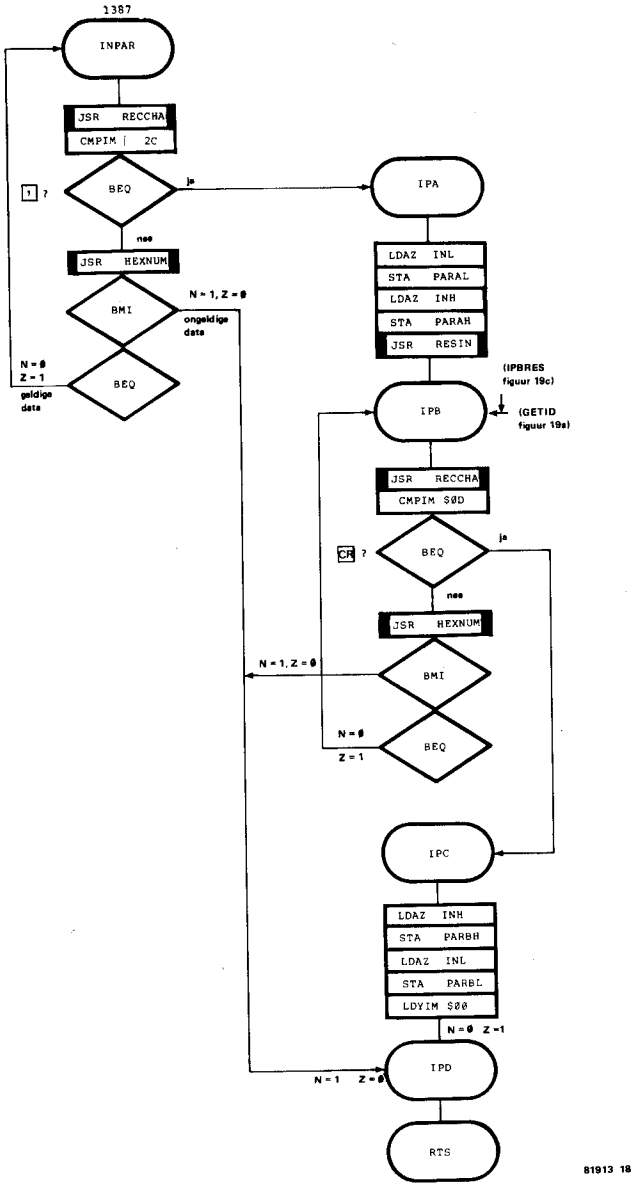
Figuur 17. Op de eerste twee instrukties na (zie dáárvoor figuur 13b) alle instrukties van de M-toetsroutine, in de vorm van de figuren 17a en 17b.

★ 17b



Zodra de kommatoets is ingedrukt (label IPA in figuur 18) is de opgave van de eerste data afgesloten. Die data gaat **naar de adresbuffers PARAL en PARAH voor het eerste adres**. Dat eerste adres kan het eerste adres zijn met de als eerste af te drukken geheugeninhoud in het geval van een hex dump (zie punt 25), of het eerste adres van een naar de cassetteband te versturen datazending (subroutine SAID; zie punt 28). Vervolgens worden de databuffers INH en INL nul gemaakt (RESIN).

★ 18



Figuur 18. De subroutine INPAR speelt een belangrijke rol bij de toetsroutines van de toetsen M, G en S, dus de toetsen met een geparametriseerde toetsfunktie. In sommige gevallen wordt alleen het tweede gedeelte, vanaf het label IPB, doorlopen.

We zijn toe aan label IPB in figuur 18. Er wordt alweer gewacht op een ingedrukte toets (RECCHA). Vervolgens rijst de vraag: is het toets CR? Dus is alle data opgegeven? Zo niet, dan ging het kennelijk nog om data en het hele verhaal van twee alinea's geleden herhaalt zich. Was het daarentegen wel dégelijk de CR-toets, dan is de opgave van de tweede data voltooid (label IPC). Die tweede data gaat **naar de adresbuffers PARBL en PARBH voor het laatste adres.** Nu worden de buffers INH en INL niet nul gemaakt. Dit in tegenstelling tot de situatie na het vaststellen van een komma. Ter afsluiting van INPAR krijgt Y een waarde nul. Daardoor is de Z-vlag 1 en de N-vlag nul geworden.

25 (vervolg M-toetsroutine: terug naar figuur 17a) Na de terugkeer uit INPAR (punt 24) bepaalt de N-vlag (BPL) hoe het verder moet. Is er tijdens INPAR ongeldige data opgegeven (inklusief het indrukken van een niet ter zake doende, niet-numerieke toets), dan volgt de gang naar het label MATD en daarmee naar het label LABJUN (figuur 1b). Na de terugmelding "JUNIOR" komen we dan bij het centrale label RESALL van de PM-hoofdroutine terecht om het opnieuw te proberen, te beginnen met het indrukken van toets M. Was evenwel tot nu toe alles in orde, dan volgt de sprong naar het label MATF. Uit de daarop volgende instrukties kan blijken dat een en ander niet meer in orde is. Want van het 16-bits getal, gevormd door (PARAH, PARAL), wordt het 16-bits getal, gevormd door (PARBH, PARBL), afgetrokken. Is het resultaat negatief, dus is de carryvlag nul, dan volgt alsnog de sprong, via MATD, naar LABJUN voor de foutmelding "JUNIOR". Waarom? Omdat het laatste adres (PARB) nooit lager mag zijn dan het eerste adres (PARA). Zowel bij een hex dump als bij het schrijven op de band van een datazending (toets S; zie punt 28).
Zit het goed met die adressen (C = Ø), dan kan met het afdrukken van de hex dump worden begonnen. De instrukties tussen de labels MATG en MATH van figuur 17a zorgen ervoor dat er achter elkaar zes spaties worden afgedrukt vanaf het begin van een nieuwe regel. Waarom dat moet kan men het beste nagaan aan de hand van een praktijkvoorbeeld van een hex dump.
Na de spaties volgt het afdrukken van de nummers Ø . . . F boven de kolommen van de hex dump. Tussen twee opeenvolgende nummers zitten twee spaties. Het slot van figuur 17a bestaat uit het gelijk maken van de inhoud van de buffers voor het werkadres (POINTH, POINTL) aan de inhoud van de buffers voor het eerste adres (PARAH, PARAL). De eerste helft van de M-toetsroutine zit erop.

26 We gaan verder met de tweede helft van de M-toetsroutine. Dus met figuur 17b. Na het label MATJ beginnen we op een nieuwe regel (CRLF); we zijn dus toe aan het afdrukken van het adres dat hoort bij de geheugenplaats waarvan de inhoud als eerste op een bepaalde rij van de hex dump wordt afgedrukt. De eerste data van de eerste rij van de hex dump bevindt zich op het adres (PARAH, PARAL) waaraan de inhoud van (POINTH, POINTL) gelijk is tijdens de allereerste passage van het label MATJ.
Dus direkt na MATJ wordt op het begin van een nieuwe regel het adres

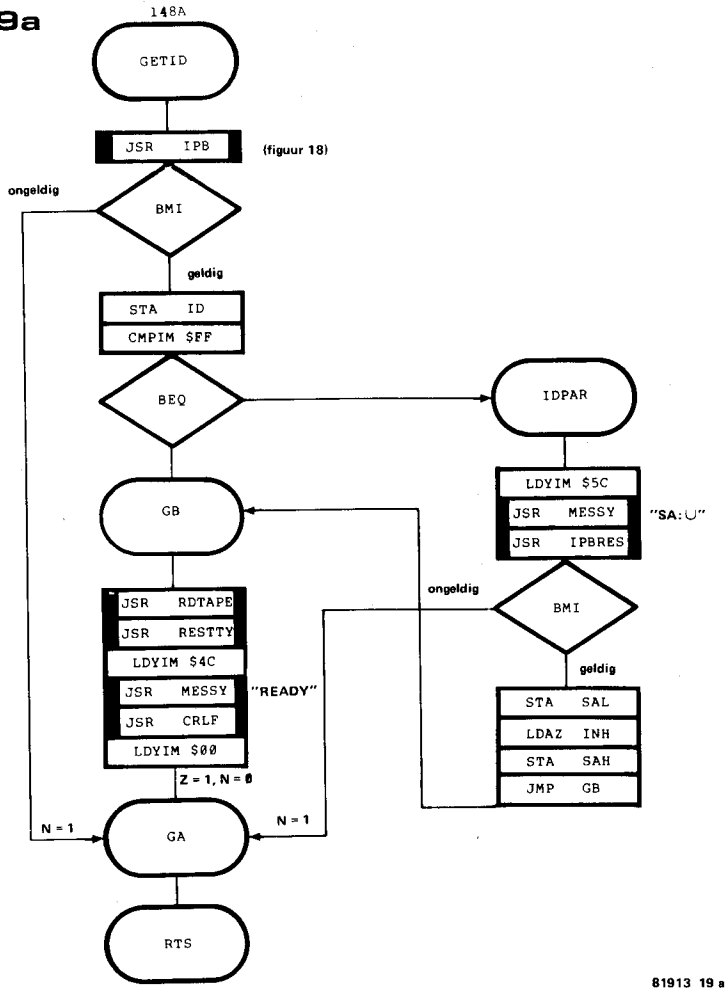
afgedrukt met de eerste, meest linkse, data van de rij. Na het afdrukken van het adres volgt de afdruk van een dubbele punt en een spatie. De subroutine ME is gelijk aan de subroutine MESSY, op de eerste CRLF na. Zie figuur 9a en punt 11. Verder wordt in het X-register – dat de beginwaarde 1Ø krijgt – het aantal eventueel nog af te drukken data van een rij van de hex dump bijgehouden.
De instrukties na het label MATK houden zich bezig met de kontrole of alle data van de hex dump al is afgedrukt. De grootte van de data-adres-wijzer POINT ten opzichte van het laatste adres (in PARB) van de hexdump is daarvoor maatgevend. Uiteindelijk vertelt de carryvlag ons hoe het ermee staat. Is het adres POINT nog niet boven PARB uitgestegen, dan (label MATL) wordt de af te drukken data opgehaald en vervolgens afgedrukt. Daarna komt er nog een spatie (PRSP) en verhogen we POINT (subroutine INCPNT; zie punt 15 en figuur 15a) en verlagen we X, ter voorbereiding van een volgende data-afdruk. Of dat op een nieuwe of op nog steeds dezelfde regel gebeurt bepaalt de toestand van de Z-vlag (BNE, BEQ). En of er nog wat aan data af te drukken valt blijkt direkt na het label MATK, dat direkt wordt bereikt (oude regel; BNE op springen) of na het afwerken van de instrukties na het label MATJ (nieuwe regel; BEQ op springen).
Zodra alle data is afgedrukt volgt er de sprong naar het begin van een nieuwe regel en de sprong naar het label LABJUN: de terugmelding "JUNIOR", als afsluiting van de hex dump. Omdat vlak voor de sprong naar LABJUN de subroutine CRLF wordt aangeroepen en omdat de terugmelding "JUNIOR" eveneens op het begin van een nieuwe regel plaatsvindt (zie de subroutines MESSY en JUNIOR) wordt er een blanko regel afgedrukt tussen de laatste regel van de hex dump en de tekst "JUNIOR".
N.B. Indien de hoeveelheid af te drukken data een veelvoud van zestien is, dus indien de hex dump uitsluitend volle regels bevat, wordt desondanks na de laatste rij van de hex dump het adres afgedrukt dat hoort bij de volgende rij (waarvoor geen af te drukken data beschikbaar is). Dit volgt uit de gang van zaken na het label MATL in figuur 17b. Zodra er op een nieuwe regel moet worden begonnen – nog niet "wetend" dat er niets meer is af te drukken want dat blijkt pas na MATK – volgt er via de BEQ de sprong naar MATJ, voor het afdrukken van een nieuw rij-adres.

27 Bij de bespreking van de **G-toetsroutine** kunnen we figuur 13b verlaten en zijn we aan de derde partij instrukties van de PM-hoofdroutine toe: figuur 13c. De G-toetsroutine omvat de instrukties die in figuur 13c direkt volgen op het label GETTAP. Het meeste werk van deze toetsroutine wordt verricht in de **subroutine GETID** van figuur 19a. Die subroutine zullen we nu eerst bespreken. Direkt na het label GETID (figuur 19a) volgt de sprong naar de subroutine IPB. Dat is geen nieuwe subroutine maar het laatste deel van de subroutine INPAR van figuur 18 en punt 24. Bij het ophalen van een datazending van de band moet er behalve het indrukken van toets G data worden opgegeven (namelijk het programmanummer ID) en vervolgens de CR-toets worden ingedrukt (het geval ID = FF komt zometeen aan de orde). Na afloop van IPB, dus na afloop van INPAR vertelt de N-vlag of het allemaal volgens de regels

is gegaan of niet. Een direkt op JSR-IPB volgende BMI leidt dan ook meteen naar AF (label GA in figuur 19a) met een N-vlag één als er gezondigd is tegen de regels voor het afwerken van de G-toetsceremonie.
In het geval van geldige data, dus een korrekt opgegeven ID, gaat de inhoud van de accu naar de geheugenplaats ID (zie hoofdstuk 16). Vlak voor het verlaten van IPB, namelijk na het label IPC en vlak voor de RTS, is de inhoud van de accu gelijk aan de inhoud van de rechter databuffer INL. Overigens: de subroutine IPB wordt pas verlaten (zondigen tegen de regels daargelaten) zodra de CR-toets is ingedrukt.
Is het zojuist ingelezen programmanummer ID gelijk aan FF, dan moet er door de gebruiker nog een startadres SA worden opgegeven: de instrukties van figuur 19a direkt na het label IDPAR. Na de terugmelding "SA:" volgt via het voorstukje IPBRES van figuur 19c opnieuw de sprong naar IPB, dus het laatste deel van INPAR, voor het ophalen van data (SA) en het wachten op de ingedrukte CR-toets. De omweg via IPBRES is nodig om de databuffers INL en INH nul te maken. Het had ook anders gekund: spring naar de instruktie JSR-RESIN, direkt vóór het label IPB in figuur 18.
In het geval dat na de tweede sprong naar IPB alles korrekt is verlopen wordt de inhoud van INH en INL gekopieerd in de geheugenplaatsen SAH respektievelijk SAL (zie hoofdstuk 16). De sprong naar het label GB vindt alsnog plaats. Het eigenlijke inlezen van de cassette-band kan beginnen. De subroutine RDTAPE komt uitgebreid aan de orde in hoofdstuk 16. De I/O is tijdens RDTAPE anders gedefinieerd dan tijdens PM; dat gebeurt aan het begin van die subroutine. Vandaar dat direkt na RDTAPE de **subroutine RESTTY** van figuur 19b volgt; de I/O krijgt hetzelfde aanzien als in figuur 1a. Rest nog de terugmelding "READY" en het één maken van de Z-vlag (N-vlag is nul). Tot slot van figuur 19a volgt de gebruikelijke RTS. Tot zover GETID.
Terug naar de G-toetsroutine. Dus terug naar figuur 13c. Indien GETID werd verlaten met een N-vlag 1, dan volgt de sprong, via het label GETERR, naar het label LABJUN, voor de foutmelding "JUNIOR". Een N-vlag nul daarentegen betekent dat alles volgens het boekje (deel 3) is gegaan; we gaan dan terug naar het centrale punt RESALL van de PM-hoofdroutine.

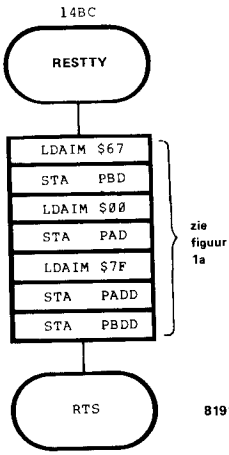
28 **De S-toetsroutine** bestaat uit de handvol instrukties die in figuur 13c direkt volgen op het label SAVID. Ook nu wordt het leeuwedeel van het werk verricht in de **subroutine SAID** van figuur 20, die we nu eerst zullen bespreken. De parametertoestanden bij de S-toets zien er als volgt uit:
Toets S – eerste data – komma – tweede data – komma – derde data – toets CR, waarbij
de eerste data een ID Ø1 . . . FE voorstelt;
de tweede data het beginadres SA voorstelt, en
de derde data het eindadres EA = LA + 1 (zie hoofdstuk 11).
De subroutine SAID begint met de vraag of de kommatoets is ingedrukt. Zo niet, dan was de opgave van de eerste data kennelijk nog niet voltooid. Dan is HEXNUM aan bod: geldige numerieke data wordt verwerkt in de databuffers INH en INL; ongeldige data levert een sprong naar het label SIB op, waarbij de N-vlag één is en de Z-vlag nul. Zodra de kommatoets

★19a



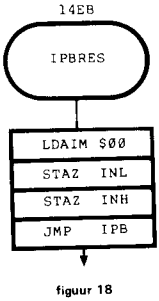
Figuur 19. De subroutine GETID vormt het leeuwedeel van de G-toetsroutine (figuur 19a). De subroutine RESTTY (figuur 19b) herstelt de I/O, zoals voor PM in figuur 1a vastgelegd, nadat de cassette-subroutine RDTAPE of DUMP (zie figuur 20) is doorlopen. In een bepaalde fase van GETID wordt via het stukje programma IPBRES (figuur 19c) naar de tweede helft van INPAR (figuur 18) gesprongen.

★ 19b



81913 19b

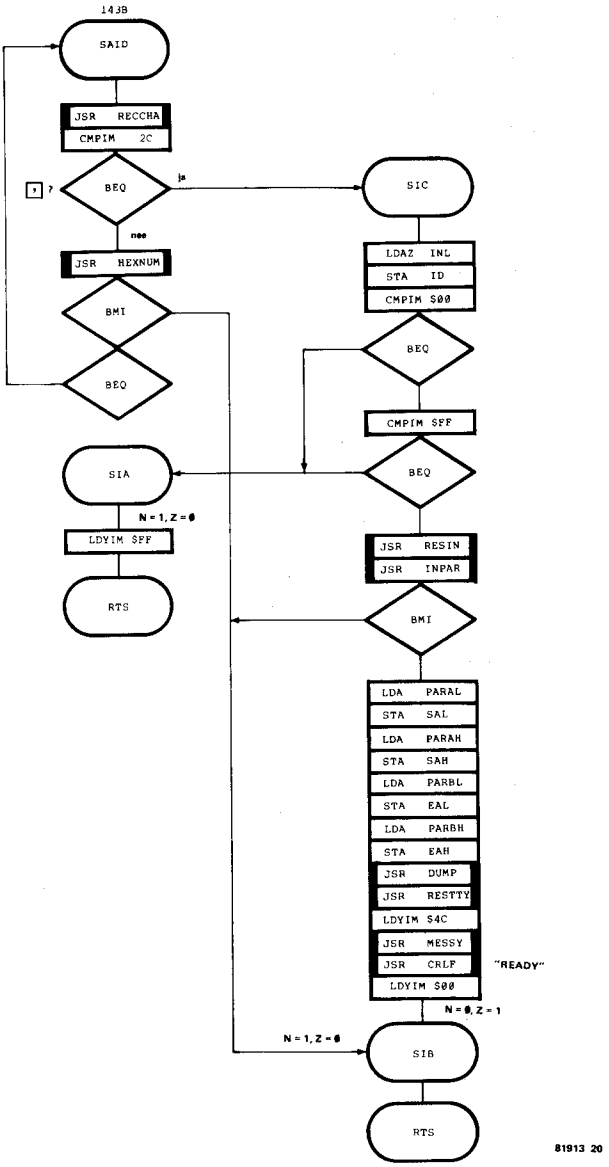
★ 19c



81913 19c

is ingedrukt volgt de sprong naar het label SIC (sic!). De inhoud van INL gaat naar ID en er wordt gekeken of er soms – geheel tegen de regels – een ID ØØ of FF is opgegeven. Want dan gaat het linea recta naar de fout-uitgang SIA, met een N-vlag één en een Z-vlag nul.
Voor het verwerken van de tweede en derde data en de afsluitende CR kunnen we vervolgens een beroep doen op de subroutine INPAR, die we in figuur 18 vinden en die is besproken in punt 24. Wèl worden eerst nog even de databuffers nul gemaakt (RESIN).
Zodra de CR-toets is ingedrukt kunnen we met de eigenlijke datazending naar de cassetteband beginnen. De SA-buffers SAH en SAL krijgen de inhoud toegewezen van de buffers PARAH en PARAL voor het eerste adres. Evenzo krijgen de EA-buffers EAH en EAL de inhoud toegewezen van de buffers PARBH en PARBL voor het laatste adres. En dan kunnen we een geheel verzorgde datareis naar de cassette-band genieten. De sub-routine DUMP is in detail in hoofdstuk 16 besproken. Net als bij RDTAPE (punt 27) is de I/O tijdens DUMP afwijkend van de situatie tijdens PM. Dat gebeurt aan het begin van DUMP. Vandaar dat ook nu na afloop van DUMP een beroep wordt gedaan op de subroutine RESTTY van figuur 19b.
Het slot van het liedje is de terugmelding "READY", en verder het nul maken van Y; de N-vlag is nul en de Z-vlag 1.
Terug naar de S-toetsroutine van figuur 13c. Na afloop van SAID laat de N-vlag zien of het allemaal korrekt is verlopen of niet. Zo ja, terug naar het centrale label van PM, dus naar RESALL. Zo niet: via GETERR naar LABJUN, voor de melding "JUNIOR", die in dit geval het karakter van een foutmelding heeft.

29 Tot slot van de PM-hoofdroutine hebben we dan nog **de datatoets-routine.** Het gaat om de instrukties die in figuur 13c volgen op het label VALNUM. In de PM-hoofdroutine bevinden we ons nu in een fase waarin alle ingedrukte toetsen die voor PM een betekenis hebben zijn



Figuur 20. De subroutine SAID vormt het leeuwedeel van de S-toetsroutine. Hij omvat ondermeer de sprong naar de TM-subroutine DUMP, voor het op de cassetteband schrijven van een datazending.

"uitgefilterd". Dus achtereenvolgens de toetsen:
+ (punt 15)
– (punt 16)
SP (punt 17)
"." (punt 18)
R (punt 19)
L (punt 21)
M (punten 23, 25 en 26)
G (punt 27)
S (punt 28)
Alle andere, nog niet genoemde toetsen zijn nu nog niet uitgefilterd. Het kan gaan om niet-numerieke toetsen – die hoogstwaarschijnlijk per ongeluk zijn ingedrukt – òf om ingedrukte numerieke toetsen. In het laatste geval gaat het om data die hoort bij de opgave van een werkadres of bij de opgave van werkdata. Want de data die hoort bij de toetsroutines van de geparametriseerde toetsfunkties M, G en S wordt niet behandeld via de RECCHA in de centrale RESALL-READCH-lus van PM.
Voor het verwerken van numerieke data is er de subroutine HEXNUM van figuur 11. Deze is besproken in punt 13. Tijdens het verblijf in HEXNUM worden alle resterende niet-numerieke toetsen eruit gesmeten via de foutmelding "WHAT?". In dat geval volgt de sprong naar het label VNA en daarmee de sprong terug naar het centrale PM-label RESALL. In het geval van een numerieke toets wordt het met deze toets overeenkomende datanibble verwerkt in de databuffers INH en INL. Na afloop volgt de sprong naar het label READCH; zie figuur 13a. Nu worden de buffers INH en INL niet nul gemaakt (RESIN, direkt na het label RESALL) omdat de opgave van data nog niet voltooid hoeft te zijn.

Hiermee is alle PM-software besproken. Rest nog een aanvullende opmerking over het BRK-toets-gebeuren.

30 Zoals eerder opgemerkt (zie hoofdstuk 12 van boek 3 en punt 8) sorteert het indrukken van de BRK-toets uitsluitend effekt tijdens het afdrukken, door de junior-computer, van het een en ander. Daarbij is altijd de elementaire afdruk-subroutine PRCHA betrokken.
Men kan echter in de praktijk vaststellen dat bij PM óók tijdens het wachten op een ingedrukte toets (RECCHA) het indrukken van de BRK-toets tot de terugmelding "JUNIOR" leidt, net zo als in het geval van het onderbreken van drukwerk (label LABJUN in figuur 1b; zie ook punt 1 en punt 6).
Hoe zit dat?
Dat zit zó:
Indien gedurende tenminste negen opeenvolgende bittijden de BRK-toets is ingedrukt wordt er in feite de ASCII-kode ØØ verzonden. Aan die kode heeft de junior-computer geen boodschap, dus volgt de terugmelding "WHAT"?. Althans, dat is de bedoeling. Want zodra begonnen is met het afdrukken van deze boodschap komt de nog steeds ingedrukte BRK-toets aan het licht en ontstaat er, zoals eerder beschreven, de terugmelding "JUNIOR". Dàt gebeurt echter pas nadat de BRK-toets is losgelaten.

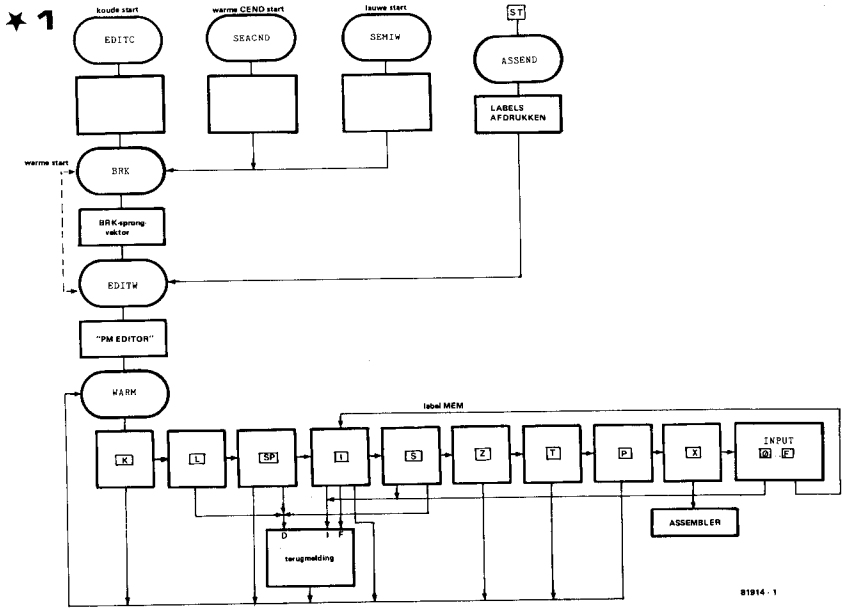
15

766 bytes PME-software

Hoe gaat het editen op tv in zijn werk?

In hoofdstuk 13 heeft u leren omgaan met de nieuwe editor, PME. U heeft als het ware leren rijden met PME en in dit hoofdstuk gunnen we u een uitgebreide blik onder de motorkap. De byte-voor-byte-kennis van PME verschaft u inzicht in de aardigheden en eigenaardigheden van dit systeemprogramma. Het programma is voor een deel gebaseerd op bestaande subroutines. Dat neemt niet weg dat er óók nieuwe subroutines zijn "geboren". En die subroutines zijn uitstekend bruikbaar voor ego-software.
Ofschoon de bespreking van PME, gezien het aantal bytes, aanzienlijk minder ruimte vergt dan de bespreking van de andere software in dit boek is ook in dit hoofdstuk gekozen voor een korte maar krachtige punt-voor-punt-behandeling. Trouwens: dat "kort maar krachtig" sluit uitstekend aan bij de filosofie van het programma PME.

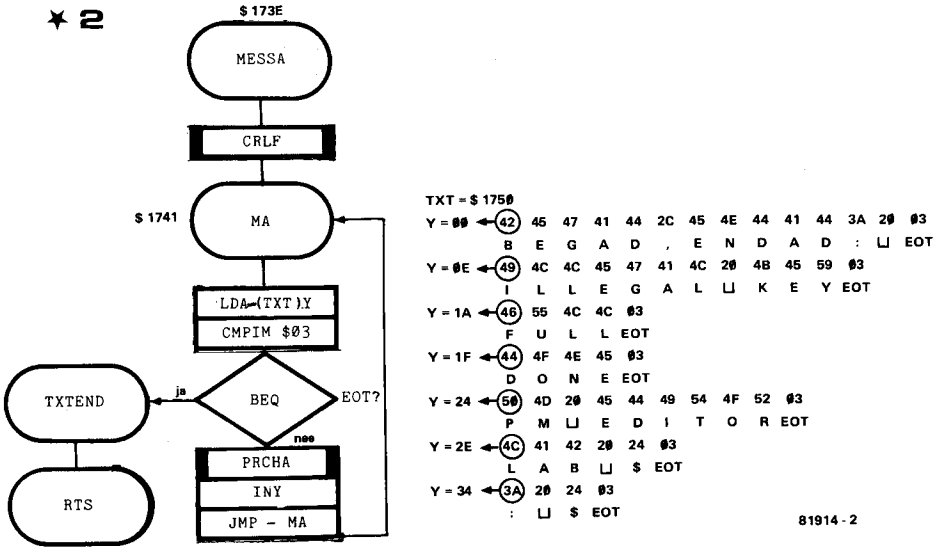
1 We beginnen maar eens met een **algemeen overzicht van PME**. Dus met een soort **blokschema**; het "programma van het programma" en tevens het programma van dit hoofdstuk. Het blokschema staat in figuur 1.
De struktuur van PME ziet er als volgt uit: Een hoofdroutine wordt voorafgegaan door een incidenteel, af en toe doorlopen voorroutine. Zoals een rivier ontstaat (stroomdiagram!) uit de samenvloeiing van een aantal kleinere rivieren en beken, zo kent de voorroutine van PME een drietal verschillende oorsprongen, die overeenkomen met de koude, de lauwe en de warme CEND-startroutine. Een vierde "zijrivier", namelijk de routine na het label ASSEND, moet men eigenlijk opvatten als een tussenroutine, die wordt doorlopen nadat een programma is geassembleerd. Dat doorlopen gebeurt op "initiatief" van de ingedrukte toets ST van het standaardtoetsenbord.



Figuur 1. Het blokschema van het systeemprogramma PME.

Meestal zal het verblijf in PME inhouden dat men zich in de hoofdroutine van PME bevindt. Dat is alles na het label WARM in figuur 1. De gang van zaken binnen de hoofdroutine ziet er bekend uit. Nadat er een ingedrukte toets is vastgesteld volgt de bekende "software-enquête": is het toets Jan? Zo ja: voer de Jan-toetsroutine uit. Zo nee. is het soms toets Piet? Zo ja: voer de Piet-toetsroutine uit. Zo nee: is het toets Henk? Enzovoorts. Na de uitvoering van de bijbehorende toetsroutine gaat het terug naar het centrale label WARM van de PM-hoofdroutine, voor het wachten op een nieuwe ingedrukte toets en het vervolgens afhandelen van de werkzaamheden die bij die toets horen.
Soms vindt de sprong terug naar WARM plaats via een tussenstation: de terugmelding "DONE", "ILLEGAL KEY" of "FULL". In het geval van de X-toetsroutine gaan we na afloop niet terug naar WARM, maar naar de assembler; het label WARM wordt eerst na een omweg (toets ST, label ASSEND, label EDITW) bereikt. Het laatste deel van de I-toetsroutine (vanaf het label MEM) is tevens het laatste deel van de input-toetsroutine.

2 **De subroutine MESSA** van figuur 2 is er voor het afdrukken van een tekst, door de junior computer, bij monde van PME. Deze subroutine is dezelfde als de subroutine MESSY van figuur 9a en van punt 11 van hoofdstuk 14. Met één verschil: het eerste adres van de opzoektabel is nu niet MESS, maar TXT. In die opzoektabel liggen de volgende, af te

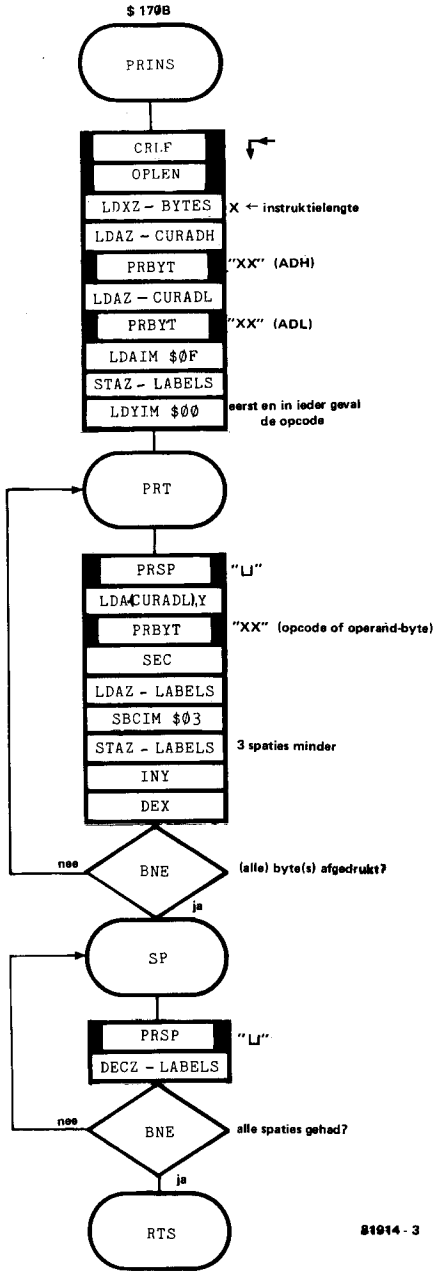


Figuur 2. De PME-subroutine MESSA is identiek aan de PM-subroutine MESSY, zij het dat een andere opzoektabel nodig is om de specifieke PME-terugmeldingen te verzorgen.

drukken teksten: "BEGAD,ENDAD", "ILLEGAL KEY", "FULL", "DONE", "PM EDITOR", "LAB $" en "$". De keuze van de af te drukken tekst wordt bepaald door de waarde van Y, direkt vóór de sprong naar de subroutine MESSA.

3 **De subroutine PRINS** van figuur 3 heeft tot taak het afdrukken van een instruktie, voorafgegaan door het adres met de opcode van die instruktie en gevolgd door een aantal spaties dat wordt bepaald door het aantal bytes van de afgedrukte instruktie. Welke instruktie wordt er eigenlijk afgedrukt? Want er zijn er meer dan één. Die instruktie wordt afgedrukt waarvan de opcode zich op een adres bevindt dat gelijk is aan de inhoud van de adreswijzer CURAD. In de hoofdstukken 5 en 8 is die CURAD "displaywijzer" genoemd; in de PME-situatie is het handiger om te spreken van "instruktiewijzer". Dus goed onthouden: CURAD heeft altijd iets te maken met de laatste instruktie, die op de linker kolom van het tv-scherm of printerpapier is afgedrukt of zal worden afgedrukt. En dàt ie op de linker kolom wordt of is afgedrukt zal nu blijken.
De subroutine PRINS van figuur 3 begint met de subroutine CRLF, welke laatste is besproken in punt 12 van hoofdstuk 14. Er wordt dus begonnen op het begin van een nieuwe regel. Er wordt dus begonnen op de beginpositie van een regel op de linker kolom. Als dat achter de rug is komt er alwéér een subroutine om de hoek kijken: OPLEN. Dat is een oude bekende van de standaard-editor. Hij staat beschreven op de pagina's 154 . . . 158 (hoofdstuk 8) van boek 2. De opcode van een instruktie gaat

✶ 3

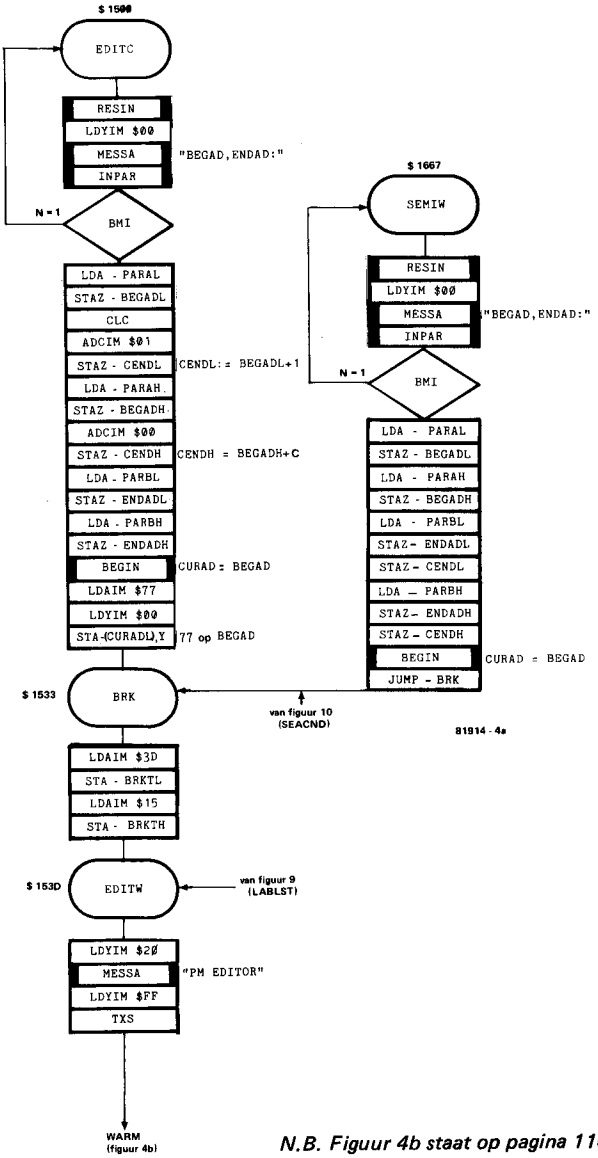


Figuur 3. De PME-subroutine PRINS zorgt ervoor dat een door de instruktiewijzer CURAD aangewezen instruktie mèt het adres van de opcode wordt afgedrukt, op dat deel van een regel dat deel uitmaakt van de linker reaktiekolom. Tevens wordt een zodanig aantal spaties op dezelfde regel afgedrukt dat een vervolgens door de gebruiker ingedrukte toets tot uiting komt op het begin van de rechter aktiekolom.

de accu in en er wordt vastgesteld of de instruktie één, twee of drie bytes lang is. De instruktielengte staat in de geheugenplaats BYTES.
In figuur 3 krijgt het X-register na OPLEN de inhoud van BYTES toegewezen. Er is dus bekend hoeveel bytes er straks moeten worden afgedrukt. Maar vóór het zover is wordt eerst de inhoud van CURADH en CURADL afgedrukt. Dat is het adres met de opcode van de af te drukken instruktie. Dat gebeurt via twee opeenvolgende aanroepen van de subroutine PRBYT, die in punt 9 van hoofdstuk 14 is besproken. Daarna krijgt de geheugenplaats LABELS de waarde ØF toegekend. Dat heeft iets te maken met het aantal spaties dat straks, na het label SP en na de afdruk van de instruktie, zal worden afgedrukt.
We zijn nu toe aan alles van figuur 3 tussen het label PRT en het label SP. Dit gedeelte van PRINS wordt, afhankelijk van de instruktielengte, dus afhankelijk van de beginwaarde van X, één, twee of drie keer achter elkaar doorlopen. De eerste doorloop vindt plaats met Y gelijk aan nul, een eventuele tweede doorloop met Y gelijk aan Ø1 en een eventuele derde doorloop met Y gelijk aan Ø3. Elke doorloop begint met het afdrukken van een spatie (PRSP; zie punt 12 van hoofdstuk 14). Daarna krijgt de accu de inhoud van de geheugenplaats toegewezen die volgt uit de inhoud van CURADL en CURADH èn uit de index Y. Met andere woorden: eerst de opcode van de instruktie, dan eventueel het eerste of enige operand-byte en dan eventueel het tweede operand-byte. Het eigenlijke drukwerk doet de subroutine PRBYT. Verder wordt de inhoud van LABELS tijdens elke doorloop van PRT met drie verlaagd. Immers: voor elk byte dat je afdrukt hoef je straks drie spaties minder af te drukken om op het begin van de rechter aktiekolom uit te komen.
Het laatste deel van PRINS, dus alles na het label SP, houdt zich bezig met het afdrukken van die spaties. Hoeveel dat er zijn hangt af van de eindstand van LABELS, dus van het aantal keren dat alles tussen PRT en SP is doorlopen en dus van de lengte van de afgedrukte instruktie. Nadat de instruktie is afgedrukt volgen er 12 spaties in het geval van een één-byte-instruktie, 9 spaties in het geval van een twee-byte-instruktie en 6 spaties in het geval van een drie-byte-instruktie.
Overigens: Niet in elke toepassing van PRINS is het afdrukken van spaties strikt nodig. Bijvoorbeeld in het geval van de P- of L-toetsroutine. Want dan vervalt gedurende een aantal regels het nut van een rechter aktiekolom. Weliswaar is in dit soort gevallen de afdruksnelheid lager dan strikt noodzakelijk is, maar de bytewinst is belangrijker dan dat snelheidsverlies.

4 Wat gebeurt er na een **koude start van PME?** Dan worden de instrukties van de voorroutine van PME (figuur 4a) vanaf het label EDITC tot aan het label WARM doorlopen. De instrukties vanaf EDITC tot aan het label BRK worden uitsluitend tijdens de koude start doorlopen, de instrukties na het label BRK worden óók in bepaalde andere gevallen doorlopen.
Direkt na het label EDITC volgt de aanroep van de subroutine RESIN. Deze subroutine hebben we in hoofdstuk 14, in punt 14, leren kennen. De inhoud van de databuffers INH en INL wordt nul. Direkt daarop volgt de aanroep van de subroutine MESSA (punt 2); er volgt de terugmelding "BEGAD, ENDAD:". Eigenlijk is het geen terugmelding, maar een vraag.

★ 4a



N.B. Figuur 4b staat op pagina 110!

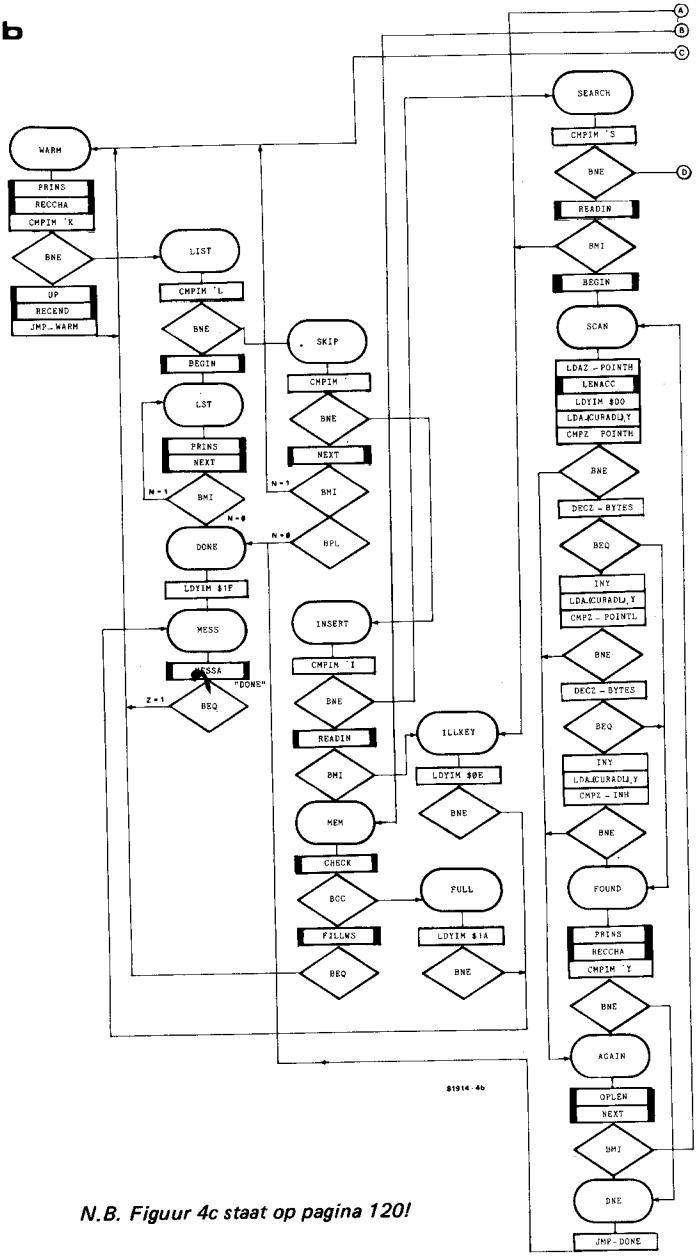
Figuur 4a. De voorroutines EDITC en SEMIW van PME, met inbegrip van een gemeenschappelijk gedeelte vanaf het label BRK.

Een vraag aan u. Want u wordt als gebruiker geacht om een BEGAD en een ENDAD op te geven. Hoe ging dat ook weer? Eerst de data van BEGAD intoetsen, dan een komma, vervolgens de data van ENDAD intoetsen en tenslotte de toets CR indrukken. Er bestáát al een subroutine die dat allemaal voor zijn rekening neemt. Namelijk INPAR, die ondermeer is gebruikt bij de M-toetsroutine van PM (hexdump). Hij is behandeld in punt 24 van hoofdstuk 14. Indien er, op welke manier dan ook, iets niet volgens de regels is gegaan tijdens de opgave van data, komma en CR, dan is na afloop van INPAR de N-vlag één en de Z-vlag nul. De BMI, die in figuur 4a op JSR-RESIN volgt, zorgt ervoor dat er helemaal opnieuw wordt begonnen ("valse start"), dus dat opnieuw de terugmelding "BEGAD, ENDAD:" op het papier of tv-scherm is te zien. In een aantal gevallen volgt overigens nog tussentijds de terugmelding "WHAT?". Dit omdat, zoals in figuur 18 van hoofdstuk 14 is te zien, op twee plaatsen in INPAR de subroutine HEXNUM wordt aangeroepen. En in het geval van een niet-numerieke toets volgt dan de voornoemde foutmelding.
Voor HEXNUM: zie figuur 11 en punt 13 van hoofdstuk 14.
Indien de opgave van BEGAD en ENDAD korrekt is voltooid volgt het "overhevelen" van de inhoud van de buffers PARAH en PARAL voor het eerste adres in BEGADH, respektievelijk BEGADL. De inhoud van de buffers PARBH en PARBL voor het tweede adres wordt gekopieerd in de geheugenplaatsen ENDADH, respektievelijk ENDADL. Verder krijgen de geheugenplaatsen CENDH en CENDL een zodanige inhoud toegewezen dat de variabele eindadreswijzer CEND op een adres staat gericht dat één hoger is dan het beginadres BEGAD van het geheugenbereik. Dat is nodig omdat op BEGAD het EOF-teken 77 komt te staan. Naarmate er instrukties en labels worden toegevoegd (Input) of tussengevoegd (Insert) schuiven die 77 en CEND op naar een hogere geheugenplaats respektievelijk adres. Naarmate er instrukties of labels worden verwijderd ("Delete", "Kill"; zie punt 6) schuiven de 77 en CEND op naar een lagere geheugenplaats respektievelijk adres.
Voordat de rode lantarendrager 77 op BEGAD kan worden gezet moet de instruktiewijzer CURAD worden gericht op BEGAD. Dat gebeurt via de subroutine BEGIN, een oude bekende van hoofdstuk 8 van boek 2. Nadat de PME-voorroutine van figuur 4a is doorlopen zal de eerste instruktie, met de opcode in BEGAD, worden afgedrukt. Direkt na de koude start is dat dus de één-byte-instruktie met de pseudo-opcode 77.

5 Het tweede deel van de PME-voorroutine van figuur 4a wordt ondermeer doorlopen na een **warme start van PME**. Een warme start begint bij het label BRK. Indien men voor deze ingang van PME kiest gaat men uit van waarden voor de grootheden BEGAD, ENDAD, CURAD en CEND die zijn vastgelegd tijdens de voorgeschiedenis van de computersessie of tijdens de voorroutines SEMIW (punt 19), SEACND (punt 20) of de tussenroutine LABLST (punt 17; tussen label EDITW). Wat er direkt na een warme start in ieder geval wèl moet worden vastgelegd is de BRK-sprongvektor. Vóór de warme start zaten we namelijk in PM, en de BRK-sprongvektor is dáár gericht op het label LABJUN, met als gevolg de terugmelding "JUNIOR" (zie hoofdstuk 14).
Het is dan ook niet verwonderlijk dat direkt na het label BRK in figuur 4a

de BRK-sprongvektor wordt gespecificeerd op het adres $153D. Dat is het adres dat hoort bij het label EDITW. Na dit laatste label volgen er enige instrukties die zorgen voor de terugmelding "PM EDITOR" en voor het resetten van de stack pointer op FF. Dat laatste kan zeer nuttig zijn na het indrukken van de BRK-toets omdat er situaties denkbaar zijn dat niet alleen drukwerk moet worden onderbroken maar dat ook de stapel dreigt "vol te lopen" (Dat zogenaamde "overlopen" van de stapel is onzin: zodra het hoogste punt, adres Ø1ØØ, is bereikt, wordt vervolgens opnieuw begonnen bij de "onderste" stapelplaats, met adres Ø1FF; zie pagina 94 en figuur 18 in de derde druk van boek 1).
N.B. Voor een warme start kan men kiezen voor adres $153D (label EDITW) in plaats van voor adres $1533 (label BRK). Dat betekent dan wèl dat de BRK-sprongvektor volgens PM gespecificeerd is, en dat verder ook blijft. Verder is de BRK-sprongvektor eveneens nog volgens PM gedikteerd tijdens de afwikkeling van de instrukties (tot aan het label EDITW) van de voorroutines EDITC en SEMIW van figuur 4a en de voorroutine SEACND van figuur 10. Het indrukken en weer loslaten van de BRK-toets tijdens het afdrukken van "BEGAD, ENDAD:" of tijdens het verblijf in INPAR (met daarin op twee plaatsen een JSR-RECCHA, zie punt 30 van hoofdstuk 14) heeft de terugmelding "JUNIOR" tot gevolg. Het indrukken en weer loslaten van de BRK-toets tijdens het verblijf in PME – waarbij de instrukties vanaf het label BRK zijn doorlopen – heeft altijd de terugmelding "PM EDITOR" tot gevolg.

6 We zijn toe aan de bespreking van de **hoofdroutine van PME**. Te beginnen met figuur 4b. En te beginnen met **de K-toetsroutine**. Dat zijn de instrukties die direkt volgen op het centrale label WARM, linksboven in figuur 4b. Begonnen wordt met het afdrukken van de instruktie. Welke instruktie dat is hangt af van de inhoud van de adreswijzer CURAD. Het afdrukken vindt plaats met behulp van de subroutine PRINS. Deze subroutine is besproken in punt 3. Zie ook figuur 3. Na PRINS volgt de subroutine RECCHA: wacht op een ingedrukte toets en zodra dat het geval is: zet de ASCII-kode van die ingedrukte toets in de accu.
Voor RECCHA: zie hoofdstuk 14, punt 5 en figuur 5.
Nadat blijkt dat de ingedrukte toets K is volgt de subroutine UP. Dat is een subroutine van de standaard-editor. Wat doet deze? Deze zorgt ervoor dat de instruktie die door de instruktiewijzer CURAD is aangewezen, in het geheugen wordt overgeschreven door alle, op die verwijderde (overschreven) instruktie volgende labels en instrukties. Het programma ("user file") wordt dus met één instruktie ingekort. Na de subroutine UP volgt een andere subroutine van de standaard-editor, te weten RECEND. De inhoud van de variabele eindadreswijzer CEND wordt verlaagd met een bedrag dat volgt uit de inhoud van de geheugenplaats BYTES. Dat laatste heeft iets met een instruktielengte te maken. Namelijk het aantal bytes van de verwijderde instruktie. Die BYTES speelt overigens óók een rol in de subroutine UP. Er hoeft hier geen beroep te worden gedaan op de subroutine OPLEN of LENACC, omdat de verwijderde instruktie de als laatste afgedrukte instruktie is; de inhoud van BYTES is daarmee in overeenstemming. Zie de JSR-OPLEN in de subroutine PRINS van punt 3 en figuur 3.



N.B. Figuur 4c staat op pagina 120!

Figuur 4b. Het eerste deel van de hoofdroutine van PME, namelijk de toetsroutines van de toetsen K, L, SP (spatiebalk, niet "S" plus "P"!), I en S.

Voor UP: zie pagina 152 en de figuren 17 en 18 van hoofdstuk 8 van boek 2.
Voor RECEND: zie pagina 149 en figuur 14 van hoofdstuk 8 van boek 2.
Ter afsluiting van de K-toetsroutine volgt de sprong terug naar het centrale label WARM van de PME-hoofdroutine. Alles wat gedaan moest worden is gedaan: de als laatste afgedrukte instruktie is uit het geheugen verwijderd. Omdat de stand/inhoud van CURAD ongewijzigd is gebleven volgt na de terugkeer naar WARM het afdrukken van die instruktie die in het geheugen direkt volgde op de zojuist verwijderde instruktie.

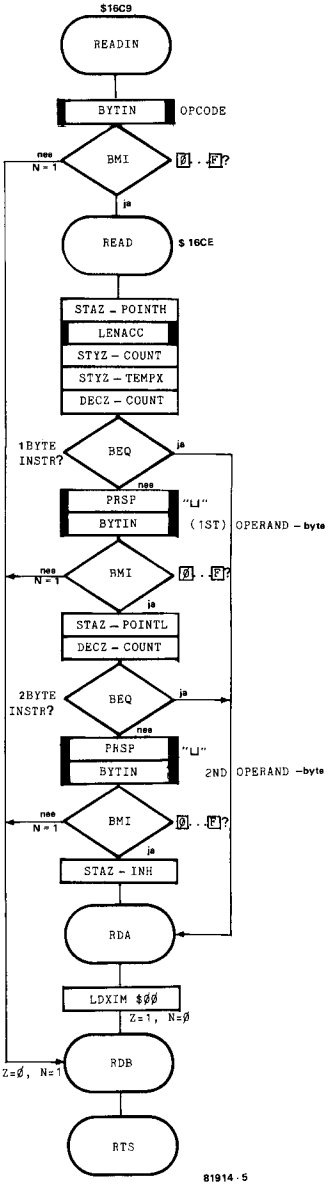
7 **De L-toetsroutine** bestaat uit alle instrukties in figuur 4b vanaf het label LIST tot aan het terugmeldingslabel DONE. Deze toetsroutine moet ervoor zorgen dat alle instrukties van het programma waaraan men werkt worden afgedrukt. Dat afdrukken moet ophouden zodra de variabele eindadreswijzer CEND door de instruktiewijzer CURAD is bereikt.
Is er een ingedrukte toets L vastgesteld, dan volgt allereerst de subroutine BEGIN: als eerste wordt de eerste instruktie van het programma, met de opcode op het adres BEGAD, afgedrukt. De rest van de L-toetsroutine bestaat uit een programmalus rond het label LST, bestaande uit de subroutines PRINS, NEXT en een BMI. De subroutine NEXT hebben we "geleend" van de standaard-editor. Hij staat beschreven in hoofdstuk 8 van boek 2. Zie de pagina's 140 en 141 en figuur 10 aldaar. De instruktiewijzer (CURAD) wordt opgehoogd met een bedrag dat in overeenstemming is met de meest recente inhoud van BYTES, dus bij deze toetsroutine in overeenstemming is met de lengte van de zojuist, tijdens PRINS afgedrukte instruktie. Het laatste deel van NEXT is er voor de kontrole of de verhoogde CURAD niet een hoger adres aanwijst dan CEND, danwel dat de inhoud van CURAD kleiner is dan de inhoud van CEND of gelijk is aan die van CEND.
In het laatste deel van NEXT wordt de inhoud van CEND afgetrokken van de inhoud van CURAD. Er geldt:
CEND $>$ CURAD $\rightarrow$ N = 1 en C = Ø
CEND $\leqslant$ CURAD $\rightarrow$ N = Ø en C = 1
De toestand van de N-vlag geldt voorzover het resultaat van de aftrekking niet negatiever is dan -127, met andere woorden: zolang het meest linkse bit van het resultaat van de aftrekking één is. In alle situaties van PME en van de standaard-editor waar de subroutine NEXT aan bod is kan men na afloop daarvan dus gaan testen met een BMI en/of BPL, in plaats van de formeel juistere test van de carry-vlag via de instruktie(s) BCC en/of BCS.
Nadat de rode lantarendrager 77 is afgedrukt (die staat op een geheugenplaats met een adres, één lager dan CEND) en nadat opnieuw NEXT is doorlopen, wordt CURAD gelijk aan CEND (bij een "opcode" 77 hoort een instruktielengte één). Dit heeft tot gevolg dat de N-vlag niet langer één is en dat dus de afdruklus rond het label LST is onderbroken. Het label DONE wordt nu afgewerkt. Na de terugmelding "DONE", dus nadat tijdens MESSA de Z-vlag één is geworden (omdat er een afsluitende EOT in de accu staat; zie figuur 2), volgt de sprong terug naar het centrale label WARM van de PM-hoofdroutine. Tijdens de dáárop volgende afwerking

van de subroutine PRINS wordt er een instruktie afgedrukt waarvan de opcode zich op het adres CEND bevindt. Zeer waarschijnlijk is dat een instruktie die niet door de gebruiker is opgegeven. Een fantasie-instruktie dus, met een fantasie-opcode. In ieder geval staat vast dat de als laatste ingedrukte instruktie niet hoort bij het programma waaraan men op dat moment aan het editen is. Omdat hij op een adres zit dat gelijk is aan CEND, de variabele eindadreswijzer.

8 **De spatietoetsroutine** is nu aan bespreking toe. Die routine bestaat uit alle instrukties in figuur 4b direkt na het label SKIP. Het indrukken van de spatietoets heeft tot gevolg dat de instruktie wordt afgedrukt die in het geheugen direkt volgt op de instruktie die vóór het indrukken van de spatiebalk als laatste is afgedrukt. Er gebeurt dus hetzelfde als bij de L-toetsroutine, maar dan voor één instruktie en niet te beginnen bij BEGAD maar te beginnen en te eindigen bij de instruktie die wordt aangewezen door de nieuwe inhoud van de instruktiewijzer CURAD. En wat dat laatste betreft: het is niet verwonderlijk dat de spatietoetsroutine de subroutine NEXT bevat. Deze wordt aangeroepen zodra blijkt dat de ingedrukte toets toets L is. Over NEXT hebben we het zojuist nog, in punt 7 uitgebreid gehad. Na NEXT volgen de instrukties BMI en BPL. Beide instrukties reageren op de N-vlag; er staat dus altijd één van de twee op springen. In het ene geval volgt de gang terug naar WARM en in het andere geval gebeurt dat pas na de terugmelding "DONE". In beide gevallen wordt er een nieuwe instruktie afgedrukt. Zolang CEND nog niet door CURAD is "ingehaald" is dat de volgende instruktie in het geheugen en zodra CURAD gelijk is geworden aan CEND wordt na "DONE" een instruktie afgedrukt waarvan de opcode zich op het adres CEND bevindt. De gang van zaken is dus wat dat betreft exakt gelijk aan het einde van de L-toetsroutine (zie punt 7).

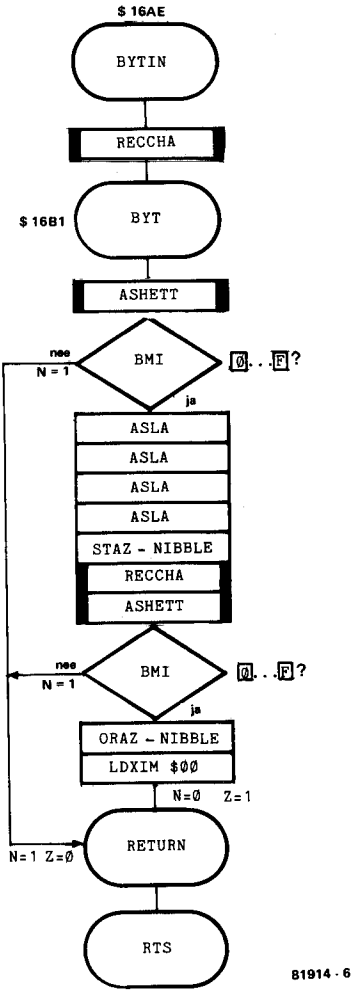
9 Voordat de I-toetsroutine zal worden besproken (zie punt 11) volgt er nu en in punt 10 een intermezzo, tijdens welk een drietal PME-subroutines zal worden besproken. Allereerst de **subroutine READIN/READ** van figuur 5. Deze subroutine is kwa funktie te vergelijken met de subroutine RDINST van de standaard-editor (zie de pagina's 142 . . . 144 en figuur 11 van boek 2). Wat doet READIN/READ? Deze zorgt voor de verwerking van een door de gebruiker opgegeven instruktie. Hij wordt gebruikt bij de Insert,- Search- en Input-toetsroutines (zie de punten 11, 12 respektievelijk 18).
Zoals in RDINST – afhankelijk van de instruktielengte – één, twee of drie keer een beroep wordt gedaan op de subroutine GETBYT, zo wordt in READIN/READ één, twee of drie keer een beroep gedaan op de **subroutine BYTIN/BYT**. Deze staat in figuur 6 en zal nu eerst in detail worden besproken.
De naam van de subroutine zegt het al: het gaat erom dat twee opeenvolgende ingedrukte numerieke toetsen Ø . . . F worden verwerkt tot een linker, respektievelijk een rechter datanibble, die, tot één databyte samengevoegd, in de accu komen te staan. Figuur 6 begint met de subroutine RECCHA: wacht op een ingedrukte toets. In de PM-subroutine ASHETT, die daarop volgt, wordt in het geval van een numerieke toets de

★ 5



Figuur 5. De subroutine READIN/READ verzorgt het inlezen van een instruktie en wordt gebruikt bij de I-, S- en Input-toetsroutine.

★ 6



Figuur 6. De subroutine BYTIN/BYT verzorgt het inlezen van een databyte, dus twee datanibbles, die horen bij twee ingedrukte numerieke toetsen. Deze subroutine wordt minstens één keer aangeroepen vanuit de subroutine READIN/READ van figuur 5.

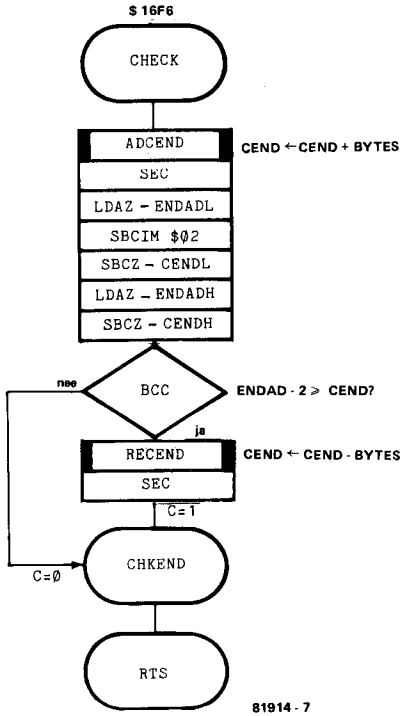
ASCII-kode van die toets omgezet in het overeenkomende datanibble. Is het geen numerieke toets, dan wordt ASHETT verlaten met een N-vlag één.
Voor de subroutine ASHETT: zie figuur 12 en punt 13 van hoofdstuk 14.
Na ASHETT en nadat is gebleken dat de N-vlag nul is staat het opgehaalde nibble rechts in de accu. Vier opeenvolgende schuifoperaties (naar links)

op de accu-inhoud hebben tot gevolg dat het zojuist ingelezen datanibble het linker nibble van de accu-inhoud is geworden. Die tijdelijk wordt opgeslagen in de geheugenplaats NIBBLE. En dan volgt nogmaals de sprong naar RECCHA en nogmaals de sprong naar ASHETT, het tweede datanibble wordt verwacht en vervolgens verwerkt. Na het kombineren van de oude en de nieuwe accu-inhoud (de instruktie ORAZ-NIBBLE) en nadat de Z-vlag één en de N-vlag nul is gemaakt is de computer klaar met BYTIN/BYT; een RTS ligt dan voor de hand. Is er, per ongeluk of expres, een niet-numerieke toets tijdens BYTIN/BYT vastgesteld, dan wordt deze subroutine verlaten met een N-vlag één en een Z-vlag nul.
N.B. De subroutine BYT is gelijk aan de subroutine BYTIN, op het wachten op de eerste numerieke toets (eerste RECCHA van figuur 6) na. BYT speelt een rol bij de Input-toetsroutine, zie punt 18.
Nu de subroutine READIN/READ van figuur 5. Die begint met de zojuist besproken subroutine BYTIN. De eerste twee datanibbles, dus de opcode van de door de gebruiker opgegeven instruktie, staan in de accu. Tenminste: als de N-vlag één is. Want anders gaan we meteen door naar RTS (N = 1; Z = Ø).
Direkt na het label READ gaat de opcode van de opgegeven instruktie naar de geheugenplaats POINTH. Uit de subroutine LENACC, die vervolgens aan het werk wordt gezet, blijkt hoe lang de instruktie is en dus of er geen, één of twee herhalingen van BYTIN in READIN optreden. De lengte van de instruktie is vertaald in de inhoud van de geheugenplaats TEMPX en in de beginwaarde van de geheugenplaats COUNT. Aan het eind van LENACC heeft Y een inhoud, gelijk aan de instruktielengte.
Indien de instruktie twee of drie bytes lang is volgt de afdruk van een spatie (PRSP) en wachten we op de opgave van het eerste of enige operand-byte (BYTIN). Dat byte gaat naar de buffer POINTL. Is de instruktie drie bytes lang, dan herhaalt zich dat spelletje; het tweede operand-byte gaat naar de buffer INH. Tot slot van het liedje (label RDA in figuur 5) krijgt X de waarde nul. Met als gevolg dat de Z-vlag één is en dat de N-vlag nul is. Indien er tussentijds een niet-numerieke toets is vastgesteld wordt READIN/READ verlaten met een N-vlag één en een Z-vlag nul. Hoe dan ook: de RTS betekent het einde van het verblijf in READIN/READ.
N.B. De subroutine READ is gelijk aan de subroutine READIN, op het wachten op het eerste byte (de opcode; de eerste BYTIN plus BMI van READIN) na. READ speelt een rol bij de Input-toetsroutine, zie punt 18.

10 Nu de **subroutine CHECK.** Die heeft iets te maken met de vraag of er voor een nieuwe toe- of tussen te voegen (kandidaat-) instruktie nog wel voldoende plaats is in het geheugen, of niet. Dat heeft verder ook alles te maken met figuur 1 van hoofdstuk 13. Er moeten, het adres ENDAD meegerekend, vier plaatsen overblijven: drie plaatsen voor het eerste gevonden label tijdens de eerste fase van het assembleren en één geheugenplaats voor het EOF-teken 77. Dat betekent dat de kwa positie op de geheugenkaart **laagste** stand van de variabele eindadreswijzer CEND die is waarbij CEND staat gericht op een adres dat twee lager is dan het

adres ENDAD. Indien CEND – rekening houdend met de lengte van de kandidaat-instruktie – lager dan deze genoemde positie komt te staan, is er geen plaats meer in het geheugen voor de kandidaat-instruktie: kandidaat afgewezen. Jammer, niets aan te doen! Of? Misschien kan ENDAD nog worden verhoogd?
De vaststelling: wel of geen plaats is vertaald in de toestand van de carryvlag na afloop van CHECK. Deze subroutine staat in figuur 7. Begonnen wordt met de subroutine ADCEND. Dit is een subroutine van de standaard-editor. Hij staat beschreven in hoofdstuk 8 van boek 2; zie pagina 145/146 en figuur 13. De variabele eindadreswijzer CEND wordt verhoogd met de lengte van een instruktie. Neem voorlopig van ons aan dat de inhoud van de geheugenplaats BYTES vlak vóór de sprong naar CHECK in overeenstemming is met de lengte van de kandidaat-instruktie.
Nadat dit is gebeurd krijgt de carryvlag een waarde die afhangt van het resultaat van inhoud ENDAD minus $Ø2 minus inhoud CEND. De instruktie BCC beslist wat de software van CHECK vervolgens te doen staat. Indien C nul is geworden is er bij het aftrekken kennelijk een borrow

★7



Figuur 7. De PME-subroutine CHECK gaat na of er nog plaats is voor een opgegeven instruktie in het door BEGAD en ENDAD vastgelegde geheugengedeelte. Het resultaat van het onderzoek is vertaald in de toestand van de carryvlag.

nodig geweest. Dit betekent dat de inhoud van ENDAD kleiner is dan de inhoud van CEND met daarbij opgeteld $Ø2. Er is dan geen plaats in het geheugen voor de kandidaat-instruktie: RTS met C= Ø.
Indien echter het resultaat van het aftrekken data nul of positieve data oplevert is CEND nog niet onder de minimumstand beland. Er is plaats voor de kandidaat-instruktie: kandidaat toegelaten en RTS met C = 1. Echter: vlak voor de RTS volgt eerst nog de subroutine RECEND. Dat is het broertje van ADCEND. Zie pagina 148 van boek 2. De variabele eindadreswijzer CEND krijgt tijdelijk de oude waarde weer terug omdat direkt na CHECK begonnen wordt met het "inschrijven" in het geheugen van de kandidaat-instruktie (zie de bespreking van de I-toetsroutine, in punt 11). Er moet eerst plaats worden gemaakt in het geheugen. En dáárvoor is het van essentieel belang dat CEND eerst nog even de oude waarde krijgt.
N.B. Merk op dat in het geval dat CHECK wordt verlaten met C = Ø geen korrektie van CEND via de subroutine RECEND plaatsvindt. Dit heeft tot gevolg dat enerzijds het EOF-teken 77 op zijn plaats blijft (er wordt immers geen instruktie aan het geheugen toegevoegd omdat er geen plaats (genoeg) voor is) en dat anderzijds CEND wel kwa inhoud groter wordt en kwa positie op de geheugenkaart lager komt te staan. Zodra er dus geen plaats meer blijkt te zijn voor een nieuwe tussen- of toe te voegen instruktie staat CEND niet langer gericht op een ardres dat één hoger is dan het adres van de geheugenplaats met daarin de EOF 77. Daar is wat aan te doen: zie punt 20.

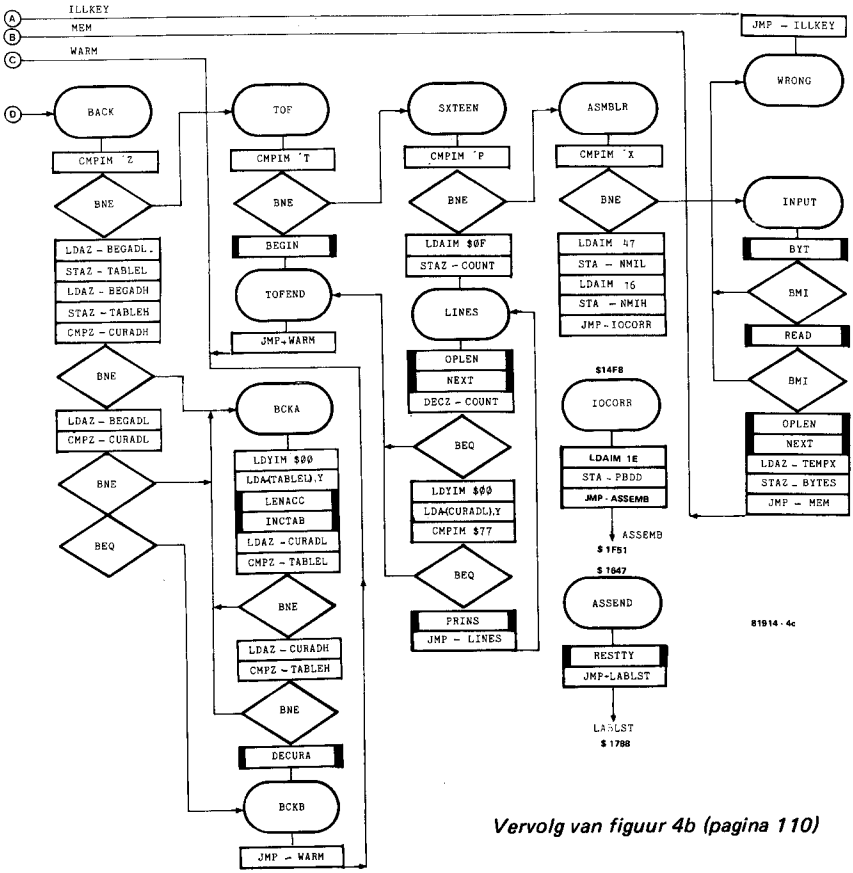
11 Zo, nu dan **de I-toetsroutine**. Dat zijn in figuur 4b de instrukties die direkt volgen op het label INSERT. Zodra blijkt dat er een toets is ingedrukt en dat het toets I is volgt er een beroep op de subroutine READIN. Die hebben we zojuist, in punt 9, besproken. De subroutine READIN wordt verlaten met:
N = Ø; Z = 1 èn
in POINTH de opcode van de te "Inserten" (tussen te voegen) instruktie;
in POINTL eventueel de eerste of enige operand van de instruktie;
in INH eventueel de tweede operand van de instruktie;
òf: N = 1, Z = Ø, indien tijdens de opgave van de instruktie een niet-numerieke toets is ingedrukt. In dat laatste geval volgt via de instruktie BMI, die direkt volgt op JSR-READIN, de sprong naar het label ILLKEY, voor de terugmelding "ILLEGAL KEY", gevolgd door de terugkeer naar het centrale label WARM.
Indien de instruktie helemaal volgens de regels is opgegeven worden de instrukties die volgen op het label MEM afgewerkt. Allereerst wordt de subroutine CHECK doorlopen. Er wordt dus gekeken of er nog wel voldoende plaats is in het geheugen voor de kandidaat-instruktie. Een en ander is in punt 10 besproken. Indien dat niet het geval is volgt via de aansluitende instruktie BCC de sprong naar het label FULL, teneinde te zorgen voor de terugmelding "FULL". Is er daarentegen nog plaats genoeg in het geheugen, dan wordt de subroutine FILLWS aangesproken: de kandidaat-instruktie is toegelaten en kan in het geheugen worden gezet.
Die subroutine FILLWS is ook al bij de standaard-editor gebruikt. Zie figuur 12 in hoofdstuk 8 van boek 2. Hij zorgt ervoor dat plaats wordt

gemaakt voor de nieuwe instruktie (subroutine DOWN). Nadat de variabele eindadreswijzer CEND is aangepast worden één, twee of drie geheugenplaatsen gevuld met de inhoud van POINTH (opcode respektievelijk eventueel de inhoud van POINTL), ((eerste) operand) respektievelijk eventueel de inhoud van INH (tweede operand). Omdat na afloop van FILLWS de Z-vlag één is voert de aansluitende instruktie BEQ ons automatisch terug naar het centrale label WARM. Omdat CURAD ongewijzigd is wordt dan de tussengevoegde instruktie afgedrukt.
N.B. Het gedeelte van de I-toetsroutine na het label MEM is tevens het tweede gedeelte van de Input-toetsroutine. Zie punt 18.

12 **De S-toetsroutine** omvat een hele waslijst aan instrukties. Dat wil zeggen: de hele rechter kolom, vanaf het label SEARCH, van figuur 4b. Nadat is vastgesteld dat toets S is ingedrukt volgt de sprong naar de subroutine READIN van punt 9. Dat betekent dus dat de op te sporen instruktie is gespecificeerd in de geheugenplaats POINTH (opcode) en eventueel, afhankelijk van de instruktielengte, verder in de geheugenplaatsen POINTL en INH. Indien er een niet-numerieke toets is ingedrukt voert de BMI, die direkt op READIN volgt, naar het label ILLKEY: er volgt de foutmelding "ILLEGAL KEY".
En daarna, vanaf het label SCAN, spelen zich de eigenlijke opsporingsakties af. Eerst gaat de opcode van de op te sporen instruktie de accu in en wordt er in LENACC gekeken hoe lang die instruktie is. Die lengte is na afloop van LENACC bekend via de inhoud van de geheugenplaats BYTES. O ja, we waren het bijna vergeten: direkt vóór SCAN (JSR-BEGIN) is de instruktiewijzer (CURAD) gericht op BEGAD: er wordt met het zoeken begonnen bij de eerste instruktie of het eerste label van het programma. De opcode van die instruktie gaat de accu in (LDA-(CURADL),Y) en wordt vergeleken met de opcode van de gezochte instruktie, dus met de inhoud van POINTH. Zijn de beide opcodes niet gelijk, dan hoeven we deze instruktie niet verder te onderzoeken; de BNE voert naar het label AGAIN, voor het onderzoek van de volgende instruktie uit het geheugen.
Door het verlagen van de inhoud van BYTES met één en een daarop volgende BEQ wordt gekeken of **de op te sporen instruktie (dus niet de onderzochte instruktie!)** één byte lang is of niet. Let wel: dit onderzoek vindt uitsluitend plaats indien de beide opcodes gelijk blijken te zijn. Is de op te sporen instruktie inderdaad één byte lang, dan volgt de sprong naar het label FOUND: de instruktie wordt mèt adres afgedrukt. Zo niet, dan wordt het volgende byte uit het geheugen opgehaald. Dit is niet perse de eerste of enige operand van de onderzochte instruktie, omdat de onderzochte instruktie slechts één byte lang kan zijn. Wèl staat het vast dat dat "volgende byte in het geheugen" wordt vergeleken met de eerste of enige operand van de op te sporen instruktie. Ook nu zijn er twee mogelijkheden: bytes ongelijk? Dan direkt dóór naar AGAIN. Bytes gelijk? Ga na of de op te sporen instruktie twee bytes lang is. Is dat het geval: dóór naar FOUND; is dat niet het geval: haal het nu volgende byte uit het geheugen op en vergelijk dat met de tweede operand van de op te sporen instruktie. Zijn ook deze twee bytes onderling gelijk, dan zijn we zonder sprongen ("branches") bij het label FOUND terechtgekomen.

(Tussenopmerking). Men had het gedeelte na het label SCAN ook anders kunnen opzetten. Na de vaststelling dat de beide opcodes aan elkaar gelijk zijn had men de beide instruktielengten met elkaar kunnen vergelijken en meteen dóór kunnen gaan naar AGAIN als dat niet het geval is. Dat spaart weliswaar tijd, maar bepáálď geen geheugenplaatsen. En het gekozen algorithme (de "aanpak") is écht waterdicht.
Direkt na het label FOUND volgt de subroutine PRINS: de op te sporen instruktie is gevonden en wordt mèt het adres van de opcode afgedrukt. Uit de software is ondubbelzinnig op te maken dat er met het opsporen wordt begonnen vanaf het begin van het programma: zie de subroutine BEGIN vlak voor het label SCAN. In hoofdstuk 13 is verteld dat men nog verderop in het geheugen naar dezelfde instruktie kan zoeken. Je moest dan de toets Y indrukken. Het is dan ook niet verwonderlijk dat direkt na PRINS de subroutine RECCA volgt. Dus dat er wordt gewacht op een ingedrukte toets. Is die ingedrukte toets de Y-toets, dan gaat PME dóór naar het label AGAIN, ter voorbereiding van het onderzoek van de volgende instruktie in het geheugen. Is het daarentegen een andere dan de Y-toets geweest, dan voert de BNE het programma naar het label DNE. Er volgt de terugmelding "DONE", ter afsluiting van de S-toetsroutine.
Direkt na het label AGAIN wordt de subroutine OPLEN aan het werk gezet. De instruktielengte van **de meest recente onderzochte instruktie** wordt bepaald en vervolgens (NEXT) krijgt de instruktiewijzer CURAD een zodanige stand dat hij (of zij? je weet dat nooit!) op de volgende instruktie in het geheugen staat gericht. Of er nog zo'n volgende instruktie is blijkt uit de stand van de N-vlag na afloop van NEXT. Deze kwestie is uitgebreid besproken bij de L-toetsroutine van punt 7. Indien er geen instrukties meer zijn te onderzoeken is het label DNE aan bod, voor de afsluitende terugmelding "DONE". U kunt uit figuur 4b opmaken dat de enige "uitweg" uit de S-toetsroutine via het label DNE loopt. Dat is in overeenstemming met wat in hoofdstuk 13 is gezegd. Namelijk dat het "searchen" pas is afgesloten na de terugmelding "DONE".

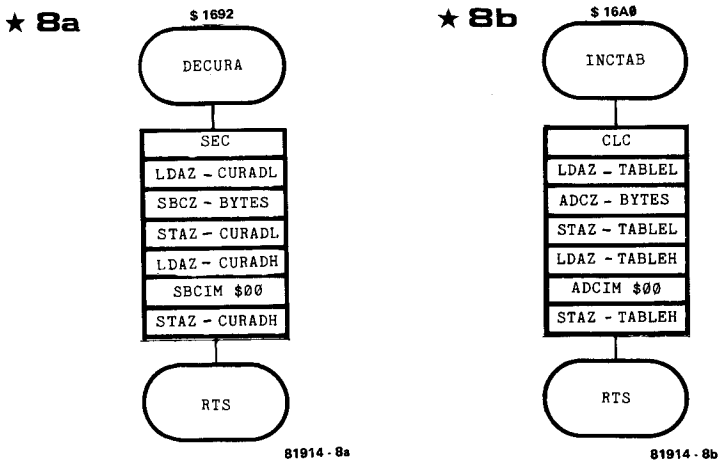
13 **De Z-toetsroutine** bestaat uit een handvol instrukties die, links in figuur 4c, direkt volgen op het label BACK. Nadat is vastgesteld dat toets Z is ingedrukt wordt de inhoud van de adreswijzer TABLE gelijk gemaakt aan de inhoud van BEGAD. Die TABLE werd totnutoe uitsluitend gebruikt bij de assembler (zie hoofdstuk 9 van boek 2), maar hebben we nu als hulpadreswijzer nodig, bij het bepalen van de lengte van de instruktie, die zich in het geheugen direkt voor de als laatste afgedrukte instruktie bevindt. Die lengte is nodig om van de inhoud van de instruktiewijzer CURAD af te trekken, zodat, na de sprong naar WARM, de vorige instruktie in het geheugen wordt afgedrukt. En dat is in overeenstemming met de instruktie-mintoetsfunktie van toets Z.
Direkt nadat TABLE aan BEGAD gelijk is gemaakt wordt er gekeken of TABLE gelijk is aan CURAD, dus of de eerste instruktie van het programma als laatste is afgedrukt. In dat geval is er geen "vorige instruktie" om af te drukken en volgt de sprong terug, via het label BCKB, naar het centrale label WARM. Met als gevolg dat opnieuw (want CURAD blijft dan ongewijzigd) die eerste instruktie wordt afgedrukt.
Hoe anders loopt het allemaal indien de instruktiewijzer CURAD niet op



Vervolg van figuur 4b (pagina 110)

Figuur 4c. Het tweede deel van de hoofdroutine van PME omvat de toetsroutines van de toetsen Z, T, P en X, alsmede de Input-toetsroutines (toetsen 0 . . . F).

het eerste adres BEGAD staat gericht. In dat geval wordt de programmalus rond het label BCKA een aantal keren doorlopen. Het is de bedoeling dat TABLE net zo lang met een bedrag, gelijk aan de lengte van de onderzochte instruktie wordt verhoogd totdat TABLE gelijk is aan CURAD. Vervolgens moet CURAD worden verlaagd met een bedrag, gelijk aan de meest recente inhoud van de geheugenplaats BYTES. Vandaar dat direkt na BCKA de opcode van de onderzochte instruktie de accu in gaat en dat de lengte van die instruktie wordt bepaald (LENACC). De inhoud van TABLE wordt nu verhoogd met een bedrag, gelijk aan die instruktielengte. Dat gebeurt in de **subroutine INCTAB** van figuur 8b.
En nu volgt het onderzoek of de verhoogde TABLE gelijk is aan CURAD of niet. Is dat niet het geval, dan gaan we terug naar BCKA voor een volgende onderzoeksronde. Is echter de nieuwe stand van TABLE gelijk



Figuur 8. De subroutines DECURA (figuur 8a) en INCTAB (figuur 8b) spelen een rol bij de Z-toetsroutine (instruktiemintoets).

aan die van CURAD, dan volgt de afwikkeling van **de subroutine DECURA van figuur 8a.** Van de inhoud van de instruktiewijzer CURAD wordt de meest recente inhoud van de geheugenplaats BYTES afgetrokken. Dat is dus hetzelfde bedrag als het bedrag dat, bij TABLE opgeteld, tot de konklusie TABLE = CURAD leidde. Nu rest nog een sprong terug, naar het centrale label WARM. Dat gaat via het tussenlabel BCKB. Direkt na WARM wordt dus de vorige instruktie afgedrukt.

14 Het verhaal van **de T-toetsroutine** is buitengewoon snel verteld. Het zijn de instrukties die in figuur 4c direkt volgen op het label TOF. Zodra PME aan de weet komt dat toets T is ingedrukt volgt de subroutine BEGIN. De instruktiewijzer CURAD wordt op BEGAD gericht en na de terugkeer naar WARM, via het label TOFEND, wordt de eerste instruktie van het programma afgedrukt. Het mag ook een label zijn, maar dat wist u al.

15 **Dan de P-toetsroutine.** Dus alle instrukties van figuur 4c die direkt volgen op het label SXTEEN. Zodra er sprake is van een ingedrukte P-toets krijgt de als instruktieteller gebruikte geheugenplaats COUNT de waarde $ØF (decimaal: 15) toegekend. Dat betekent dat er naast de als laatste afgedrukte instruktie (met op dezelfde regel op de rechter kolom
een afgedrukte P) normaal gesproken 15 instrukties worden afgedrukt. Dus in totaal 16 instrukties. Van die zestien instrukties is er dus al één afgedrukt vóór het indrukken van toets T. Veertien instrukties worden er gedurende de programmalus rond het label LINES afgedrukt en de vijftiende wordt afgedrukt direkt na de terugkeer naar het centrale label WARM.

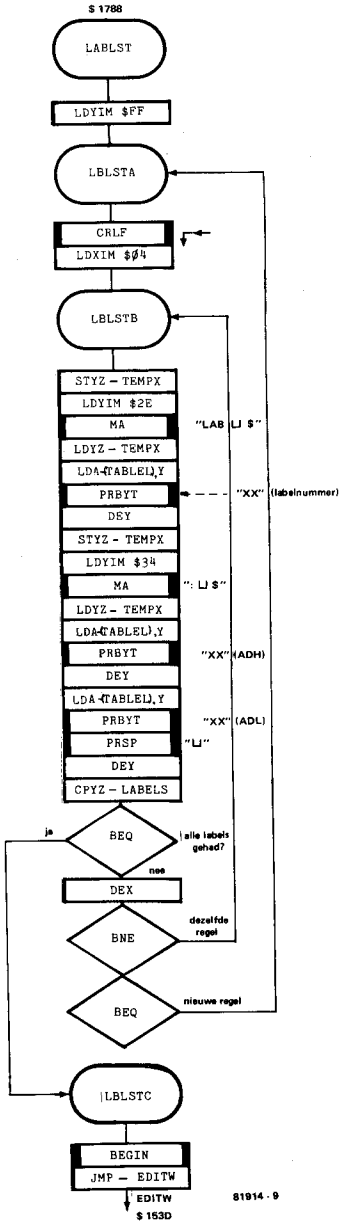
Nu de details. Direkt na LINES wordt de **lengte van de al afgedrukte instruktie** bepaald (LENACC). De nieuwe inhoud van de instruktiewijzer CURAD is in overeenstemming daarmee aangepast (NEXT) en COUNT wordt met één verlaagd. Indien dat tot een eindwaarde nul van COUNT leidt volgt via de BEQ en het tussenlabel TOFEND de sprong terug naar WARM, voor de afdruk van de vijftiende instruktie. Zolang de inhoud van COUNT nog niet nul is geworden volgt het onderzoek of de opcode van de af te drukken instruktie 77 is. Dus of de laatste, afsluitende "instruktie" van het programma aan bod is. Is dat het geval, dan wordt deze instruktie als laatste afgedrukt, dat wil zeggen: na de sprong terug naar WARM, via TOFEND.
Maar normaal gesproken wordt nu de instruktie afgedrukt (subroutine PRINS) en gaat PME terug naar LINES. De eerste vier instrukties, volgend op het label LINES, worden 15 keer doorlopen omdat je de inhoud van COUNT 15 keer kunt verlagen voordat deze nul wordt. Veertien keer wordt de rest van het programma na LINES afgedrukt en veertien keer volgt de subroutine PRINS direkt. De vijftiende doorloop geeft aanleiding tot de sprong naar WARM: de vijftiende en laatste instruktie wordt afgedrukt. Dit alles onder de aanname dat er niet tussentijds een "instruktie" met de opcode 77 is ontdekt.

16 **De X-toetsroutine** bestaat uit de instrukties van figuur 4c die direkt aansluiten op het label ASMBLR, en verder uit de drie instrukties die volgen op het label IOCORR. De toetsroutine heeft alles te maken met de voorbereiding van het assembleren van het programma. Direkt nadat er is vastgesteld dat toets X is ingedrukt wordt de NMI-sprongvektor gespecificeerd op een adres dat hoort bij het label ASSEND. Nadat het programma is geassembleerd en nadat de toets ST/NMI van het standaard-toetsenbord is ingedrukt volgt de sprong terug naar PME, naar het label ASSEND. En wat er dàn gebeurt is in punt 17 besproken.
Nadat de NMI-sprongvektor is gespecificeerd volgt de sprong naar het label IOCORR. Het richtingsregister PBDD van poort B krijgt dezelfde inhoud toegewezen als het geval is na de afwikkeling van de RESET-voorroutine van de standaard-monitor. Zie hoofdstuk 7 van boek 2. De overige I/O hoeft niet te worden aangepast. Nú weerhoudt niets PME ervan om naar de assembler te springen: JMP-ASSEMB. Dat betekent dat het programma nu gaat worden geassembleerd. Hoe dat precies in zijn werk gaat kunt u uitgebreid lezen in hoofdstuk 9 van boek 2.

17 **Na het assembleren.** Het programma bevindt zich nu "in het kraambed". Zodra het assembleren, op initiatief van toets X, is voltooid lichten de zes displays van het zeven-segmentdisplay op. U dient nu de toets ST in te drukken. Dat heeft tot gevolg dat er een NMI optreedt. De NMI-sprongvektor vertelt het programma dat de instrukties vanaf het label ASSEND moeten worden afgewerkt. En dat begint met de subroutine RESTTY. Deze subroutine is in PM gebruikt voor het herstel van de I/O zoals die geldt voor PM (èn voor PME!), nadat de cassette-subroutine DUMP of RDTAPE is afgehandeld. Voor details: zie figuur 19b en de punten 27 en 28 van hoofdstuk 14. Eigenlijk hoeft alleen PBDD de

oude waarde terug te krijgen (zie punt 16), maar dat kost meer bytes dan de aanroep van RESTTY.
We weten dat na het assembleren en na het indrukken van toets ST alle labels worden afgedrukt, die in het zojuist geassembleerde programma voorkwamen. Dat gebeurt in de **label-afdrukroutine LABLST** van figuur 9. Tijdens deze routine krijgt de adreswijzer TABLE, die in punt 13 tijdelijk als hulpadreswijzer is "geleend", zijn oorspronkelijke funktie als onderdeel van de "tabeleindadreswijzer", terug. Aan het begin van de eerste fase van het assembleren krijgt TABLE een inhoud die overeenkomt met ENDAD minus FF. De beginwaarde van de geheugenplaats LABELS is FF. De tabelwijzer TABLE + LABELS bepaalt op welke geheugenplaats – die deel uitmaakt van de "labelgeheugenruimte" – er een labelnummer, danwel de bijbehorende ADH, danwel de bijbehorende ADL wordt opgeslagen. De beginwaarde van de tabelwijzer is zodanig dat deze staat gericht op ENDAD. Na de afwerking van een label (gevonden èn opgeborgen) wordt de inhoud van LABELS met $Ø3 verlaagd. Tot zover enige herhalingsoefeningen van hoofdstuk 9; zie de pagina's 165 . . . 167 van boek 2.
In de routine LABLST van figuur 9 worden de tussenwaarden van LABELS nagebootst in het Y-register. Dus LABLST begint met het aan FF gelijk maken van de inhoud van het Y-register. De inhoud van TABLE is, zoals gezegd, ENDAD minus FF. De instrukties vanaf het label LBLSTB houden zich bezig met het in de juiste volgorde afdrukken van de diverse labelonderdelen, waarbij telkens na het afdrukken van een onderdeel de inhoud van Y met één wordt verlaagd (DEY).
Maar vóórdat label LBLSTB aan bod is volgt eerst nog een tweetal instrukties vanaf het label LBLSTA. Er wordt begonnen op het begin van een nieuwe regel en het als labelteller gebruikte X-register krijgt de beginwaarde $Ø4: op elke regel mogen maximaal vier labels worden afgedrukt.
Op twee plaatsen na het label LBLSTB wordt de inhoud van Y tijdelijk bewaard in de geheugenplaats TEMPX. Dat is nodig omdat er twee terugmeldingen door PME moeten worden "uitgeroepen". En dáárvoor is Y óók nodig. Dat is allereerst de melding "LAB $". Die wordt gevolgd door het in de accu zetten van het nummer van het (eerste) label dat tijdens de eerste fase van het assembleren is gevonden en opgeborgen. Dat gaat via de instruktie LDA-(TABLEL),Y waarbij Y een waarde heeft die gelijk is aan FF minus een aantal keren $Ø3. Dat aantal is tijdens de eerste doorloop van LBLSTB gelijk aan nul en tijdens de laatste doorloop gelijk aan het totale aantal labels.
Na het afdrukken van het labelnummer (PRBYT) volgt de terugmelding ": $". Daarna wordt eerst het linker adresbyte ADH en dan het rechter adresbyte ADL van het label afgedrukt. Nadat er vervolgens een spatie is afgedrukt (PRSP), om de labels op een regel van elkaar te scheiden, volgt de test (CPYZ-LABELS plus BEQ) of er nog labels zijn af te drukken of niet. Zo ja, dan wordt X met één verlaagd. De waarde van X geeft aan hoeveel labels er nog op de regel kunnen worden afgedrukt. En dan gaan we òf naar LBLSTA terug (ga verder met afdrukken op het begin van een nieuwe regel), òf naar LBLSTB: ga verder op dezelfde regel.
Indien alle labels zijn afgedrukt volgt de sprong naar het label LABLSTC. Nadat de instruktiewijzer CURAD op de eerste instruktie van het geas-

★ 9



Figuur 9. De label-toetsroutine LBLST wordt doorlopen nadat het programma is geassembleerd en nadat de NMI/ST-toets van het standaard-hexadecimale toetsenbord is ingedrukt. Hij zorgt ervoor dat van elke gevonden label het nummer en het bijbehorende adres wordt afgedrukt, met maximaal vier labels op een regel. De labels worden afgedrukt in een volgorde met toenemend adres.

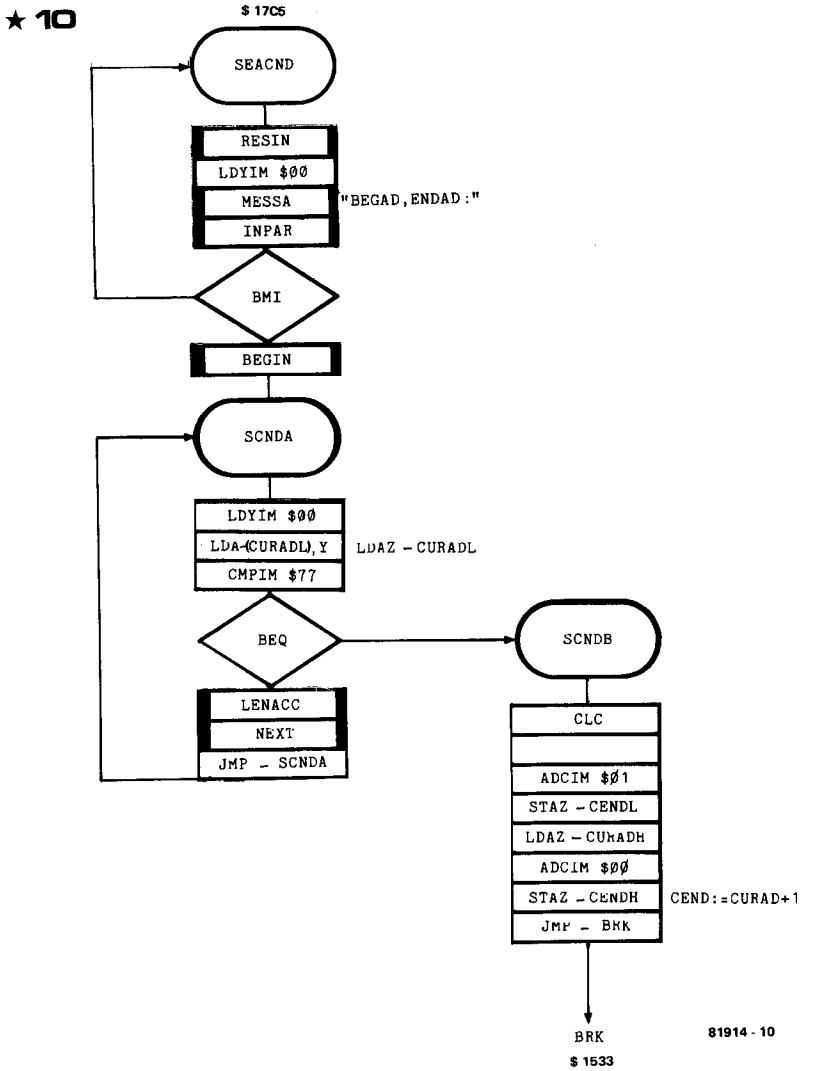
sembleerde programma is gezet (BEGIN) volgt de sprong naar het label EDITW (zie figuur 4a). Na de terugmelding "PM EDITOR" wordt de eerste instruktie van het geassembleerde programma afgedrukt.
Nu nog even die kwestie van die CPYZ-LABELS. Hoe zit dat precies? Wel, telkens nadat, tijdens de eerste fase van het assembleren, een label is gevonden, en opgeborgen, wordt de inhoud van LABELS met $Ø3 verlaagd, vanuit de beginwaarde FF. In figuur 9 wordt, na LBLSTB de inhoud van Y drie keer met één verlaagd (DEY). De eindstand van Y nadat alle labels zijn afgedrukt is dus dezelfde als de eindstand van LABELS nadat alle labels van het te assembleren programma zijn gevonden, en in de labelgeheugenruimte opgeslagen. Voor deze kwestie raadplege men óók figuur 4 op pagina 166 van boek 2.

18 Tot slot van de PME-hoofdroutine bespreken we de **Input-toetsroutine**. Zoals in hoofdstuk 13 van dit boek is uiteengezet is het niet nodig dat vooraf een niet-numerieke toets (funktietoets) wordt ingedrukt. Er kan dus zondermeer worden begonnen met de opgave van de toe te voegen instruktie, via het indrukken van minstens twee numerieke toetsen. Dit betekent dat, indien het laatste label INPUT (zie figuur 4c) van de PME-hoofdroutine wordt bereikt, de accu hoogstwaarschijnlijk de ASCII-kode van een numerieke toets zal bevatten. Dit is in overeenstemming met de regel dat numerieke data automatisch wordt behandeld als een deel van een toe te voegen instruktie, tenzij voorafgaand daaraan de toets S of I is ingedrukt. En als dàt het geval is wordt de te volgen data niet via het label INPUT binnengehaald. Zie hiertoe de punten 11 en 12 van dit hoofdstuk.
De Input-toetsroutine omvat de instrukties in figuur 4c vanaf het label INPUT. Het begint met de aanroep van de subroutine BYT. Deze laatste is gelijk aan de subroutine BYTIN, op de eerste RECCHA van BYTIN na. Die is niet nodig omdat de RECCHA na het centrale label WARM als zodanig dienst doet. Zie figuur 6 en punt 9.
Na BYT staat de opcode van de toe te voegen instruktie in de accu. De N-vlag vertelt ons of er een niet-numerieke toets is "geoogst". In dat geval voert namelijk de op BYT volgende instruktie BMI via het tussenlabel WRONG naar het label ILLKEY, voor de terugmelding "ILLEGAL KEY".
En dan is men toe aan de subroutine READ. Die valt bijna helemaal samen met READIN op de eerste twee instrukties van READIN na. Zie figuur 5 en punt 9. READ begint met het in POINTH zetten van de zojuist ingelezen opcode; vervolgens wordt de rest van de instruktie ingelezen. Een aansluitende instruktie BMI zorgt ook nu voor een eventuele terugmelding "ILLEGAL KEY".
En dan vervolgens het welbekende duo OPLEN plus NEXT. De instruktiewijzer CURAD wordt verhoogd met de lengte van de instruktie die als laatste is afgedrukt. Dus niet met de lengte van de nieuwe, toe te voegen instruktie! Dit betekent dat de nieuwe instruktie, als ie in het geheugen wordt toegelaten, op een plaats of plaatsen terechtkomt, direkt volgend op de plaats(en) van de als laatste afgedrukte instruktie. De geheugenplaats BYTES bevat de lengte van de nieuwe instruktie nadat de inhoud van TEMPX – tot standgekomen tijdens READ/READIN – in BYTES

is gekopieerd. Zodra dat gebeurd is volgt de sprong naar het label MEM. De rest van de bespreking van de Input-toetsroutine is identiek aan de tweede helft van de bespreking van de I-toetsroutine. Zie hiervoor punt 11.
Nog één opmerking. Of er nu plaats is voor de toe te voegen kandidaat-instruktie of niet – de instruktiewijzer CURAD wordt opgehoogd. Dit is de reden dat na de terugmelding "FULL" een instruktie wordt afgedrukt die hoort bij de verhoogde stand van CURAD.

19 De voorroutine die hoort bij **de lauwe start van PME** bestaat uit de instrukties die in figuur 4a volgen op het label SEMIW. Het begin van deze voorroutine is exakt gelijk aan het begin van de voorroutine EDITC, zie punt 4. Dus: de ingangsbuffers nul maken, de terugmelding "BEGAD, ENDAD:", de opgave van BEGAD en ENDAD door de gebruiker (subroutine INPAR) en het opnieuw beginnen (BMI) als er bij die opgave iets fout is gegaan. Vervolgens krijgen de geheugenplaatsen BEGADL, BEGADH, ENDADL en ENDADH een inhoud die in overeenstemming is met de twee opgegeven adressen. Verder wordt de variabele eindadreswijzer CEND op het eindadres ENDAD gericht en de instruktiewijzer CURAD wijst op het beginadres BEGAD (JSR-BEGIN). Na de sprong naar het label BRK van figuur 4a wordt de BRK-sprongvektor gespecificeerd. Na de terugmelding van "PM EDITOR" komen we terecht bij het centrale label WARM van de hoofdroutine PME.
De lauwe startingang SEMIW van PME is met name bedoeld voor het bekijken van een programma, in RAM of in EPROM, dat kant en klaar is. En met name de toetsen SP, S, L en P hebben dan een belangrijke taak te verrichten. Bij de bespreking van de genoemde toetsfunkties, in dit hoofdstuk, hebben we kunnen zien dat "het ophoudt" zodra de variabele eindadreswijzer CEND is bereikt. Welnu: het grote voordeel van de lauwe startingang van PME is dat CEND automatisch gelijk wordt gemaakt aan ENDAD – die men zèlf kan kiezen! Overigens: die lauwe start is eveneens in hoofdstuk 5 van boek 2 (standaard-editor) aan de orde gekomen. Alleen moet je daar een hele serie buffers stuk voor stuk, "met de hand", specificeren.

20 Tot slot van dit hoofdstuk de bespreking van de voorroutine die hoort bij **de warme CEND-startingang** van PME. Deze voorroutine staat in figuur 10. Het startlabel heet SEACND. Het begin ziet er net zo uit als het begin van EDITC en SEMIW. We beginnen de bespreking van figuur 10 dan ook bij het label SCNDA. Nadat de instruktiewijzer CURAD op het eerste adres BEGAD is gericht (JSR-BEGIN) wordt de opcode van de onderzochte instruktie de accu binnengehaald en gaan we na of het soms de pseudo-opcode 77 is. Zo ja: werk de instrukties vanaf het label SCNDB af en zo nee: verhoog CURAD met de lengte van de meest recente onderzochte instruktie en ga de nieuwe, door CURAD aangewezen instruktie onderzoeken op een opcode 77: LENACC, NEXT plus JMP-SCNDA.
Zodra de opcode 77 is vastgesteld (label SCNDB) wordt het adres CURAD met de opcode 77 met één verhoogd. Deze met één verhoogde waarde wordt toegekend aan de variabele eindadreswijzer CEND. Na de sprong naar het label BRK van figuur 4a volgt de specifikatie van de BRK-sprong-



Figuur 10. De voorroutine SEACND wordt doorlopen nadat PME is gestart via de warme CEND-startingang. De oorspronkelijke inhoud van de variabele eindadres-wijzer CEND wordt hersteld, mits er een "opcode" 77 voorkomt in het programma!

vektor en de terugmelding "PM EDITOR": we zijn bij het centrale label WARM van de hoofdroutine van PME aangeland. Het van de band gelezen, al dan niet geassembleerde programma heeft de oorspronkelijke BEGAD en CENDAD teruggekregen. De waarde van ENDAD hoeft niet perse gelijk

te zijn aan de oorspronkelijke waarde. Dit omdat men eventueel aan het begin van de warme CEND-start van PME een hogere ENDAD kan opgeven omdat men verwacht dat de toevoeging van instrukties aan het van de band gelezen programma dat noodzakelijk maakt. Want anders loopt men het gevaar van de terugmelding "FULL". Het wordt natuurlijk vervelend als ENDAD al op het uiterste randje van RAM-geheugen is gespecificeerd: op het adres $Ø7FF.
En CURAD? Na het afwerken van de instrukties van figuur 10 is de eindwaarde van CURAD het adres met de opcode 77. Dus niet het adres CEND! Kijk maar: bij de instrukties vanaf het label SCNDB van figuur 10 komen geen instrukties STAZ-CURADL en STAZ-CURADH voor.
U ziet in figuur 10 ook wat er gebeurt als er geen 77 voorkomt in het onderzochte programma: de programmalus rond het label SCNDA wordt oneindig lang doorlopen!
Indien u de Input- en I-toetsroutines goed hebt bestudeerd kunt u nog een andere toepassing van de warme CEND-ingang van PME bedenken. Indien namelijk een kandidaat-instruktie in het geheugen is geweigerd (terugmelding "FULL"), is CEND wèl, maar de 77 niet opgeschoven. Zie de laatste alinea van punt 10 (subroutine CHECK). Indien het nog mogelijk is om ENDAD te verhogen, dus indien er alsnog plaats kan worden gemaakt voor de kandidaat-instruktie en eventuele dáárop volgende instrukties, kan men de verhoogde ENDAD (en dezelfde BEGAD!) het beste opgeven via de warme CEND-startingang van PME. Want nadat dit is gebeurd loopt CEND automatisch weer in de pas met het adres met de opcode 77.

16

1152 bytes TM-software

De cassette-software

De ondertitel van hoofdstuk 10 van boek 3 luidt: "Data van, naar en áán de lopende band". Het zal duidelijk zijn dat daarmee in ieder geval de cassette-band is bedoeld. Maar dat "lopende band" wekt eveneens associaties op met een fabrikage-proces en dan rijst natuurlijk de vraag hoe het allemaal in zijn werk gaat. Hoe ziet de software eruit die de verzorging van een datazending naar de cassette-band tot taak heeft? Hoe leest de junior-computer een bepaalde, door de gebruiker gewenste datazending van de band? Hoe zit het hoofdprogramma van TM in elkaar?
Vragen en nog eens vragen. Met als gevolg een aantal uitgebreide antwoorden. En wel in dit hoofdstuk. Eerst gaan we de "super-subroutine" DUMP bespreken. Deze subroutine wordt door de junior-computer aan het werk gezet zodra er een datazending op de band moet worden geschreven. Hij wordt niet alleen vanuit TM maar ook vanuit het systeemprogramma PM aangeroepen; tevens is deze subroutine in eigen software toe te passen.
Het zusje van DUMP is de "super-subroutine" RDTAPE. Deze subroutine is er voor het van de band lezen van een vooraf door de gebruiker opgegeven datazending. Voor deze subroutine gelden dezelfde toepassingsmogelijkheden als voor DUMP.
Het systeemprogramma Tape Management, ook wel "Tape Monitor" genoemd, is opgebouwd rond de twee cassette-sub-routines. Het programma is een alternatief voor PM – dat nagenoeg dezelfde cassette-toetsfunkties kent – in het geval dat men afziet van een uitbreiding van de junior-computer met randapparatuur. Het programma TM is namelijk gebaseerd op

het gebruik van het standaard-toetsenbord in kombinatie met het eveneens al "in den beginne" aanwezige zesdelige zeven-segmentdisplay.
Net als in de andere software-hoofdstukken van dit boek zullen de diverse onderdelen punt voor punt worden besproken. Dit hoofdstuk kan men dus eveneens opvatten als een soort "data-zending", waarbij de punten de bytes voorstellen. En na het lezen van déze zin heeft u de 255 synchronisatie-karakters al achter de rug!

Even in het kort het programma van dit hoofdstuk: Eerst gaan we het hebben over de subroutine DUMP en de bijbehorende subroutines (je zou kunnen spreken van "sub-subroutines"; punten 1 t/m 9). Daarna volgt een uitgebreide bespreking van RDTAPE en zijn "sub-subroutines" (punten 10 t/m 19). Pas dan zijn we toe aan de hoofdroutine en de subroutines van TM (punten 20 t/m 28). En tot slot volgt een "datazending" over de PLL-testsoftware: de punten 29 en 30. Maar nu dus eerst DUMP.

1 Het gedetailleerde stroomdiagram van **de subroutine DUMP** staat in de figuren 1a (eerste helft) en 1b (tweede helft). Laten we maar eens gewoon bovenaan figuur 1a beginnen, dus met de instrukties vanaf het label DUMP tot aan het label DUMPT. Wat zien we? Vier geheugenplaatsen, namelijk HIGHER, LOWER, FIRST en SECOND, krijgen een bepaalde inhoud toegewezen. Deze geheugenplaatsen spelen een belangrijke rol bij de totale tijdsduur (de bittijd), alsmede de relatieve tijdsduren 3600 Hz en 2400 Hz, van de naar de cassette-band verzonden bits, die deel uitmaken van een databyte of van de ASCII-kode van een verzonden karakter. In de punten 3 en 4 gaan we uitgebreid op deze kwestie in.
Pas vanaf het label DUMPT in figuur 1a bemoeit de software zich met het dresseren van de I/O (poorten A en B van de PIA). Men zou intuïtief kunnen verwachten dat dit als allereerste gebeurt. Waarom eigenlijk eerst die vier tijd-geheugenplaatsen specificeren? Het antwoord op deze vraag is niet zo moeilijk te bedenken. De subroutine DUMP wordt niet alleen vanuit TM of PM aangeroepen, maar is ook toe te passen in ego-software. Indien de gebruiker een bittijd wil kiezen die afwijkt van die van de junior-computer, dus indien de gekozen snelheid waarmee bits worden geschreven en gelezen afwijkt van die van de junior-computer, hoeft de gebruiker slechts de aangepaste acht instrukties vanaf het label DUMP in zijn software op te nemen en de subroutine DUMPT (niet DUMP!) aan te roepen. In punt 9 is de lees- en schrijfsnelheid aangepast aan het KIM-computersysteem.
De eerste negen instrukties vanaf het label DUMPT van figuur 1a hebben te maken met het vastleggen van de I/O ten dienste van het schrijven van

de datazending op de cassetteband. Wat heeft dit voor gevolgen voor poort B? Alle acht poortlijnen van poort B worden als uitgang geschakeld. Verder krijgt PBD een inhoud $47. Dit heeft tot gevolg dat:

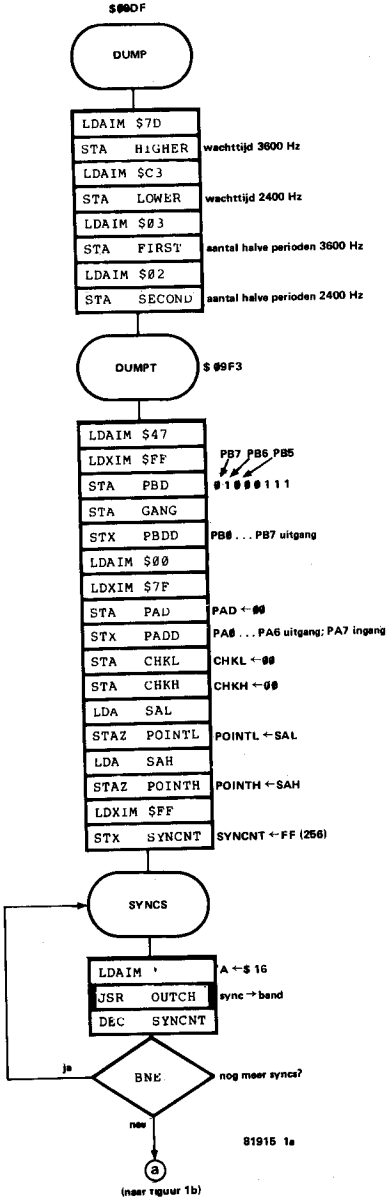
- de als cassette-dataingang/uitgang gebruikte poortlijn PB7 als uitgang is geschakeld; op de uitgang staat een logische nul;
- poortlijn PB6 als relais-stuuruitgang is geschakeld; op de uitgang staat een logische één. Dus is transistor T2 gesperd. Het INPUT-relais Re1 is niet bekrachtigd en de groene INPUT-LED D4 is gedoofd (zie figuur 2 op pagina 13 van boek 3);
- poortlijn PB5 eveneens als relais-stuuruitgang is geschakeld. Op de uitgang staat een logische nul. Transistor T3 geleidt. Het OUTPUT-relais Re2 is bekrachtigd en de rode OUTPUT-LED D5 licht op;
- (omdat geldt: PB4 PB3 PB2 PB1 = ØØ11 = DCBA) de displayschakelaar (IC7 van de standaard-junior-computer; zie figuur 9 op pagina 110 van boek 2) in de neutrale stand staat. Hoe verder ook de segmentschakelaar (poort A) is gedefinieerd – geen enkele van de zes displays kan oplichten en geen enkele van de toetsen kan op indrukken worden betrapt. Een en ander is in overeenstemming met het feit dat tijdens het op de band schrijven van een datazending het display is gedoofd (je ziet alleen de rode OUTPUT-LED oplichten) en de toetsen van het standaard-toetsenbord inaktief zijn;
- de als seriële datauitgang gebruikte poortlijn PBØ als zodanig funktioneert, ondanks dat we nu niets met randapparatuur (met uitzondering van de cassette-recorder!) te maken hebben. Op uitgang PBØ komt een logische één te staan. Dit komt overeen met de rusttoestand van deze seriële datalijn; er wordt niets verzonden, en wàt er aan interessants wordt verzonden loopt via poortlijn PB7.

En dan nu poort A. De als segmentschakelaar gebruikte poortlijnen PAØ . . . . PA6 zijn als uitgang geschakeld en de als seriële dataingang gebruikte poortlijn PA7 is als zodanig geschakeld. De inhoud nul, die aan PAD is toegekend betekent dat alle segmenten in principe, wat poort A betreft, kunnen oplichten, ware het niet dat de toestand van PBD dat onmogelijk maakt. Indien er geen karakters door het randapparaat worden verstuurd heeft het bit PA7 een logisch nivo één. Dat komt overeen met het logische rustnivo van deze seriële datalijn. Overigens: aan eventuele, door een randapparaat naar de junior-computer verzonden karakters heeft de software van DUMP, en trouwens ook die van TM, geen enkele boodschap.

We gaan verder met figuur 1a. Allereerst zij opgemerkt dat de inhoud van de geheugenplaats GANG uiteindelijk altijd gelijk is aan de inhoud van PBD. Waarom die geheugenplaats zo nodig moet komt in punt 2 aan de orde. Verder krijgen de geheugenplaatsen CHKH en CHKL, voor de berekening van de kontrolebytes, een beginwaarde nul. Nu zijn we toe aan de eigenlijke voorbereiding van het op de band schrijven van een datazending. De inhoud van de geheugenplaatsen SAL en SAH, voor het eerste adres van het te verzenden datablok, wordt gekopieerd in de inhoud van de buffers POINTL respektievelijk POINTH, die samen de adreswijzer POINT vastleggen. En de laatste instrukties vóór het label SYNCS van figuur 1a houden zich bezig met het $FF maken van de inhoud van de als syncteller gebruikte geheugenplaats SYNCNT.

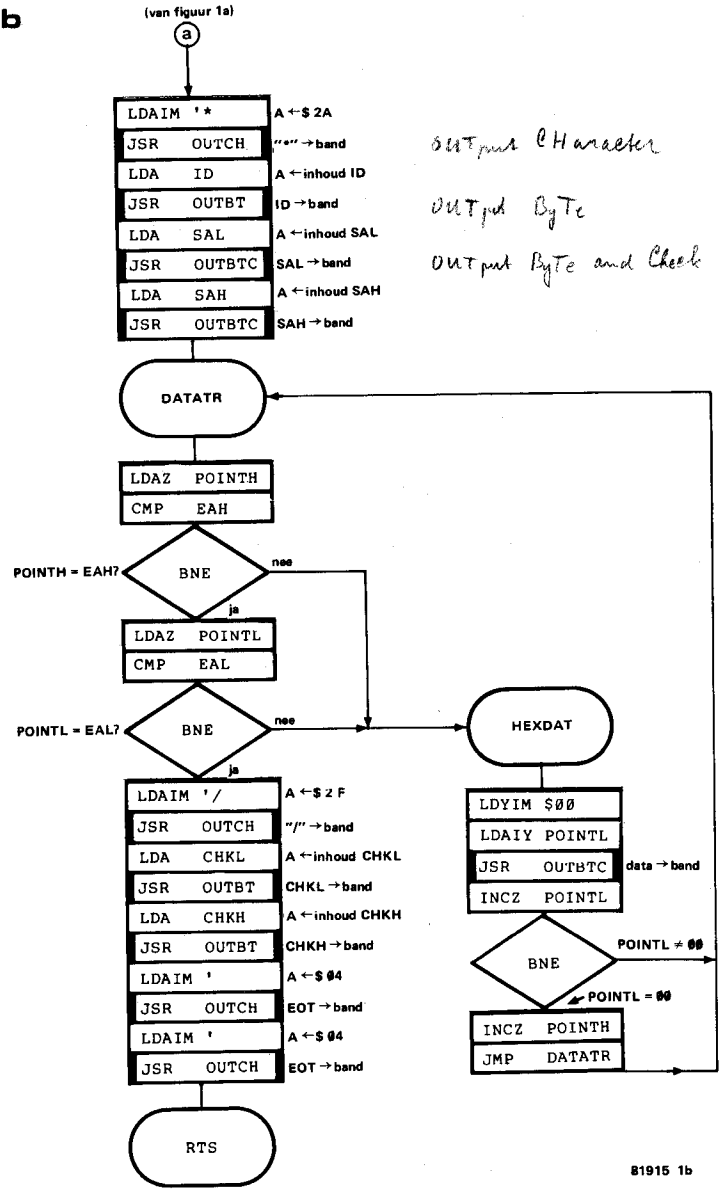
N.B. Voortzetting van de bespreking van DUMP: zie punt 7.

★ 1a



Figuur 1a. Het eerste gedeelte van het gedetailleerde stroomdiagram van de "super-subroutine" DUMP/DUMPT.

★ 1b



Figuur 1b. Het tweede gedeelte van het gedetailleerde stroomdiagram van DUMP/ DUMPT.

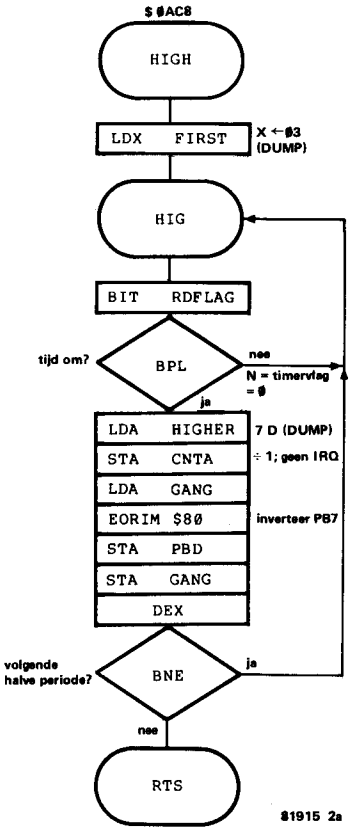
2 Nu eerst een aantal subroutines van DUMP die te maken hebben met het verzenden naar de band van een bit respektievelijk een ASCII-karakter, een datanibble en een databyte. Eerst de verzending van een bit.
Op verscheidene plaatsen in boek 3 is het verzenden van bits naar de band al aan de orde gekomen. Ons toespitsend op de software, waarmee een en ander moet worden gerealiseerd vatten we het verhaal van de bittijden en van de hoge en lage frekwentie samen:

- Poortlijn PB7 is het "doorgeefluik" voor het door de junior-computer via software gemaakte uitgangssignaal, dat bestemd is voor de cassette-recorder.
- Een bit is vertaald in een signaal dat altijd begint met een blokgolf van een hoge frekwentie (3600 Hz) en altijd eindigt met een blokgolf met een lage frekwentie (2400 Hz).
- Het logische uitgangsnivo van PB7 moet op tijdstippen worden geïnverteerd (één wordt nul, nul wordt één) die samenhangen met de blokgolffrekwentie die op dat moment aan bod is.
- Een **bit één** levert een uitgangssignaal op dat bestaat uit **drie halve perioden 3600 Hz, gevolgd door vier halve perioden 2400 Hz.**
- Een **bit nul** levert een uitgangssignaal op dat bestaat uit **zes halve perioden 3600 Hz, gevolgd door twee halve perioden 2400 Hz.**

We gaan nu twee subroutines bespreken:
a. de subroutine HIGH van figuur 2a verzorgt een uitgangssignaal dat bestaat uit drie halve perioden 3600 Hz;
b. de subroutine LOW van figuur 2b verzorgt een uitgangssignaal dat bestaat uit twee halve perioden 2400 Hz.
We stellen vast dat het voor de verzending van een bit één nodig is dat één keer de subroutine HIGH moet worden aangeroepen, direkt gevolgd door twee keer achterelkaar de subroutine LOW. Voor de verzending van een bit nul moet eerst twee keer achter elkaar een beroep worden gedaan op de subroutine HIGH, direkt gevolgd door één keer de subroutine LOW. Verder is het belangrijk om vast te stellen dat, ongeacht het logische nivo van het te verzenden bit, de subroutines HIGH en LOW elkaar tijdens het verzenden van data of ASCII-karakters-sec altijd afwisselen.

De subroutine HIGH van figuur 2a, in detail besproken. Wat moet er gebeuren? Er moet drie keer een halve 3600 Hz-periodetijd worden gewacht. Telkens na zo'n tijdje wachten moet het logische nivo van PB7 worden geïnverteerd. Het aantal keren wachten ziet men terug in de inhoud van de geheugenplaats FIRST, die aan het begin van HIGH wordt gekopieerd in het X-register.
Bij het gedurende een halve periodetijd 3600 Hz (HIGH) of 2400 Hz (subroutine LOW, zie punt 3) wachten wordt dankbaar gebruik gemaakt van de mogelijkheden van de interval-timer van de PIA, zoals beschreven in hoofdstuk 6, in boek 2. De timer wordt gestart (begin aftellen) door $7D in de geheugenplaats CNTA te schrijven. Er is dus sprake van een deelfaktor 1 en na de time out ontstaat er geen IRQ. Zodra er sprake is van een time out wordt de wachtlus, bestaande uit de instrukties BIT-RDFLAG en BPL van figuur 2a, doorbroken. Vrijwel direkt na een time out wordt de timer opnieuw gestart. Een korte tijd later wordt het logische nivo van poortlijn PB7 op zijn kop gezet. Geïnverteerd dus. En daarmee

★ 2a

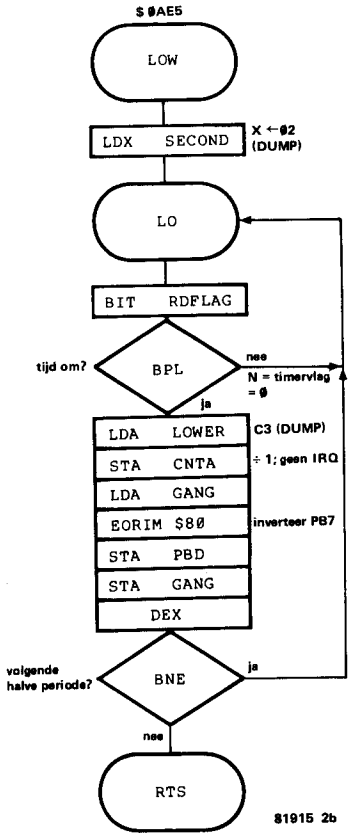


Figuur 2a. De subroutine HIGH verzorgt — bij een bit nul twee keer achter elkaar doorlopen — de 3600 Hz-fase van de verzending van een bit naar de band.

zijn we middenin die GANG-geschiedenis beland.
Uit het wezen van de EOR-funktie (zie pagina 66 van boek 1) kan men opmaken dat de instruktie EOR #8Ø inderdaad tot gevolg heeft dat bit 7 van de accuinhoud wordt geïnverteerd, en dat de andere zeven bits ongewijzigd zijn. Waarom dan niet
LDA-PBD
EOR #8Ø
STA-PBD
in plaats van de instrukties
LDA-GANG
EOR #8Ø
STA-PBD
STA-GANG
van figuur 2a? Wel, heel eenvoudig omdat het lezen van een als uitgang

geschakelde poortlijn altijd tot gevolg heeft dat een bit één wordt binnengehaald! En dat kan natuurlijk nooit de bedoeling zijn. Vandaar de noodzaak van de "schaduw-geheugenplaats" GANG en vandaar dat op een aantal plaatsen in de software een instruktie STA-PBD altijd vergezeld gaat met de instruktie STA-GANG.
Omdat, vanuit een beginwaarde $Ø3 voor X, de instrukties na het label HIG van figuur 2a drie keer worden doorlopen (vlak voor de BNE gaan we X verlagen via DEX) wordt er tijdens de subroutine HIGH drie keer de interval-timer gestart. Dat is dus dik in orde. Echter: er wordt óók maar drie keer de inhoud van PB7 geïnverteerd en als je drie halve periodes van 3600 Hz moet wachten moet dat toch vier keer gebeuren? Denk daarbij aan de lagere-schoolsommen van het type: als er langs een weg van honderd meter een rij bomen moet worden geplant met om de tien meter een boom, hoeveel bomen heb je dan nodig? Elf, en geen tien! Kortom: hoe zit dat?
Heel eenvoudig. Tijdens de laatste doorloop van de instrukties na het label HIG is X vóór de laatste DEX Ø1. Na het starten van de interval-timer en na het "omturnen" van PB7 en na de DEX wordt de subroutine HIGH verlaten (RTS). We hebben gezien dat het aantal keren dat CNTA tijdens HIGH wordt beschreven wèl in overeenstemming is met het aantal halve perioden. Dit betekent dat het wachten op de time out nà het verlaten van HIGH het afwerken van de laatste halve periode inhoudt. Het eerstvolgende starttijdstip van de interval-timer – opnieuw in de subroutine HIGH, danwel in de subroutine LOW – vindt plaats vlak voor de eerstvolgende inversie van PB7. Dit laatste tijdstip betekent de voltooiing van een vorig aantal halve perioden (3600 Hz of 2400 Hz) en tevens het begin van een nieuw aantal halve perioden, van 3600 Hz.
Dit betekent dat de instruktie BPL van figuur 2a pas dan niet meer tot springen (naar HIG) aanleiding geeft indien er een time out van de interval-timer is vastgesteld, die hoort bij de laatste vorige halve periode.
Hoe zit het precies met de wachttijden? Bij het starten van het aftellen wordt de interval-timer geladen met $7D. Het duurt dus 7 x 16 + 13 + 1 = 126 μs voordat er een time out optreedt. Nemen we aan dat er een vast tijdsverschil is tussen de start van de interval-timer en het ompolen van PB7. Dat tijdsverschil is 10 μs: de tijd, nodig voor de uitvoering van de instrukties LDA-GANG, EOR #8Ø en STA-PBD. De tijd tussen twee opeenvolgende inversies van PB7 is dus gelijk aan de tijd tussen twee opeenvolgende starts van de interval-timer.
En hoe lang duurt dàt? In ieder geval 126 μs plus de tijd die nodig is voor de uitvoering van de instrukties LDA-HIGHER en STA-CNTA. Dus minstens 126 + 8 = 134 μs. Daar komen nog een paar μs bij omdat de wachtlus BIT – RDFLAG + BPL niet op exakt hetzelfde tijdstip hoeft te worden doorbroken als op het tijdstip van de time out. Maximaal komt er 7 μs bij: de tijd, nodig om de genoemde wachtlus één keer volledig te doorlopen.
Dus de tijd tussen twee opeenvolgende inversies van PB7, de halve periodetijd, ligt ergens tussen de 134 en 141 μs. Een periodetijd 3600 Hz komt overeen met 277,8 μs, derhalve een halve periodetijd met 138,9 μs. Het verschil tussen de teorie en de software-praktijk is verwaarloosbaar klein.

★ 2b



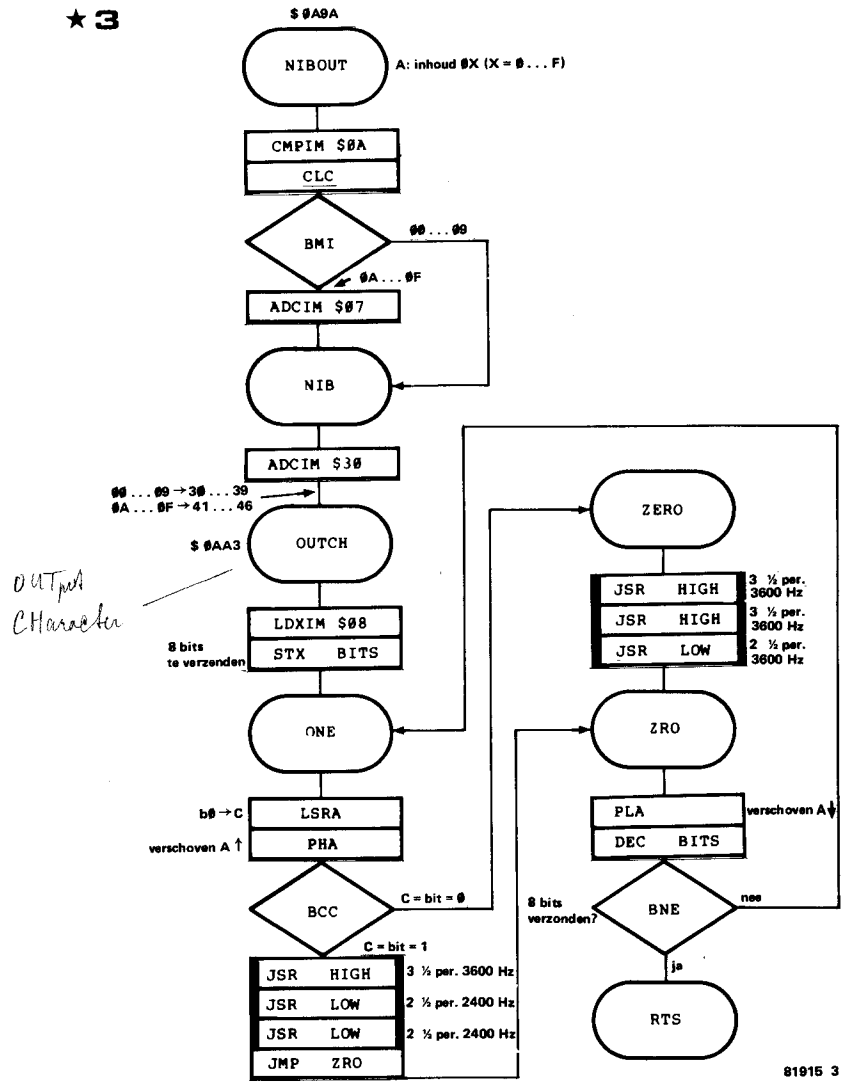
Figuur 2b. De subroutine LOW verzorgt — bij een bit één twee keer achter elkaar doorlopen — de 2400 Hz-fase van de verzending van een bit naar de band.

3 De bespreking van **de subroutine LOW** van figuur 2b kunnen we erg kort houden omdat deze geheel analoog is aan die van de subroutine HIGH van figuur 2a en punt 2. Alleen gaat het nu om twee halve perioden 2400 Hz. Vandaar dat het X-register nu aan het begin van LOW de waarde $Ø2 krijgt toegekend, namelijk de inhoud van de geheugenplaats SECOND. De wachtlus BIT – RDFLAG + BPL, met X gelijk aan $Ø2, wordt doorbroken zodra de laatste halve periode van het vorige aantal halve periodes 3600 Hz (subroutine HIGH) of 2400 Hz (subroutine LOW) is voltooid. Vrijwel direkt daarna, 8 à 15 μs later (met maximaal 7 μs voor het te laat ontdekken van de time out), wordt de interval-timer opnieuw gestart. En weer 10 μs later wordt PB7 omgeturnd. Na het verlagen van X (DEX) herhaalt het beschreven spelletje zich nog één keer. Maar op de time out, ten gevolge van de laatste start van de interval-timer, wordt niet gewacht: RTS. Deze time out wordt nà de afsluitende RTS ontdekt tijdens de wachtlus

BIT – RDFLAG + BPL van de direkt aansluitende of eerstvolgende subroutine HIGH of LOW.
Even de tijden checken. De interval-timer krijgt een beginwaarde $C3, namelijk de inhoud van de geheugenplaats LOWER. Het duurt dus 13 x 16 + 3 + 1 = 196 μs voordat er een time out optreedt. Daarbij moeten we nog 8 μs optellen: de tijd, nodig voor de uitvoering van de instrukties LDA-LOWER en STA-CNTA (zie ook HIGH; punt 2). Dus dan zitten we totaal op 196 + 8 + 204 μs. Nemen we weer aan dat de software van LOW er maximaal 7 μs te laat achter komt dat er een time out is, dan komen we uit op 204 à 211 μs. Een periodetijd 2400 Hz komt overeen met 416,7 μs, een halve periodetijd 2400 Hz derhalve met 208,3 μs. Ook nu is de overeenkomst groot tussen de teorie en de praktijk.

4 **Enige algemene opmerkingen over HIGH en LOW.** Allereerst zij opgemerkt dat bij de subroutines HIGH en LOW de nadruk ligt op het om de halve periodetijd "ompolen" van PB7. Het absolute nivo is volslagen onbelangrijk. Het gaat om de frekwentie-informatie, dus om de tijd tussen twee opeenvolgende inversies van PB7.
Een aanroep van de subroutine HIGH wordt altijd direkt gevolgd door òf nog een HIGH, òf een subroutine LOW. Dit omdat de verzending van een bit naar de band altijd begint met minstens één HIGH en altijd eindigt met minstens één LOW. Dit betekent dat het wachten op de time out na de laatste halve periode 3600 Hz van HIGH voornamelijk plaatsvindt tijdens de BIT – RDFLAG + BPL-wachtlus van de aansluitende subroutine HIGH of LOW.
Bij LOW ligt dat anders. Zodra de enige of laatste subroutine LOW (hangt ervan af welk bit is verzonden) is afgewerkt, moet de laatste halve periode 2400 Hz worden afgewerkt. De bijbehorende time out moet worden vastgesteld in de eerstvolgende BIT – RDFLAG + BPL-wachtlus van de eerstvolgende subroutine HIGH. Het verblijf in die wachtlus zal echter korter duren dan in het geval van de overgang van HIGH naar LOW (verzending van één bepaald bit), waar we het een alinea geleden over hadden. Waarom? Omdat er intussen instrukties zijn afgewerkt die in het teken staan van de voorbereiding van het volgende te verzenden bit (label ONE in figuur 3; zie punt 5), datanibble (label NIBOUT) of ASCII-karakter (label OUTCH in figuur 3). Het zal duidelijk zijn dat dat aantal instrukties niet zodanig veel tijd in beslag mag nemen dat er al een time out is opgetreden voordat de eerstvolgende subroutine HIGH wordt aangeroepen. Er is ongeveer 200 μs beschikbaar en dat is genoeg voor de uitvoering van pakweg 50 instrukties van elk 4 μs. Dat is meer dan voldoende. Gelukkig is dat allemaal niet het geval met die dreiging van de overschrijding van de tijdslimiet.
We gaan nu kijken hoe de bits, bytes en ASCII-karakters stuk voor stuk naar de cassette-band worden gestuurd.

5 **De subroutine NIBOUT/OUTCH** van figuur 3 houdt zich bezig met de verzending naar de cassette-band van alle acht bits van een ASCII-karakter, waarvan de kode in de accu staat. Die taak wordt uitgevoerd door de instrukties van figuur 3 vanaf het label OUTCH. In sommige gevallen wordt daaraan voorafgaand het programma vanaf het label NIBOUT

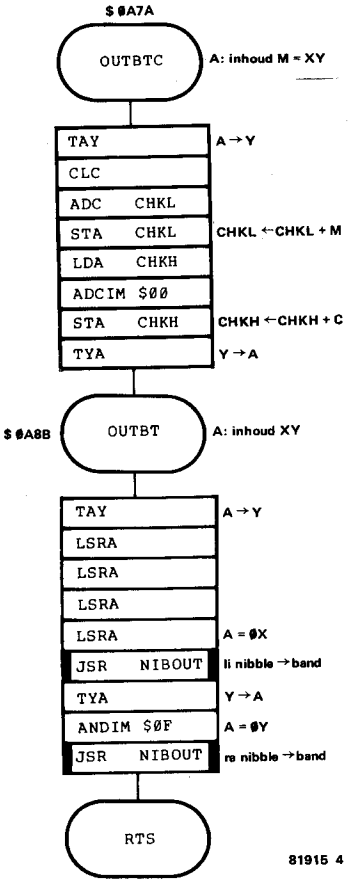


Figuur 3. Van de subroutine NIBOUT/OUTCH neemt de tweede helft, namelijk de instrukties vanaf het label OUTCH, de verzending naar de band van een ASCII-karakter voor zijn rekening. Het NIBOUT-gedeelte wordt doorlopen tijdens de verzending van een datanibble en zorgt ervoor dat het datanibble wordt omgezet in de bijpassende ASCII-kode.

van figuur 3 afgewerkt. Laten we dat deel eerst maar eens in detail gaan bespreken.
Een databyte gaat naar de band in de vorm van twee zendingen: voor elk datanibble één. De subroutine NIBOUT neemt de verzending van één datanibble voor zijn rekening. Voordat men toe is aan de eigenlijke verzending (alles in figuur 3 vanaf het label OUTCH) moet dat datanibble worden omgezet in de bijbehorende ASCII-kode.
Vlak voor de sprong naar NIBOUT is de inhoud van de accu gelijk aan ØX, met voor X de mogelijkheden Ø . . . F. Waarom dat zo is kan men aan de weet komen in punt 6 en figuur 4 (subroutine OUTBTC/OUTBT). Voor nibbles Ø . . . 9 geldt een ASCII-kode 3Ø . . . 39 en voor nibbles A . . F de ASCII-kode 41 . . . 46. In het ene geval moet dus $3Ø bij de inhoud van de accu worden opgeteld en in het andere geval wordt de accu opgehoogd met $37. Dus er komt altijd $3Ø bij en soms óók nog $Ø7. De instruktie CMP # ØA scheidt de bokken Ø . . . 9 van de schapen A . . . F.
De subroutine OUTCH begint met het toekennen van een inhoud $Ø8 van de als bitteller gebruikte geheugenplaats BITS. En dan neemt de software van OUTCH acht keer achter elkaar een loopje met het label ONE van figuur 3. Telkens gaat er een bit naar de band. Eerst gaan alle bits van de accu-inhoud een plaatsje naar rechts (LSRA). Het meest rechtse bit bØ beïnvloedt de carryvlag. Daarna wordt de **verschoven** accu-inhoud tijdelijk op de stapel gezet (PHA). Nu is het zover dat het bit kan worden verzonden. De carryvlag (BCC) zegt hoe dat moet gebeuren, want die heeft een stand die overeenkomt met de waarde (Ø of 1) van het te verzenden bit. Er volgt nu òf één keer een HIGH en twee keer een LOW (bit 1), òf twee keer een HIGH en één keer een LOW (label ZERO; bit nul). In ieder geval is het eind van het liedje (label ZRO) dat de verschoven accu-inhoud weer van de stapel wordt gehaald (PLA) en dat deze opnieuw kan worden verschoven (sprong naar ONE) voor de verzending van het volgende bit. Mits er natuurlijk nog minstens één bit te verzenden is. Want anders zijn we klaar.
N.B. Het verhaal van de subroutines HIGH en LOW is beschreven in de punten 2, 3 en 4.

6 **De subroutine OUTBTC/OUTBT** van figuur 4 zorgt ervoor dat een databyte naar de cassette-band wordt verzonden in de vorm van twee opeenvolgende verzendingen van een datanibble. Die taak wordt verricht door de instrukties vanaf het label OUTBT van figuur 4. In sommige gevallen moet er echter eerst nog wat anders gebeuren: Het 8-bits getal, gevormd door het databyte, moet worden opgeteld bij het 16-bits getal, gevormd door de inhoud van de geheugenplaatsen CHKH en CHKL. Dat heeft te maken met de aanpassing van de kontrolebytes CHKH en CHKL. Die checksom-toestanden worden uitgevoerd door de instrukties vanaf het label OUTBTC tot aan het label OUTBT, in figuur 4.
We gaan ervan uit dat vlak voor de aanroep van de subroutine OUTBTC het te verzenden databyte in de accu staat. Dat byte wordt meteen na het begin opgeborgen in het Y-register (TAY). Vervolgens wordt dit byte opgeteld bij het getal (CHKH, CHKL) De bekende standaard-zestien bits-optelling dus. Na het weer ophalen van de oorspronkelijke accu-inhoud (TYA) is het OUTBTC-gedeelte afgewikkeld.

★ 4



Figuur 4. Van de subroutine OUTBTC/OUTBT neemt de tweede helft, namelijk de instrukties vanaf het label OUTBT de verzending naar de band van een databyte voor zijn rekening. Dat gebeurt in de vorm van twee opeenvolgende verzendingen van een datanibble. Het gedeelte vanaf OUTBTC wordt doorlopen indien de inhoud van de geheugenplaatsen CHKL en eventueel CHKH moet worden aangepast, in overeenstemming met de getalwaarde die het te verzenden databyte vertegenwoordigt.

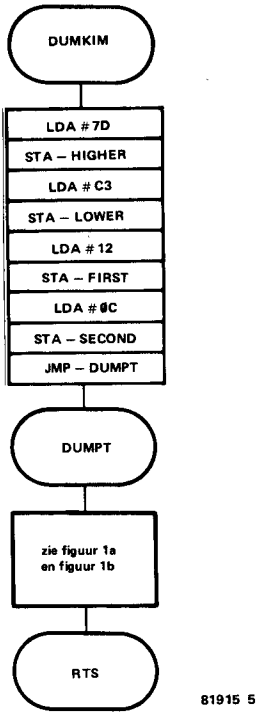
De subroutine OUTBT begint eveneens met het opbergen van de accu-inhoud in het Y-register. Daarna volgen vier instrukties LSRA. Met als gevolg dat het linker nibble van de nieuwe accu-inhoud nul is en het rechter nibble gelijk is aan het oorspronkelijke linker nibble. Dàt nibble wordt nu verzonden: de subroutine NIBOUT van punt 5 en figuur 3. Daarna krijgen we de oorspronkelijke accu-inhoud terug (TYA) en maken het linker nibble nul via de bekende truuk AND # ØF. Rest de verzending van het rechter nibble (subroutine NIBOUT) en een RTS, want we zijn klaar.

7 (vervolg van punt 1) Bij de bespreking van de subroutine DUMP/DUMPT waren we bij het label SYNCS van figuur 1a terecht gekomen. Het gaat om de verzending van 255 syncs (= synchronisatiekarakters) naar de band. Dat betekent dus: 255 keer de accu laden met de ASCII-kode $16 van de sync, 255 keer dit karakter (het officiële ASCII-karakter SYN) naar de band zenden (OUTCH; zie punt 5 en figuur 3) en 256 keer de inhoud van SYNCNT met één verlagen. De 256-ste keer vormt voor de aansluitende instruktie BNE aanleiding om het programma, onderaan figuur 1a, te verlaten en verder te gaan met het programma, bovenaan in figuur 1b.
We weten uit hoofdstuk 11 van boek 3 dat na de synchronisatiekarakters het data-beginkarakter naar de band gaat. Het hoeft dan ook geen verbazing op te wekken dat, bovenaan in figuur 1b, de accu wordt geladen met $2A, de ASCII-kode van "*", en dat daarop de sprong naar de subroutine OUTCH volgt. En wat is er dàn aan de orde? Achtereenvolgens gaan we naar de band: het programmanummer (ID) en de adresbytes SAH en SAL van het eerste adres van het te verzenden datablok worden naar de band gestuurd. In alle drie gevallen gaat het om de verzending van een databyte. Dus in ieder geval de subroutine OUTBT wordt doorlopen. De databytes SAH en SAL moeten worden betrokken bij het CHKH/CHKL-gebeuren. Vandaar in dat geval de aanroep van de subroutine OUTBTC, zie figuur 4 punt 6.

8 Het **laatste gedeelte van de subroutine DUMP** bestaat uit de instrukties vanaf het label DATATR in figuur 1b. De naam zegt het al: "datatransport". Want we zijn nu toe aan de verzending van het bij de datazending horende datablok. Dus alle data vanaf het adres SA tot het adres EA, oftewel tot en met LA, met LA gelijk aan EA-1. Het adres van de te verzenden data wordt aangewezen door de welbekende adreswijzer POINT. Die is indertijd (zie figuur 1a en punt 1) aan SA gelijk gemaakt.
De instrukties, direkt na het label DATATR gaan na of POINT soms gelijk is aan EA. Zo ja, dan volgt de afwikkeling van het "staartstuk" van de datazending. Daarover straks meer. Zo nee, dan volgt de sprong naar het label HEXDAT. De inhoud van de geheugenplaats, aangewezen door POINT, gaat de accu in en, na het aanpassen van CHKL en eventueel CHKH, naar de band: het beroep op de subroutine OUTBTC. Daarna wordt de adreswijzer POINT gericht op de volgende geheugenplaats en gaan we terug naar DATATR, mogelijk met als resultaat een nieuwe verzending van een databyte.
Zodra het hele datablok van de datazending naar de band is gestuurd staat de BNE die in figuur 1b volgt op de instruktie CMP-EAL, niet langer op springen (naar HEXDAT). De datazending moet worden afgerond met achtereenvolgens het data-eindkarakter "/" (ASCII-kode $2F), de eindstand van CHKL (via de subroutine OUTBT, dus zonder aanpassing van CHKL en eventueel CHKH), de eindstand van CHKH en twee eindezendingstekens EOT (ASCII-kode $Ø4). En dan zijn we klaar met de verzorging van de datazending. En zijn we dus eveneens klaar met de bespreking van de "super-subroutine" DUMP, op één klein dingetje na.

9 Het is al eerder opgemerkt: de subroutine DUMP, danwel de subroutine DUMPT is óók te gebruiken, dus in te passen in zelf te ontwerpen software. Indien men uitgaat van een verzendsnelheid die overeenkomt met die van de junior-computer kan men volstaan met:
JSR-DUMP.
Indien men echter een afwijkende verzendsnelheid kiest (en dan waarschijnlijk een lagere, want de betrouwbaarheid van de data-overdracht neemt sterk af naarmate de verzendsnelheid groter wordt), moet men de inhoud van minstens twee van de vier geheugenplaatsen HIGHER, LOWER, FIRST en SECOND wijzigen. In wezen moet men alle vier geheugenplaatsen opnieuw specificeren en naar de subroutine DUMPT springen. Dus niet naar DUMP! In het geval dat men met KIM-snelheden wil werken krijgt men niet te maken met een subroutine DUMP, maar met **de subroutine DUMKIM** van figuur 5. Eerst worden de vier genoemde geheugenplaatsen gespecificeerd, dan volgt de sprong naar de subroutine DUMPT, die we uitgebreid hebben besproken. We zien dat de inhoud van de geheugenplaatsen HIGHER en LOWER ongewijzigd is. Dat klopt,

★ 5



Figuur 5. De subroutine DUMKIM is gelijk aan de subroutine DUMP, maar dan aangepast aan de voor de KIM-computer geldende bandlees- en schrijfsnelheid.
Overigens: veel KIM-systemen werken met een schrijf/leessnelheid die gelijk is aan die van DUMP/DUMPT en RDTAPE: de zogenoemde "Hypertape"-software.

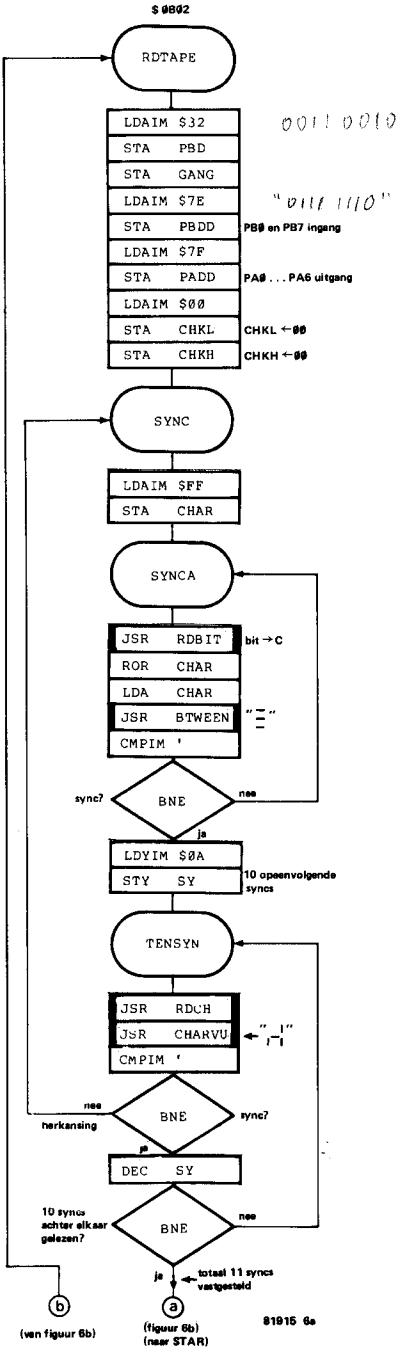
omdat de uitgangssignalen, behorend bij het verzenden naar de band van een bit, van de KIM en van de junior-computer gelijkvormig zijn (zie figuur 6 van pagina 93 van boek 3). De inhoud van FIRST en SECOND zijn echter zes keer zo hoog, omdat het bij de KIM zes keer zo traag toegaat.

10 We zijn toe aan de bespreking van de **"super-subroutine" RDTAPE,** ten behoeve van het van de cassette-band lezen van een door de gebruiker vooraf gespecificeerde datazending en het vervolgens in het geheugen van de junior-computer schrijven van het bij die datazending horende datablok. Het stroomdiagram van RDTAPE staat in de figuren 6a (eerste helft) en 6b (tweede helft). In dit punt bespreken we de eerste instrukties van figuur 6a.
De instrukties vanaf het label RDTAPE tot aan het label SYNC staan voor het overgrote deel in het teken van het vastleggen van de I/O: de poorten A en B van de PIA. We zien dat aan PBD de inhoud $32 wordt toegekend en aan PBDD de inhoud 7E. Dit betekent dat:

- de als cassette-dataingang/uitgang gebruikte poortlijn PB7 als ingang is geschakeld; op die ingang staat een logische nul;
- de als seriële datauitgang gebruikte poortlijn PBØ nu – in tegenstelling tot de situatie bij DUMP – als ingang is geschakeld, dus in wezen is uitgeschakeld;
- poortlijn PB6 als relais-stuuruitgang is geschakeld. Op de uitgang staat een logische nul. Dat heeft tot gevolg dat transistor T2 geleidt. De groene INPUT-LED D4 licht op. Het INPUT-relais Re1 is bekrachtigd;
- poortlijn PB5 eveneens als relais-stuuruitgang is geschakeld. Op de uitgang staat een logische één. Dit heeft tot gevolg dat transistor T3 niet geleidt. Dus spert. De rode OUTPUT-LED D5 is gedoofd en het OUTPUT-relais Re2 is niet bekrachtigd;
- (omdat geldt: PB4 PB3 PB2 PB1 = 1ØØ1) de displayschakelaar (IC7 van de standaard-junior-computer; zie figuur 9 op pagina 110 van boek 2) in een stand staat die inhoudt dat het display Di6 – dat is de meest rechtse van de zes – kan oplichten. Van hoofdstuk 11 van boek 3 weten we dat óók Di5 zo af en toe een oplichtende rol speelt. Over deze kwestie meer bijzonderheden in punt 15, bij de bespreking van de subroutine CHARVU/VU;

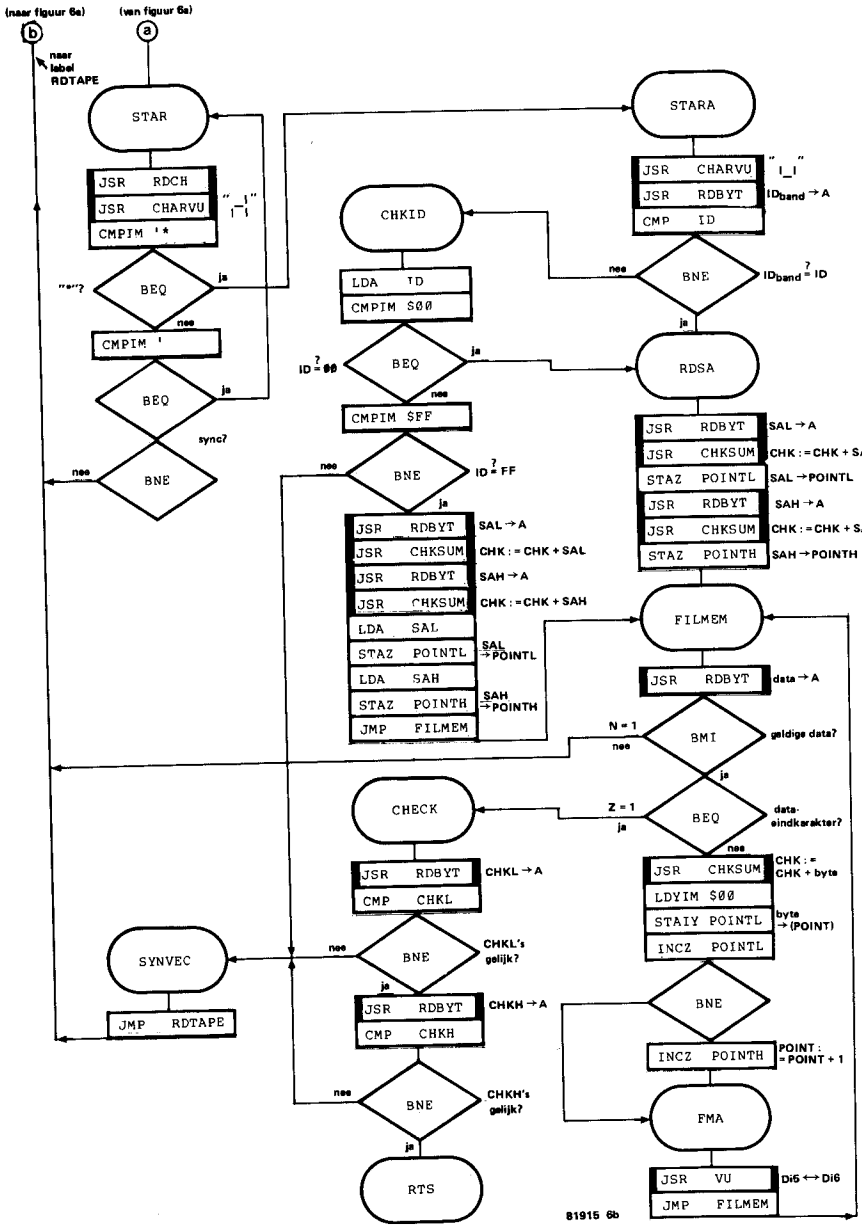
En dan nu poort A. Aan PADD wordt een inhoud $7F toegekend, zo blijkt uit figuur 6a. Dat betekent dat de als segmentschakelaar gebruikte poortlijnen PAØ . . . PA6 als uitgang zijn geschakeld en verder dat de als seriële dataingang gebruikte poortlijn PA7 als zodanig is geschakeld. Echter: de software van RDTAPE heeft verder geen boodschap aan eventuele, door een randapparaat naar de junior-computer verzonder karakters. Die software is nu uitsluitend geïnteresseerd in karakters, die door het "randapparaat" cassette-recorder worden verzonden. En dat loopt niet via PA7 maar via PB7.
In figuur 6a komen we géén instruktie STA-PAD tegen. Hoe zit dat? Wel, heel eenvoudig. Het register PAD krijgt pas een bepaalde inhoud toegekend tijdens het verblijf in de subroutine BTWEEN (zie punt 16) of in de subroutine CHARVU/VU van punt 15. De inhoud van PAD is namelijk niet altijd dezelfde.

★ 6a



Figuur 6a. De eerste helft van het gedetailleerde stroomdiagram van de "super-subroutine" RDTAPE.

★ 6b

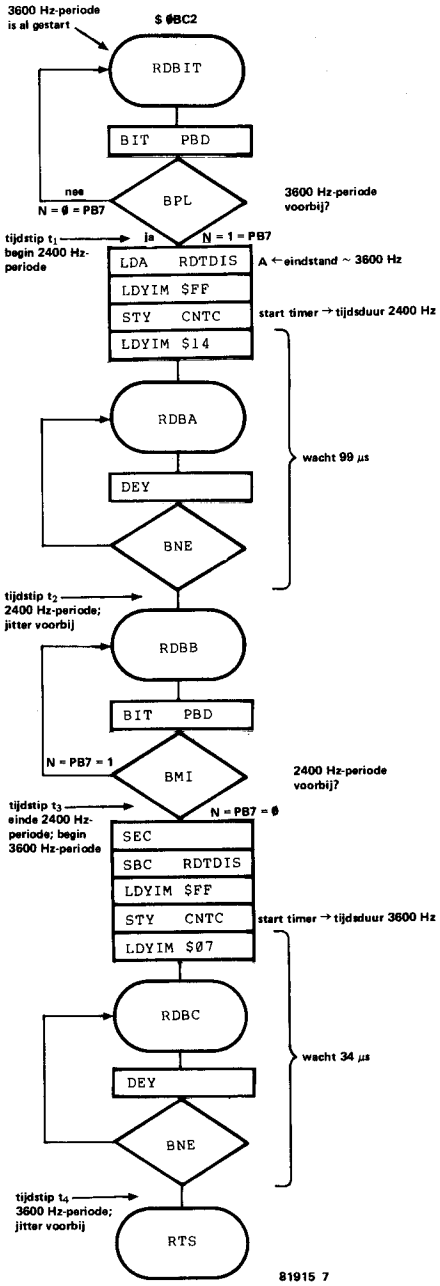


Figuur 6b. De tweede helft van het gedetailleerde stroomdiagram van RDTAPE.

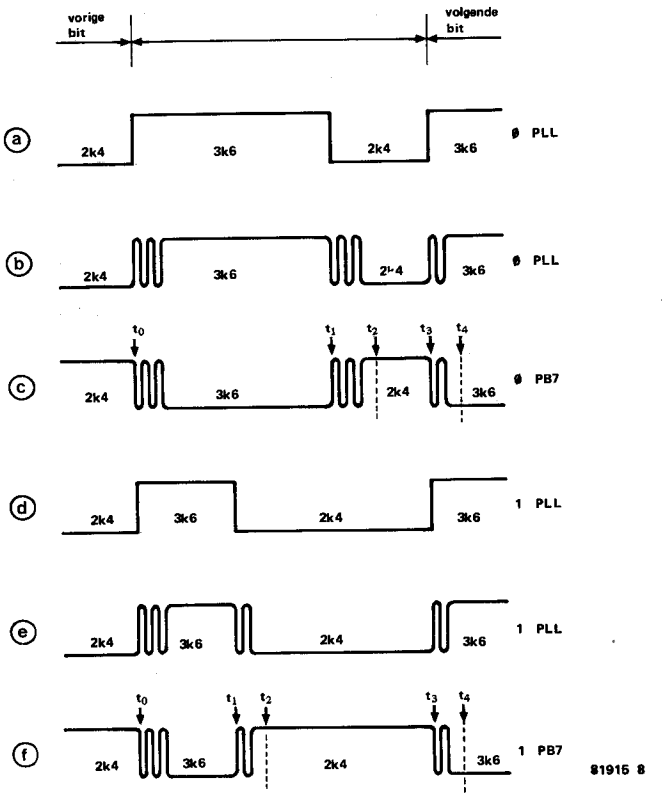
Voordat we verder gaan met de bespreking van RDTAPE zullen nu eerst alle, door RDTAPE gebruikte subroutines worden besproken.
N.B. Vervolg bespreking van RDTAPE in punt 17.

11 De meest elementaire subroutine van RDTAPE is wel **de subroutine RDBIT** van figuur 7. Deze subroutine zorgt ervoor dat een bit van de band wordt gelezen en dat na afloop van deze subroutine de carryvlag ons vertelt of het zojuist ingelezen bit een één of een nul is. Deze subroutine gaan we nu in detail bespreken. Helaas is dat een vrij ingewikkelde kwestie.
Voordat we RDBIT in detail gaan bespreken gaan we het eerst maar eens hebben over de zes pulsdiagrammen van figuur 8a t/m figuur 8f. Zes logische "heen-en-weer-plaatjes" dus. De figuren 8a (bit nul) en 8d (bit één) komen ons bekend voor van boek 3, de hoofdstukken 10 en 11. Het gaat in beide gevallen om een stukje uitgangssignaal van de PLL in het ideale geval, dus zonder dat de ook al in boek 3 genoemde zogenaamde "PLL-jitter" optreedt. In werkelijkheid ziet het PLL-uitgangssignaal eruit als geschetst in figuur 8b (bit nul) en figuur 8e (bit één). De PLL-jitter is aangegeven met een paar "trillingen" tussen de nivo's één en nul. Dus net als bij de kontaktdender van toetsen (zie figuur 8 op pagina 107 van boek 2). In werkelijkheid ziet het er allemaal nog wat ingewikkelder uit. Maar waar het om gaat is dat er een paar opeenvolgende wisselingen van logisch nivo optreden die niets te maken hebben met het einde van een zojuist aangebroken 3600 Hz-periode of met het einde van een zojuist aangebroken 2400 Hz-periode.
Voordat het PLL-uitgangssignaal naar de als ingang geschakelde poortlijn PB7 gaat wordt het eerst nog geïnverteerd; vandaar dat we bij de bespreking van RDBIT te maken krijgen met de figuren 8c en 8f.
Het totale ingangssignaal op PB7 bestaat uit een aaneenschakeling van stukjes figuur 8c en stukjes figuur 8f. Die aaneenschakeling hangt samen met de opeenvolgende waarden van de bits die van de band worden gelezen. Hóe de bittrein er ook precies uitziet, één ding is zeker: altijd wisselen de 3600 Hz-perioden (begin van een bit) en de 2400 Hz-perioden (tweede fase van een bit) elkaar af. Deze wetenschap is essentieel voor een goed inzicht in de werking van RDBIT.
En dan nu RDBIT. Figuur 7 dus. Het begint met een wachtlus: de instrukties BIT-PBD en BPL. Er wordt net zo lang gewacht totdat de N-vlag, dus bit 7 van PBD, dus PB7, één wordt. Bekijken we de plaatjes van de figuren 8c en 8f goed, dan zien we dat het één worden van PB7 gebeurt aan het eind van een 3600 Hz-periode en dus aan het begin van een 2400 Hz-periode. We zitten dan dus al op de overgang van de eerste fase van een bit naar de tweede fase. Dat gebeurt op een tijdstip t1. Meteen daarna volgt de instruktie LDA-RDTDIS: het timer-dataregister wordt gelezen. Er wordt een momentopname gemaakt van de interval-timer, en wel een opname in de accu.
Nu heeft het lezen van de timer alleen dan zin indien de interval-timer ooit, in het recente verleden, is gestart. Welnu, dat is inderdaad gebeurd. De stand van de timer is vrijwel direkt na afloop van de 3600 Hz-periode gelezen, op een tijdstip t1. Op het tijdstip t0, aan het begin van de 3600 Hz-periode, is de timer voor het laatst gestart. Dat wil zeggen: op

★7



Figuur 7. De subroutine RDBIT is er voor het lezen van de band van één bit. Na afloop is de carryvlag gelijk aan het ingelezen bit.



Figuur 8. Het PLL-uitgangssignaal voor een bit nul (ⓐ . . . ⓒ) en voor een bit één (ⓓ . . . ⓕ). De software van RDTAPE (subroutine RDBIT) berust op de nivo's op poortlijn PB7. Derhalve zijn de plaatjes ⓒ en ⓕ van belang.

een tijdstip, vóór de sprong naar RDBIT. En hóe is dat dan tot stand gekomen? Antwoord: tijdens de laatste fase van de uitvoering van RDBIT en dat is tijdens de vorige aanroep van RDBIT. En hier stuiten we op een wezenlijk kenmerk van RDBIT: Tijdens elke doorloop van RDBIT wordt een bit ingelezen en in de carry-vlag gezet. De laatste fase van RDBIT houdt tevens een voorbereiding in voor de uitvoering van de volgende RDBIT; de eerste fase van RDBIT is voorbereid tijdens de laatste fase van de vorige RDBIT. En hoe komt dat? Dat komt omdat we te maken hebben met een bittrein van bits van de band, en dus met een keiharde regelmaat van 3600 Hz – 2400 Hz – 3600 Hz – enzovoorts.
Terug naar figuur 7. Na het lezen, in de accu, van de 3600 Hz-eindstand wordt de interval-timer opnieuw gestart. De instrukties LDY #FF en STY-CNTC houden in dat, vanuit een beginwaarde FF, de inhoud van het timer-dataregister om de 64 μs met één wordt verlaagd. Zie hoofdstuk 6 van boek 2. Na de time out treedt geen IRQ op.

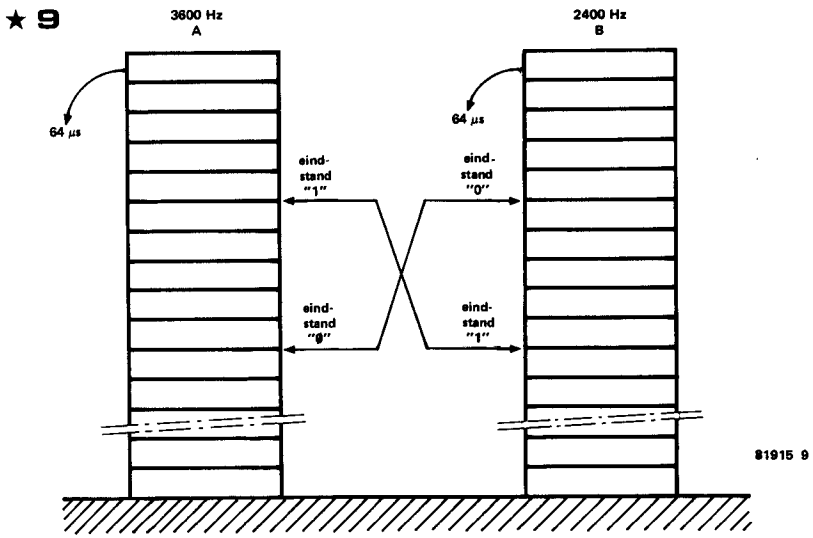
We zijn nu toe aan het label RDBA van figuur 7. Het nut van de wachtlus DEY + BNE rond het label RDBA is dat er gewacht wordt totdat de PLL-jitterperiode – als gevolg van de overgang van 3600 Hz naar 2400 Hz – gegarandeerd voorbij is. Er wordt dus gewacht totdat tijdstip t2 van figuur 8c of 8f is aangebroken. Doen we dit niet, dan zou de vòlgende wachtlus – te weten de wachtlus rond het label RDBB, die vaststelt wanneer de 2400 Hz-periode is afgelopen – vroegtijdig worden doorbroken, met alle nare gevolgen vandien. Hiermee wordt dus de tegenhanger van een valse start, namelijk een valse finish, voorkomen.
Zodra, op het tijdstip t3, de 2400 Hz-periode voorbij is levert de instruktie BMI, die direkt volgt op de BIT-PBD van het label RDBB, geen sprong meer op. We zijn dan toe aan de buitengewoon interessante instrukties SEC en SBC-RDTDIS, die we apart, in punt 12, zullen behandelen. Hier volstaan we met de opmerking dat na afloop van deze instrukties de carryvlag gelijk is aan het zojuist van de band gelezen bit. Nadat dit is gebeurd wordt de interval-timer opnieuw gestart, op dezelfde manier als al eerder tijdens RDBIT is gebeurd. Dit is de voorbereiding voor het begin van een nieuwe, toekomstige RDBIT waar we het eerder over hadden. Immers: de interval-timer wordt gestart op (nagenoeg) het tijdstip t3.
De laatste instrukties van RDBIT, vanaf het label RDBC, zorgen ook nu voor een vertraging. Met als gevolg dat de eerste wachtlus van de eerstvolgende RDBIT wordt bereikt op een tijdstip dat de PLL-jitter, die gepaard gaat met de overgang van 2400 Hz op 3600 Hz, gegarandeerd voorbij is. Dus de eerste wachtlus van de volgende RDBIT wordt op zijn vroegst bereikt op een tijdstip t4.
Over tijdstippen gesproken: Opgemerkt zij dat **het tijdstip t3 van het ingelezen bit is op te vatten als het tijdstip t0 voor het volgende in te lezen bit**. Zie de figuren 8c en 8f!

12 Nu nog de bespreking van de instrukties SEC en SBC-RDTDIS van RDBIT (figuur 7). We merken op dat de SEC ervoor zorgt dat vóór het aftrekken de carry één is en dus de borrow nul: er zijn "geen oude schulden te vereffenen". Als gevolg van de instruktie SBC-RDTDIS wordt van de accu-inhoud, dus van de eindstand van de timer na afloop van de 3600 Hz-periode, de eindstand van de timer na afloop van de 2400 Hz-periode afgetrokken. Immers: wat er van de inhoud van de accu moet worden afgetrokken komt tot stand door het lezen van de timer via SBC-RDTDIS.
Zowel aan het begin van een 3600 Hz-periode als aan het begin van een 2400 Hz-periode start het aftellen vanaf een beginstand $FF. Om de 64 μs wordt de stand met één verlaagd. Het is belangrijk om vast te stellen dat hoe langer een 3600 Hz-periode of een 2400 Hz-periode duurt, des te lager de bijbehorende eindstand van de timer is.
Even terug naar het aftrekken van daarnet. Als de inhoud van de accu groter is dan of gelijk is aan de inhoud van het timer-dataregister op het tijdstip van SBC-RDTDIS, levert het resultaat van het aftrekken van een carry C = 1 op. Dat komt omdat er niet hoeft te worden geleend. Dus de borrow is nul. En er geldt: carry = $\overline{\text{borrow}}$. Als de inhoud van de accu groter is dan de inhoud van het timer-dataregister, is de 3600 Hz-periode korter geweest dan de 2400 Hz-periode. En dàt betekent dat (zie figuur 8)

we te maken hadden met een bit één.
Geheel analoog kan men aantonen dat we in het geval van een carry C = Ø, als gevolg van het aftrekken, te maken hebben met een bit nul. Dus:

C = 1 → bit 1
C = Ø → bit Ø

Toen één van de schrijvers nadacht over die droge, vervelende carry-geschiedenis, zag hij één van zijn kinderen spelen met Lego-blokken. En toen werd figuur 9 geboren. Stelt u zich twee torens A en B voor. Beide zijn opgebouwd uit 256 ($FF!) blokken. Tijdens het afwerken van een 3600 Hz-periode wordt van toren A om de 64 μs een blok verwijderd. Hetzelfde gebeurt met toren B tijdens het afwerken van een 2400 Hz-periode. Nadat er een 3600 Hz-periode en een 2400 Hz-periode voorbij zijn is de toren A hoger dan toren B in het geval van een bit één: A min B = (een) positief (hoogteverschil). Toren A is lager dan toren B in het geval van een bit nul: A min B = (een) negatief (hoogteverschil).
N.B. De jitter-wachtlussen van RDBIT hebben geen invloed op de beide timer-eindstanden.
We hebben te maken met nominale tijdsverhoudingen 3600 Hz/2400 Hz van 2:1 of 1:2. Het resultaat van het aftrekken is dus altijd òf ondubbelzinnig "dik positief", òf "dik negatief", zelfs als men rekening houdt met een niet optimaal afgeregelde PLL. Bijvoorbeeld vanwege een afwijkende bandsnelheid. Voor deze kwestie: zie hoofdstuk 11 van boek 3.

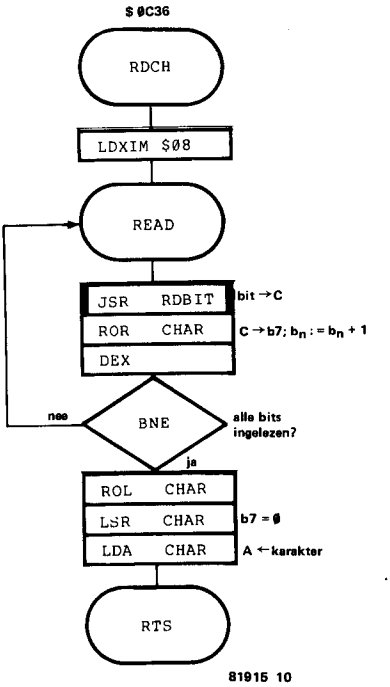


Figuur 9. Dit torenmodel is van toepassing op de werking van de subroutine RDBIT van figuur 7. Er geldt de — op het eerste gezicht merkwaardige — regel: hoe lager, hoe langer.

Uit enig rekenwerk – dat wij u zullen besparen – volgt dat één bit totaal ca. 1250 μs duurt:
bit nul: 833 μs 3600 Hz + 417 μs 2400 Hz;
bit één: 417 μs 3600 Hz + 833 μs 2400 Hz.
Een bittijd van 1250 μs komt overeen met een baudrate van 800 bits per sekonde.
Omdat de timer om de 64 μs met één wordt verlaagd zullen er gedurende 833 μs zo'n 13 verlagingen optreden en gedurende 417 μs 6 verlagingen met één. Dit betekent (beginwaarde $FF, dat is decimaal 256) dat er nooit het gevaar dreigt dat er een time out optreedt. Anders gezegd: er dreigt nooit het gevaar dat de torens van figuur 9 volledig worden afgebroken. Men kan gerust de zes keer tragere KIM-datazendingen inlezen. Want of je nou van 256 13 of 6 x 13 = 78 aftrekt – de eindstand van de interval-timer is gegarandeerd positief.
Er is nog één vraag rond de RDBIT te beantwoorden. Voor het eerste van de band gelezen bit is er geen voorgaande RDBIT waarin het begin van een 3600 Hz-periode wordt vastgesteld in de vorm van het starten van de interval-timer. Is dat nou niet vervelend? Helemaal niet, want dat eerste bit hoort bij het eerste, van de band gelezen synchronisatie-karakter. En als het misschien fout gaat met dat eerste bit (er is altijd een kans dat het wèl goed gaat), gaat het misschien fout met het eerste synchronisatie-karakter. En wat dàn nog!? Er zijn er nòg 254!
Tot slot van dit punt nog een opmerking die te vergelijken is met punt 4 (de elementaire subroutines HIGH en LOW van DUMP). Het zal duidelijk zijn dat de tijd die de uitvoering van de laatste instrukties van RDBIT kost (na het opnieuw starten van de timer), opgeteld bij de tijd, die de uitvoering van instrukties vergt die vóór de volgende RDBIT moeten worden uitgevoerd, niet zo groot mag worden dat de 3600 Hz-periode al voorbij is. Anders gezegd: Er zijn pakweg 400 μs beschikbaar voor de laatste vier instrukties van RDBIT (inklusief de RTS) plus de instrukties vóór de volgende aanroep van RDBIT. Het blijkt dat die tijd bij lange na niet wordt gebruikt.

13 Na het inlezen van een bit (RDBIT) volgt het inlezen van een ASCII-karakter. Dus het inlezen van acht bits achter elkaar. Met andere woorden: we zijn toe aan de bespreking van **de subroutine RDCH** van figuur 10.
Van DUMP herinneren we ons dat van een te verzenden karakter als eerste het meest rechtse bit bØ naar de band gaat en als laatste het meest linkse bit b7. Zie figuur 3 en de bespreking van de subroutine OUTCH, in punt 5. Dat betekent dat, als we acht bits achter elkaar van de band lezen die allemaal bij hetzelfde ASCII-karakter horen, als eerste bit bØ wordt opgevist en als laatste bit b7. Daarmee ligt de gang van zaken in RDCH vast.
Het begint met de toekenning van een inhoud $ØØ8 aan het als bitteller gebruikte X-register. En dan volgt acht keer achter elkaar de uitvoering van de instrukties JSR-RDBIT (lees het bit van de band), ROR-CHAR (schuif alle bits van de geheugenplaats CHAR een plaatsje naar rechts en maak b7 gelijk aan C) en DEX (voorbereiding volgende bit).
Zodra alle bits zijn ingelezen volgen de instrukties ROL-CHAR en LSR-

★ 10

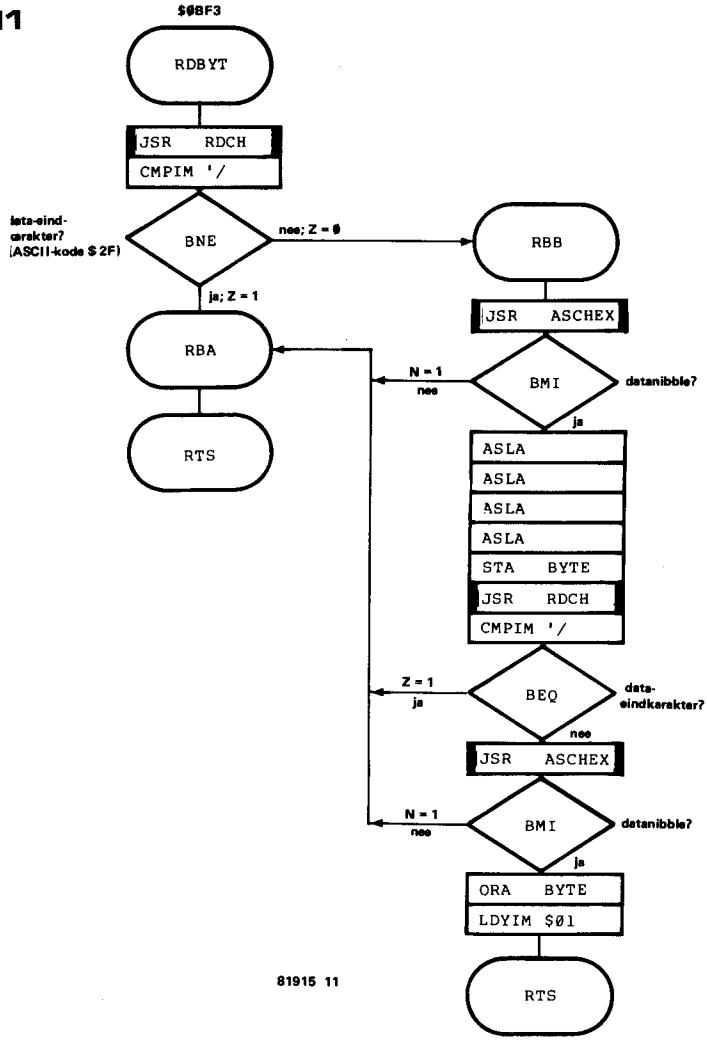


Figuur 10. De subroutine RDCH zorgt voor het lezen van de band van een ASCII-karakter.

CHAR. Dat heeft tot gevolg dat het bit b7 van CHAR gegarandeerd nul is. Dit bit b7 kàn als pariteitsbit worden gebruikt, maar dat doen we niet. Bij de verzending naar de band van ASCII-karakters is die b7 nul gemaakt. De inhoud van de geheugenplaatsen CHKH en CHKL is dáárop gebaseerd. Het gegarandeerd nul maken van b7 heeft tot gevolg dat een – door welke omstandigheden dan ook – ingelezen b7, gelijk aan één, géén problemen geeft met de eindwaarde van CHKH en van CHKL (zie punt 19). Indien daarentegen een datazending wordt gelezen waar wèl sprake is van de mogelijkheid dat b7 "officieel" één kan zijn (en dat dus de inhoud van CHKH en CHKL daarmee in overeenstemming is) zal het altijd nul zijn van b7 bij het verlaten van RDCH tot gevolg hebben dat er een grote kans is dat eindwaarden CHKH/CHKL niet overeenstemmen en dat dus de subroutine RDTAPE nooit zal worden verlaten (zie punt 19).
De laatste twee instrukties van RDCH zorgen voor het kopiëren van CHAR in de accu (LDA-CHAR) en de afronding (RTS).

14 Na het inlezen van een bit (punten 11 en 12) en van een ASCII-karakter (punt 13) ligt het voor de hand om ons bezig te gaan houden met het inlezen van een databyte, in de vorm van het twee keer achter elkaar inlezen van een nibble. Laten we datgene dat voor de hand

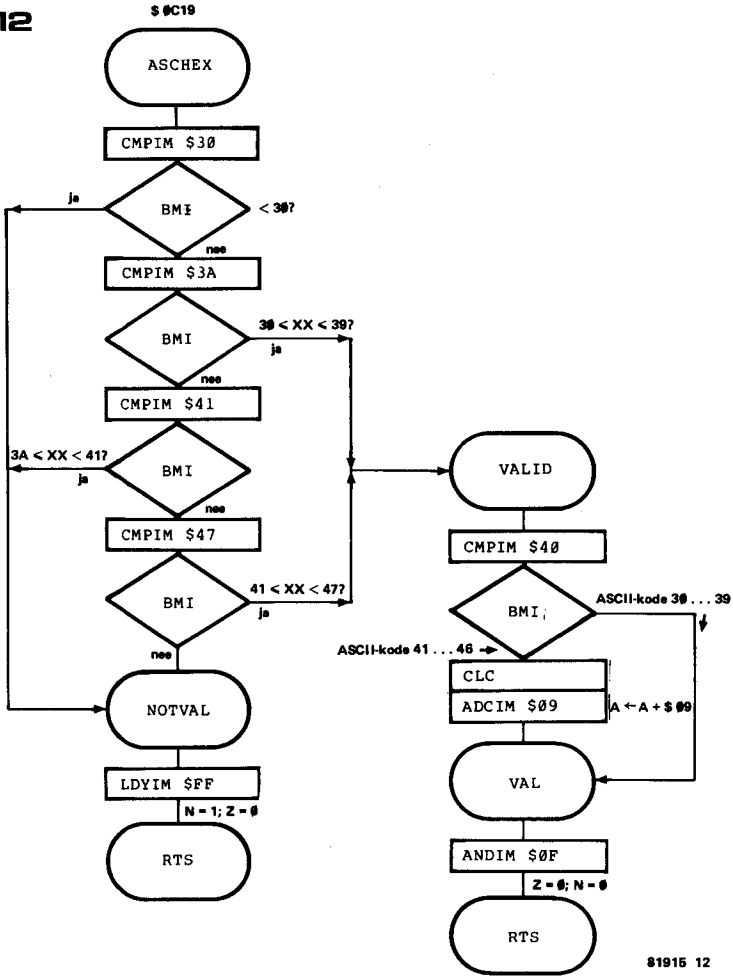
★ 11



Figuur 11. De subroutine RDBYT zorgt voor het lezen van de band van een databyte. Dat gebeurt door twee keer achter elkaar een datanibble van de band te lezen.

ligt maar gaan doen en **de subroutine RDBYT** van figuur 11 bespreken.
Van de software van DUMP weten we dat van een databyte eerst het linker nibble wordt verzonden en dan het rechter nibble; zie punt 6. Bij het teruglezen komen we dus eerst het linker nibble en dan het rechter nibble tegen.
De subroutine RDBYT begint met een andere subroutine. Namelijk

★ 12



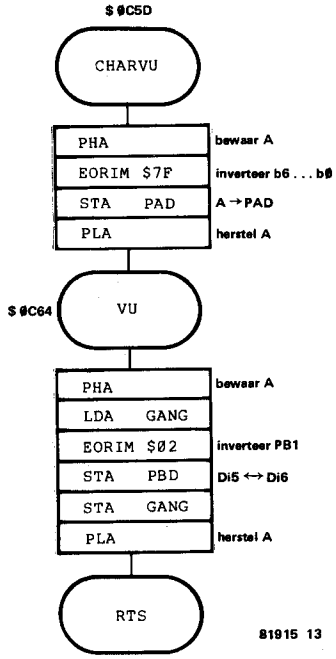
Figuur 12. De "subroutine" ASCHEX (of nòg juister: "sub-sub-subroutine") wordt twee keer aangeroepen vanuit RDBYT (zie figuur 11). Hij zorgt ervoor dat de ASCII-kode van een datanibble wordt omgezet in het datanibble.

RDCH. Zie punt 13 en figuur 10. Er wordt dus een ASCII-karakter van de band gelezen en in de accu gezet. Er zijn nu twee mogelijkheden voor dat ASCII-karakter: òf het is de kode van het data-eindkarakter "/", òf het is de kode van een datanibble. In het ene geval verlaten we RDBYT meteen met een Z-vlag één, in het andere geval gaan we door naar het label RBB. En daar wordt voor de eerste keer een beroep gedaan op **de subroutine ASCHEX** van figuur 12, die we nu eerst zullen bespreken.

Indien het om de ASCII-kode van een datanibble gaat moet de ASCII-kode worden omgezet in de bijbehorende waarde van dat nibble. Dat gebeurt overigens pas nadat de software van ASCHEX zich ervan verzekerd heeft dat het inderdaad gaat om de ASCII-kode van een datanibble.
Voor datanibbles Ø . . . 9 geldt een ASCII-kode 3Ø . . . 39 en voor datanibbles A . . . F een ASCII-kode 41 . . . 46. Mocht er sprake zijn van een andere ASCII-kode dan de al genoemde kodes, dan moet dat kenbaar worden gemaakt in de stand van de een of andere vlag en moet eerst de subroutine ASCHEX en later de subroutine RDBYT worden verlaten.
De subroutine ASCHEX begint met een vraag-en-antwoord-spelletje, dat bestaat uit vier keer een instruktie CMP, gevolgd door een BMI. ASCII-kodes die niet bij een datanibble horen leiden tot het label NOTVAL: ASCHEX wordt verlaten met een N-vlag één. ASCII-kodes die wèl bij een datanibble horen leiden tot het label VALID. Nu kan de ASCII-kode worden "omgebouwd" tot de waarde van het datanibble. In de gevallen Ø . . . 9 kan men daartoe volstaan met het nul maken van het linker nibble (de instruktie AND #ØF), in de gevallen A . . . F moet er eerst nog $Ø9 bij de accu-inhoud worden opgeteld. De subroutine ASCHEX wordt verlaten met een N-vlag één.
Terug naar RDBYT en terug naar figuur 11. De BMI na de eerste ASCHEX (label RBB) geeft direkt al aanleiding tot een RTS indien de N-vlag één is, dus indien er geen ASCII-kode van een datanibble werd opgevist. Is dat wèl het geval, dan volgen er vier instrukties ASLA. Het behandelde datanibble wordt dus het linker nibble van de accu èn van de geheugenplaats BYTE. Daarna wordt er wéér een ASCII-karakter ingelezen: JSR-RDCH. Als blijkt dat het gaat om een data-eindkarakter verlaten we RDBYT (label RBA) met een Z-vlag één. Zoniet, dan doorlopen we de ASCHEX-molen opnieuw. En mits de aansluitende BMI dat goed vindt gaan we, met behulp van de instruktie ORA-BYTE, het nieuwe datanibble dienst laten doen als rechter nibble van de accu-inhoud. Vlak voor de afsluitende RTS krijgt Y een waarde $Ø1 toegekend. Daardoor zijn zowel de N-vlag als de Z-vlag nul.

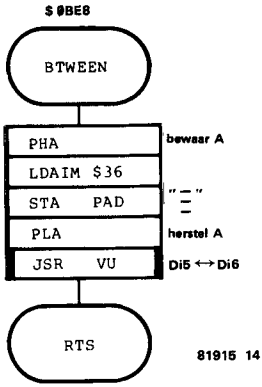
15 Nu nog een paar subroutines van RDTAPE die te maken hebben met het laten zien, op twee van de zes zevensegmentdisplays, van de stand van zaken rond het al dan niet inlezen van een datazending. Subroutines dus, die zorgen voor de drie plaatjes van figuur 7 (hoofdstuk 11) op pagina 96 van boek 3. Allereerst **de subroutine CHARVU/VU** van figuur 13. In de meeste gevallen wordt alles vanaf het label CHARVU tot en met de RTS doorlopen, in één ander geval (subroutine BTWEEN; zie figuur 14) wordt een beroep gedaan op de instrukties vanaf het label VU. Dat laatste is óók het geval tijdens het inschrijven in het geheugen van het gezochte, en gevonden datablok.
De subroutine CHARVU begint met bewaren, op de stapel, van de accu-inhoud. Vervolgens worden de bits bØ . . . b6 van de accu-inhoud geïnverteerd, met behulp van de instruktie EOR #7F. De nieuwe accu-inhoud is het nieuwe segmentpatroon (STA-PAD). En nu we het tòch over segmentpatronen hebben: in figuur 15 staan alle 128 mogelijke segmentpatronen bij elkaar. Patronen van dezelfde rij hebben een onderling identiek linker nibble H. Patronen van dezelfde kolom hebben een onderling identiek

★ 13

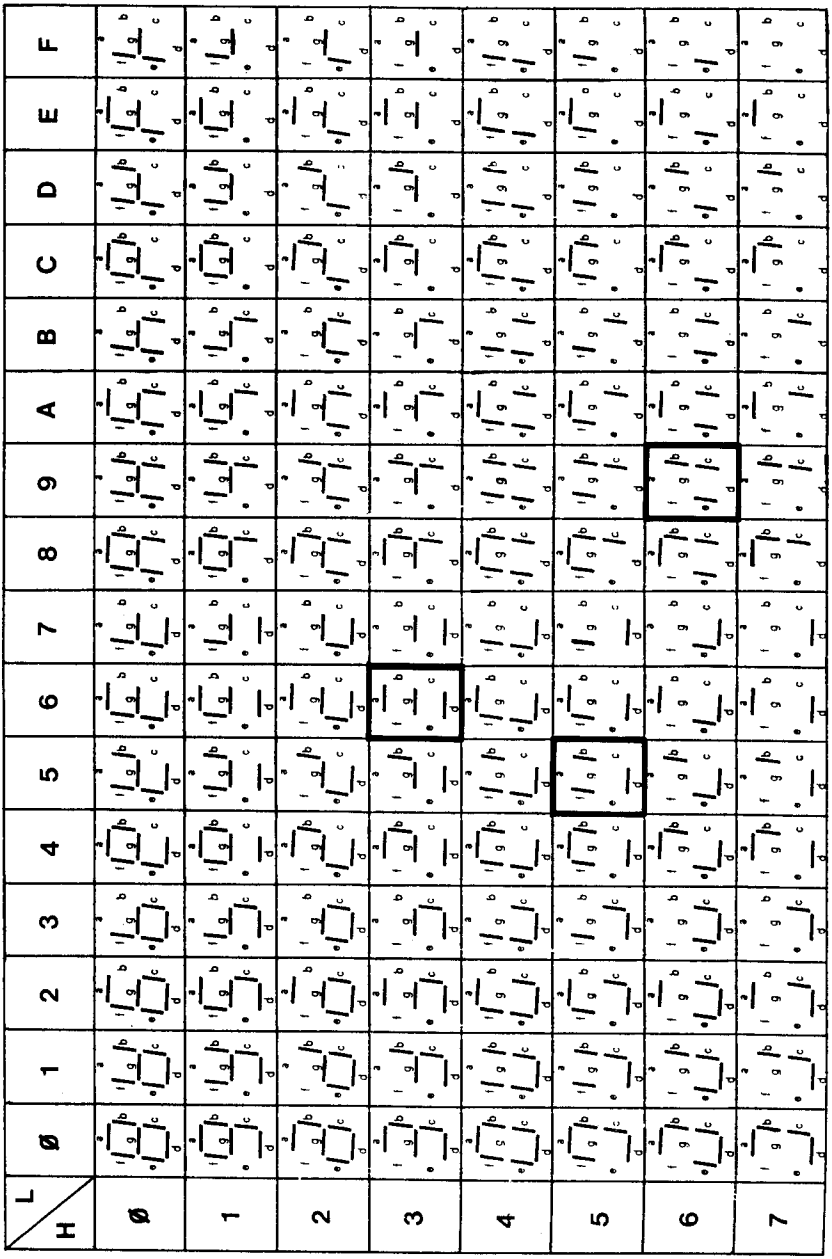


Figuur 13. Van de subroutine CHARVU/VU zorgen de instrukties vanaf het label VU ervoor dat de displays Di5 en Di6 elkaar aflossen. Omdat CHARVU en VU op regelmatige tijdstippen worden aangeroepen is er sprake van "display-multiplexing": de beide displays lichten schijnbaar gelijktijdig op. Het gedeelte vanaf CHARVU zorgt ervoor dat de, op bit b7 na, geïnverteerde accu-inhoud dienst doet als segmentpatroon op poort A.

★ 14



Figuur 14. De subroutine BTWEEN zorgt voor de weergave van het "zoekplaatje": de drie horizontale segmenten lichten op. Zolang er geen syncs zijn vastgesteld licht Di6 op, af en toe afgewisseld door Di5. Zodra het lezen van de 255 syncs van de band begint wisselen Di5 en Di6 elkaar regelmatig af.



Figuur 15. Een overzichtstabel van alle 128 mogelijke segmentpatronen; H is het linker nibble van PAD, L het rechter nibble.

rechter nibble L. Het preciese waarom? van die geïnverteerde accu-inhoud komt aan de orde bij de verdere bespreking van de subroutine RDTAPE (zie punt 17). Na de STA-PAD wordt de oorspronkelijke accu-inhoud weer in ere hersteld (PLA). Waarom wordt de inhoud van de accu eigenlijk bewaard? Omdat de accu een ASCII-karakter bevat dat nog verder moet worden verwerkt, maar vóórdat die verdere verwerking plaatsvindt wordt dat ASCII-karakter in geïnverteerde vorm aan het display "gemeld". Zo zit dat.
Om dezelfde reden is de eerste instruktie van de subroutine VU een PHA. En dáárop volgen vier instrukties die het volgende doen:

A = PBD: Ø Ø 1 1 Ø Ø 1 Ø
EOR #Ø2: Ø Ø Ø Ø Ø Ø 1 Ø
resultaat: Ø Ø 1 1 Ø Ø Ø Ø (A = PBD = GANG)

(nul als bits gelijk, één als bits ongelijk)
We zijn uitgegaan van een beginwaarde $32 voor PBD, zoals die aan het begin van RDTAPE is vastgelegd. Die beginwaarde had tot gevolg dat het display Di6 kon oplichten. Zie punt 10. Het resultaat van het "EORren" is dat het bit PB1 is geïnverteerd. Dat betekent dat de displayschakelaar (PB1 . . . PB4) in een zodanige stand is gezet dat nu het display Di5 kan oplichten. Met andere woorden: er wordt overgegaan van Di6 op Di5. Was daarentegen vóór de aanroep van VU het display Di5 aan bod, dan is na het "EORren" display Di6 weer aan bod. Met andere woorden: Na elke doorloop van VU gaat het ene display uit en het andere display aan; de displays lossen elkaar af. Indien er gedurende een bepaalde tijd maar vaak genoeg achter elkaar een JSR-CHARVU of een JSR-VU plaatsvindt lijkt het net alsof beide displays Di5 en Di6 tegelijk oplichten! Dat is niets meer en niets minder dan het principe van de "display-multiplexing", dat we in hoofdstuk 7 (boek 2 dus) hebben leren kennen.

16 **De subroutine BTWEEN** van figuur 14 wordt op die momenten aangeroepen dat het "zoekplaatje", het tussenkarakter (drie horizontale oplichtende segmenten, zie figuur 7 (hoofdstuk 11) op pagina 96 van boek 3), plaatje ①, wordt weergegeven. Afgezien van instrukties voor het bewaren van de accu-inhoud gaat het om LDA # 36 (het bijbehorende segmentpatroon, zie figuur 15), STA-PAD en JSR-VU (het ene display uit, het andere display aan). De situatie van PAD blijft net zo lang bestaan totdat, als gevolg van een JSR-CHARVU, een ander segmentpatroon op PAD wordt gezet.

17 (vervolg van punt 10) We gaan verder met de bespreking van de subroutine RDTAPE. We waren toe aan het label SYNC van figuur 6a. Dáár gaan we nu verder. Het label SYNC begint met het toekennen van de inhoud $FF aan de geheugenplaats CHAR. Die CHAR speelde een rol bij RDCH, zie punt 13. De beginwaarde van CHAR lag niet vast aan het begin van RDCH en dat hoeft ook niet per se. Waarom dat hier wèl gebeurt zal u op een later tijdstip, twee alinea's verder worden geopenbaard.
Vanaf het label SYNCA is RDTAPE bezig met het inlezen van het eerste de beste bit van de band. Dat inlezen gaat via de subroutine RDBIT (punten 11 en 12). Zodra er een bit is vastgesteld gaat dit bit dienst doen als bit b7 van CHAR (ROR-CHAR) en van A (LDA-CHAR). Vervolgens de

instruktie JSR-BTWEEN: het tussenkarakter (drie horizontale segmenten) verschijnt in beeld. Zie punt 16. En daarna krijgen we de instruktie CMP # 16. Men vraagt zich dus af of er al een synchronisatie-karakter is ingelezen. Want $16 is de ASCII-kode van een synchronisatie-karakter.
Men hoeft er niet verbaasd over te zijn dat er na de eerste RDBIT, dus na de eerste passage van het label SYNCA, nog geen sprake kan zijn van een synchronisatie-karakter. Immers: na de eerste passage van SYNCA is de inhoud van CHAR
Ø1111111 of alwéér:
11111111.
Vergelijk dat met de ASCII-kode $16:
ØØØ1Ø11Ø.
De beginwaarde ($FF) van CHAR is zodanig dat er niet toevallig al een sync wordt vastgesteld terwijl er nog geen 8 bits achter elkaar zijn ingelezen. Dat komt omdat het bit bØ van $16 nul is en het bit bØ van de beginwaarde van FF één.
Zodra er eindelijk dan een sync is ingelezen krijgt de geheugenplaats SY een inhoud $ØA toegekend en is de software van RDTAPE bij het label TENSYN van figuur 6a terechtgekomen. Het is de bedoeling dat er tien syncs achter elkaar, en niet onderbroken door een ander karakter, van de band worden gelezen. Zodra dit het geval is betekent dat voor RDTAPE het ondubbelzinnige begin van een datazending. Indien de BNE na de instruktie CMP # 16 op springen staat volgt de sprong terug naar het label SYNC, voor de herhaling van de sync-procedure. Dit is de herkansing, waarvan sprake is indien er tussentijds een ander karakter dan een sync wordt vastgesteld.
Binnen de programmalus rond het label TENSYN wordt de sprong naar RDCH gevolgd door de sprong naar CHARVU: er gaat iets gebeuren met het display. Tenzij er tussentijds een van een sync afwijkend karakter is ingelezen, tijdens de voorafgaande RDCH, staat er $16 in de accu. Indien men de bits bØ . . . b6 van de accu-inhoud inverteert (zie de bespreking van CHARVU, in punt 15) wordt de inhoud van PAD tijdens CHARVU $69. Figuur 15 leert ons dat dan het sync-plaatje ② van figuur 7 (hoofdstuk 11, boek 3, pagina 96) is te zien. En wel op het display Di5 èn het display Di6. Dat komt omdat elke doorloop van CHARVU gepaard gaat met het verwisselen van funkties: de ene gaat aan, de andere gaat uit. En verder zij vastgesteld dat gedurende het de revue passeren van de syncs zeer regelmatig de subroutine CHARVU wordt aangeroepen; hetzij via de programmalus rond TENSYN, hetzij via de aansluitende programmalus rond het label STAR (zie boyenaan figuur 6b). Omdat Di5 en Di6 elkaar dus snel afwisselen in hun "oplichtend" werk is er sprake van de gewenste "display-multiplexing": elk display moet op zijn beurt wachten, maar schijnbaar zijn ze beide tegelijk aan de beurt.

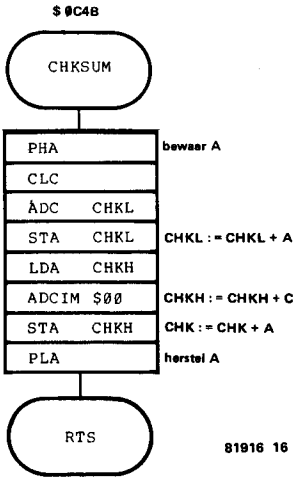
18 Nadat er tien ononderbroken syncs zijn vastgesteld, dus totaal elf syncs (eentje vóórdat het label TENSYN is bereikt), zijn we toe aan het label STAR, bovenaan in figuur 6b. De programmalus rond dit label wordt pas onderbroken als alle resterende syncs (maximaal 255 min 11 = 244 stuks) de revue zijn gepasseerd. Er zijn twee mogelijkheden voor het onderbreken van de lus. De eerste mogelijkheid is dat het data-beginkarak-

ter "*" wordt ingelezen. Het programma gaat dan verder vanaf het label STARA. De tweede mogelijkheid is dat er een karakter is ingelezen dat geen sync voorstelt en ook geen "*". Er is dan kennelijk iets fout gegaan en we beginnen dan helemaal opnieuw: terug naar het label RDTAPE.
Het label STARA van figuur 6b wordt dus bereikt zodra het data-begin-karakter "*" is ingelezen. De ASCII-kode van dit karakter is $2A. Indien men de bits bØ . . . b6 inverteert en de aldus gewijzigde accu-inhoud toekent aan PAD (zie CHARVU, punt 15), krijgen we te maken met een segmentpatroon $55. Figuur 15 leert ons dat dan, tijdens de CHARVU direkt na het label STARA, plaatsje ③ van figuur 7 (hoodstuk 11, boek 3, pagina 96) is te zien. Het "hebbes-plaatje" dus.
Of dat plaatje echter gedurende langere tijd is te zien of niet hangt af van de nu volgende fase van RDTAPE, die zich bezighoudt met het inlezen van het programmanummer ID. Na afloop van de subroutine RDBYT is het bij de datazending horende programmanummer ingelezen. Dit nummer staat in de accu en wordt vergeleken met de inhoud van de geheugenplaats ID. Met andere woorden: er wordt gekeken of het gevonden ID overeenkomt met de, vooraf door de gebruiker opgegeven, gewenste ID. Is dat het geval, dan gaat het programma direkt verder met de instrukties vanaf het label RDSA. Is dat echter niet het geval, dan wordt er eerst gekeken of er soms sprake is van het speciale, door de gebruiker opgegeven ID ØØ of FF. Dat nakijken gebeurt via de instrukties vanaf het label CHKID.
Indien blijkt dat er geen ID ØØ en ook geen ID FF is opgegeven volgt de sprong terug, via het tussenlabel SYNVEC, naar het begin van RDTAPE. Kennelijk is RDTAPE bezig met een datazending die we helemaal niet willen hebben en moeten we wachten totdat de gewenste datazending door RDTAPE in behandeling wordt genomen. En dat heeft RDTAPE niet zelf in de hand, want dat hangt af van wat de cassette-band te bieden heeft. Het is overigens deze situatie die aanleiding geeft tot een zeer kortstondige melding (áls men het al in de gaten heeft) van het "hebbes-plaatje" ③ (CHARVU, direkt na het label STARA). Dit plaatje is te zien gedurende een tijd, die begint met de JSR-CHARVU (label STARA) en eindigt met de JSR-BTWEEN, na de terugkeer naar het begin van RDTAPE (zie figuur 6a).

Nu eerst een intermezzo. **De subroutine CHKSUM** van figuur 16 zorgt ervoor dat bij het 16-bits getal CHK=(CHKH,CHKL) een bedrag wordt opgeteld dat gelijk is aan de inhoud van de accu (een acht bits getal) vlak vóór JSR-CHEKSUM. Díe accu-inhoud blijft behouden (PHA respektievelijk PLA in figuur 16).
Nadat we het geval van een niet gevraagde ID behandeld hebben blijven er drie mogelijkheden over:

1. **Gevonden ID Ø1 . . . FE.** De instrukties vanaf het label RDSA tot aan het label FILMEM worden direkt uitgevoerd. Dat wil zeggen dat het rechter startadresbyte SAL wordt ingelezen (RDBYT) en wordt verwerkt in het getal CHK. Vervolgens van hetzelfde laken een pak voor het linker startadresbyte SAH. De adreswijzer POINT wordt gericht op het startadres SA van het bij de datazending horende datablok.
2. **Opgegeven ID ØØ.** Er wordt alsnog naar het label RDSA gesprongen. Zie 1. Dit klopt, want bij een ID ØØ wordt weliswaar het van de band gelezen ID genegeerd, maar niet het van de band gelezen startadres SA.

★ 16



Figuur 16. De subroutine CHKSUM zorgt voor het optellen bij het 16 bits getal, gevormd door de inhoud van de geheugenplaatsen CHKH en CHKL, van een 8 bits getal, dat een zojuist ingelezen databyte voorstelt.

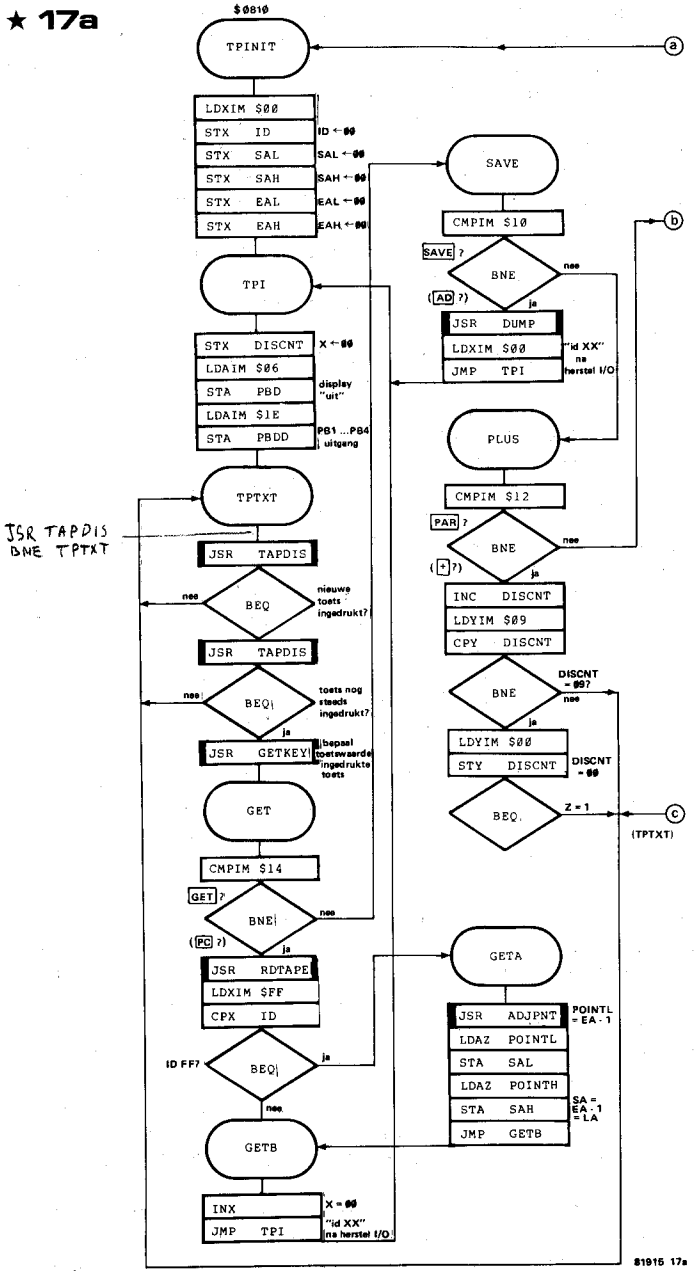
3. **Opgegeven ID FF.** Het van de band gelezen startadres (achtereenvolgens SAL en SAH) wordt in het getal CHK verwerkt (twee keer de subroutine CHECKSUM). Vervolgens wordt de adreswijzer POINT gericht op een **door de gebruiker opgegeven startadres (inhoud geheugenplaatsen SAL en SAH).** Dit klopt, want bij een ID FF worden zowel het ID van de band als de SA van de band genegeerd. Dat desondanks SAL en SAH van de band worden gelezen en verwerkt in het getal CHK, is noodzakelijk omdat anders vrijwel zeker het eindbedrag van CHK niet klopt, en dat zou aanleiding geven tot foutieve konklusies. Na het specificeren van POINT springen we naar het label FILMEM.

19 De instrukties vanaf het label FILMEM van figuur 6b houden zich bezig met het inlezen van de band van het bij de datazending horende datablok, en met het inschrijven van dit datablok in het geheugen van de junior-computer. Het begint met de aanroep van de subroutine RDBYT. Zie punt 14 en de figuren 11 en 12. Blijkt het na afloop van RDBYT om ongeldige data te gaan (N=1), dan gaan we "terug naar AF": RDTAPE. Blijkt het na afloop van RDBYT om een data-eindkarakter "/" te gaan, dan gaat RDTAPE verder met het label CHECK. In alle andere gevallen kan het uitsluitend gaan om geldige data, die wordt verwerkt in het getal CHK (JSR-CHECKSUM). Nadat dit is gebeurd wordt de zojuist ingelezen data ingeschreven in een geheugenplaats, die wordt aangewezen door de adreswijzer POINT. De beginwaarde van POINT is zodanig dat de eerste geheugenplaats die wordt beschreven een adres SA heeft.
Na het inschrijven wordt POINT met één verhoogd en zijn we toe aan het label FMA van figuur 6b. Er wordt een beroep gedaan op de subroutine VU van punt 15 en figuur 13. Dit heeft tot gevolg dat het display Di5/Di6

wordt afgelost door het display Di6/Di5. Op poort PAD staat overigens nog steeds het segmentpatroon, zoals dat is vastgesteld tijdens de JSR-CHARVU direkt na het label STARA. Dus het "hebbes-plaatje" ③ is nog steeds zichtbaar, en wel op zowel Di5 als Di6, omdat telkens na de behandeling van één databyte (programmalus rond het label FILMEM) het aflossen van de displays plaatsvindt: het al eerder besproken principe van de display-multiplexing ("display-ploegendienst"). De afsluitende instruktie JMP-FILMEM zorgt ervoor dat opnieuw een databyte in behandeling kan worden genomen.
En dan de laatste instrukties van RDTAPE. Dus de instrukties vanaf het label CHECK in figuur 6b. Zodra er een data-eindkarakter "/" is vastgesteld wordt de rechter helft CHKL van het op de band "genoteerde" getal CHK (eindwaarde) ingelezen. In de accu dus. Dit wordt vergeleken met de eindwaarde CHKL, tot stand gekomen tijdens het lezen van de band: de inhoud van de geheugenplaats CHKL. Indien de beide eindwaarden niet overeenstemmen weten we meteen hoe laat het is: via het tussenlabel SYNVEC helemaal terug naar het begin: RDTAPE. Stemmen de beide eindwaarden echter wèl overeen, dan is het zinvol om na te gaan of datzelfde het geval is voor de beide eindwaarden CHKH, dus de linker helften van de beide CHK's. Zijn ook deze eindwaarden aan elkaar gelijk, dan is RDTAPE afgerond: RTS. Is dat echter niet het geval, dan gaan we alsnog via SYNVEC terug naar het begin.
We zien dat in het geval van niet overeenstemmende kontrole-bytes CHKL of CHKH het bij de datazending horende datablok wèl in het geheugen van de junior-computer wordt gezet, maar dat de subroutine RDTAPE niet wordt verlaten. Dit is ook ècht te zien. Want het "hebbes-plaatje" ③ wordt gevolgd door het "zoekplaatje" ① (zie figuur 7 van hoofdstuk 11). En normaal gesproken is het display gedoofd als RDTAPE wordt verlaten na een aanroep van RDTAPE vanuit PM. Dat het zover is zien we bij PM aan de terugmelding "READY", en verder aan het gedoofde display. In het geval dat RDTAPE wordt aangeroepen vanuit TM kunnen we evenéén zien dat we klaar zijn met het inlezen van een datazending. Er volgt dan de terugmelding "id XX" (zie verderop in dit hoofdstuk).

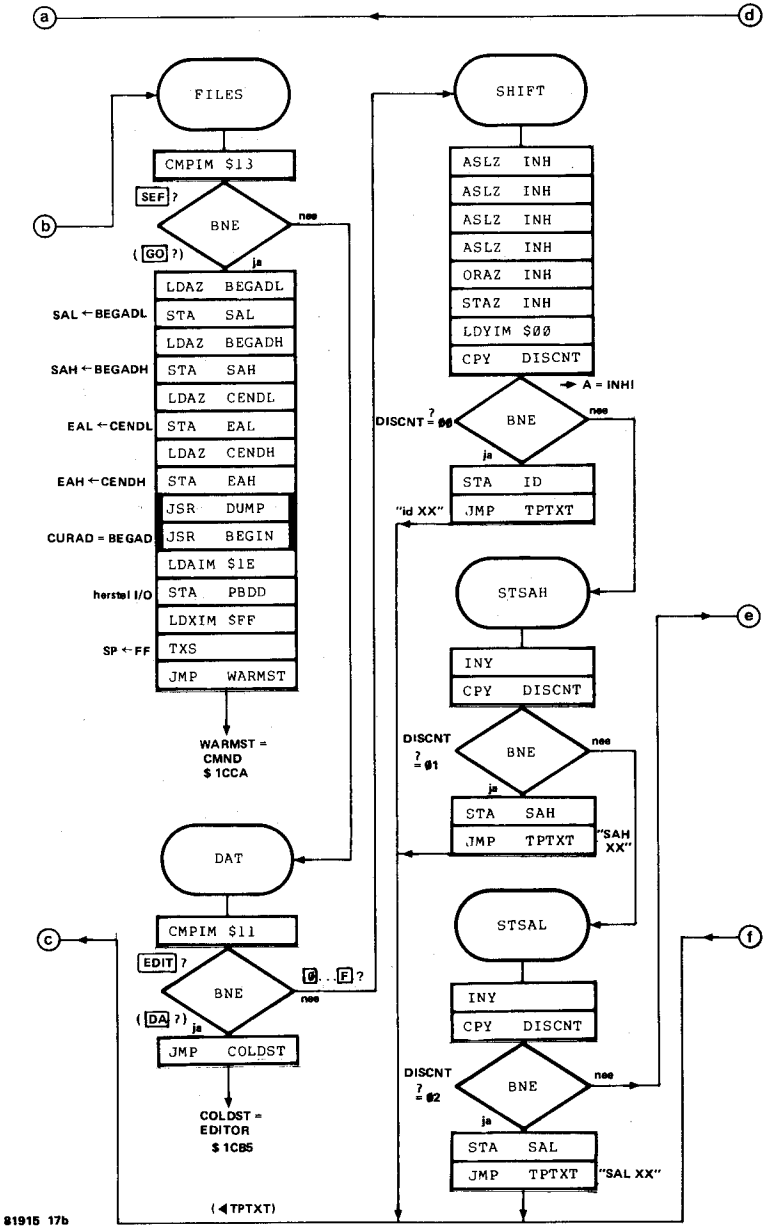
20 Het stroomdiagram van de **hoofdroutine van het systeemprogramma TM** vindt u in de figuren 17a, 17b en 17c. De struktuur van het hoofdprogramma wijkt niet af van die van de andere systeemprogramma's van de junior-computer. Er is dus sprake van een voorroutine, te weten de instrukties vanaf het label TPINIT tot aan het label TPI, gevolgd door de instrukties vanaf het label TPI tot aan het label TPTXT. Dat laatste label is tevens het centrale label van de hoofdroutine van TM. Naar dit label wordt teruggesprongen na afloop van de PAR-toetsroutine (zie punt 24) en nadat door de gebruiker opgegeven data is geladen in één van de negen geheugenplaatsen ID, SAH, SAL, EAH, EAL, BEGADH, BEGADL, ENDADH en ENDADL. Na afloop van de GET-toetsroutine (punt 22) en van de SAVE-toetsroutine (punt 23) komen we pas in TPTXT terecht nadat het tweede deel van de TM-voorroutine, vanaf het label TPI, is doorlopen. Na afloop van de SEF-toetsroutine (punt 25) en van de EDIT-toetsroutine (punt 25) is TM verlaten; we zitten dan in de editor, die is ondergebracht in de standaard-EPROM van de junior-computer.

★ 17a

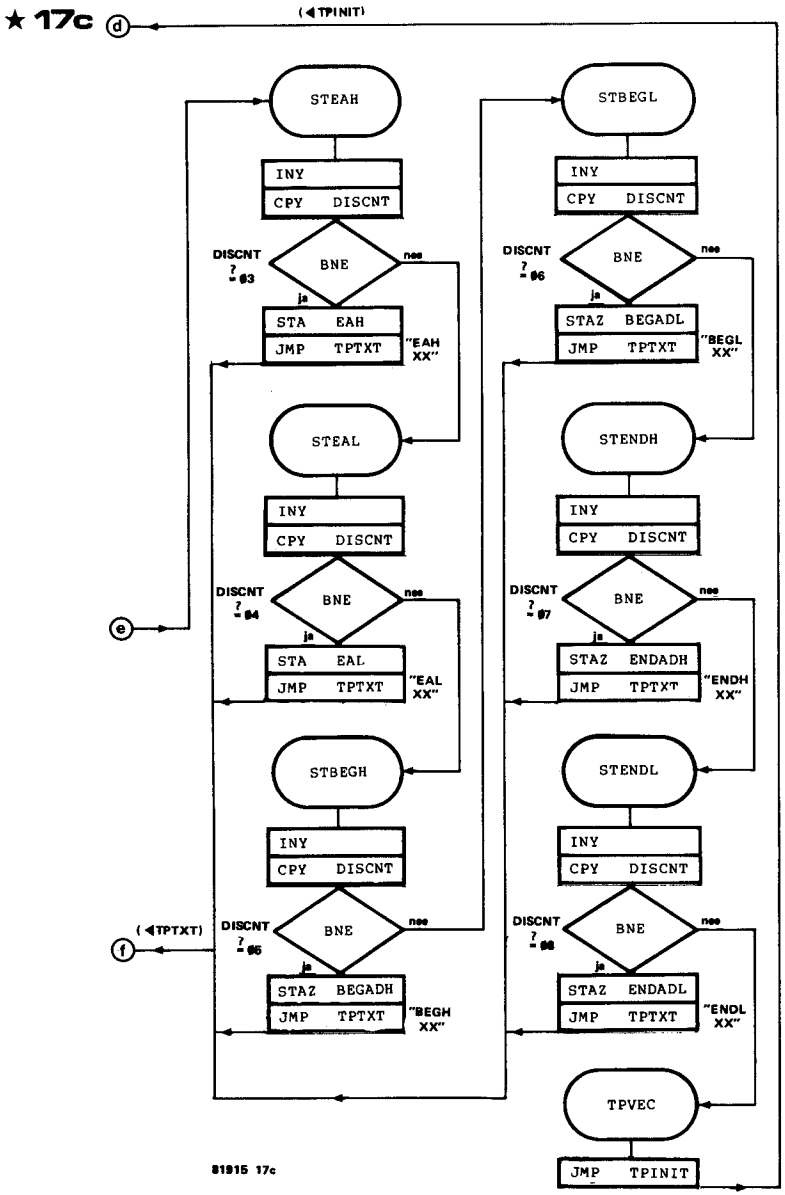


Figuur 17a. Het eerste gedeelte van de hoofdroutine van het systeemprogramma TM.

★ 17b



Figuur 17b. Het tweede gedeelte van de hoofdroutine van TM.



Figuur 17c. Het derde gedeelte van de hoofdroutine van TM.

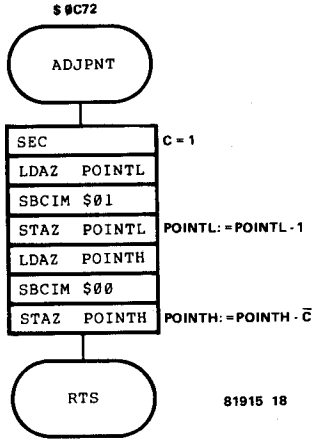
Direkt na het centrale label TPTXT wordt de relevante geheugenplaats met naam en inhoud op het display getoond; vervolgens is het een kwestie van wachten op een ingedrukte toets. Afhankelijk van welke toets is ingedrukt wordt er een toetsroutine of de datatoetsroutine afgewerkt. En dan kan het spelletje opnieuw worden gespeeld.
Een snelle blik op de software van TM leert dat op veel plaatsen de geheugenplaats DISCNT voorkomt. De inhoud van deze geheugenplaats doet dienst als **parameterteller**. De stand van de parameterteller bepaalt op welke van de negen geheugenplaatsen er data moet worden geladen, àls er data wordt opgegeven door de gebruiker. Verder bepaalt die parameterteller welke van de negen geheugenplaatsen op het display zichtbaar moet worden gemaakt, met naam en inhoud. Voor DISCNT gelden de volgende mogelijkheden:
DISCNT = Ø0 → weergave/opgave ID
DISCNT = Ø1 → weergave/opgave SAH
DISCNT = Ø2 → weergave/opgave SAL
DISCNT = Ø3 → weergave/opgave EAH
DISCNT = Ø4 → weergave/opgave EAL
DISCNT = Ø5 → weergave/opgave BEGADH
DISCNT = Ø6 → weergave/opgave BEGADL
DISCNT = Ø7 → weergave/opgave ENDADH
DISCNT = Ø8 → weergave/opgave ENDADL
Tot zover het globale overzicht van TM. Nu de details.

21 De **voorroutine van TM** bestaat uit de instrukties in figuur 17a vanaf het label TPINIT. De vier geheugenplaatsen, die direkt iets hebben te maken met het cassette-gebeuren, namelijk ID, SAH, SAL, EAH en EAL, krijgen een inhoud nul toegewezen. Na het label TPI krijgt de parameterteller DISCNT een inhoud nul. Immers: het label TPI kan vanuit drie richtingen worden bereikt. In alle drie gevallen is X gelijk aan nul. Met andere woorden: na elke doorloop door TPI wordt de geheugenplaats ID met naam en inhoud weergegeven op het display.
En dan gaan we poort B dresseren. Een inhoud $Ø6 voor PBD betekent dat de displayschakelaar (PB1 ... PB4 zijn tot uitgang verklaard) in de neutrale stand staat. Het display is "uit" en er worden ook geen ingedrukte toetsen ontdekt. Een korte pauze dus. Voor deze kwestie verwijzen we u naar hoofdstuk 7 (boek 2), evenals voor de diepere achtergrond van de instrukties van figuur 17a vanaf het centrale label TPTXT tot aan het label GET. De subroutine TAPDIS lijkt sterk op de bekende SCANDS-subroutine. Zie hiervoor punt 27 en figuur 19. Het eind van het liedje, bij het label GET, is in ieder geval dat een ingedrukte toets met de bijpassende toetswaarde (JSR-GETKEY) in de accu staat. Afhankelijk daarvan kan er een toetsroutine of de datatoetsroutine worden uitgevoerd.

22 **De GET-toetsroutine** bestaat uit de instrukties, linksonder in figuur 17a, vanaf het label GET. Zodra blijkt dat de GET-toets is ingedrukt volgt de sprong naar de subroutine RDTAPE. Zie hiervoor de punten 10 t/m 19 van dit hoofdstuk. Alles wat van de cassette-band binnenkomt wordt gelezen en zodra de door de gebruiker gewenste datazending aan bod is wordt het bijbehorende datablok in het geheugen van

de junior-computer gezet. Vooraf is door de gebruiker een ID opgegeven en, in het geval van een ID FF, tevens een SAH en een SAL.
Na de JSR-RDTAPE wordt er gekeken of de zojuist binnengehaalde zending is binnengehaald via de ID=FF-truuk. Indien dat niet het geval is (label GETB) krijgt X de waarde nul en gaan we via TPI terug naar TPTXT. We krijgen "id XX" te zien. De omweg via TPI is nodig om dat de I/O weer in overeenstemming moet worden gebracht met de TM-situatie. De voor de "super-subroutines" DUMP/DUMPT en RDTAPE van toepassing zijnde I/O wijkt namelijk af van de voor TM geldende I/O.
Indien er sprake was en is van een ID FF wordt het label GETB bereikt via een omweg: de instrukties vanaf het label GETA. Daar komen we allereerst **de subroutine ADJPNT** van figuur 18 tegen. Dat is een simpel geval: de adreswijzer POINT wordt met één opgehoogd. Wáár is dat goed voor?
De adreswijzer POINT staat tijdens RDTAPE gericht op de geheugenplaats die wordt geladen met het meest recente, van de band gelezen databyte. Nadat dit is gebeurd wordt die adreswijzer met één verhoogd. Zie punt 19 en figuur 6b. Zodra het komplete datablok is ingelezen staat POINT gericht op een adres dat één hoger is dan het adres van het laatste byte van het datablok. Met andere woorden: POINT = EA = LA + 1. Het eindadres is niet direkt gespecificeerd op de band, maar volgt indirekt uit SA (hetzij van de band, hetzij, bij ID = FF, via de gebruiker) plus het aantal bytes van het datablok plus één.
Welnu: na JSR-ADJPNT staat POINT gericht op het laatste adres LA van het zojuist ingelezen datablok. Waarom is dat gedaan? Er is, nà de vier op JSR-ADJPNT volgende instrukties, een nieuwe SA gespecificeerd. Die nieuwe SA vindt toepassing in het geval dat een aantal datablokken, behorend tot (stukken) programma's in ongeassembleerde vorm, aan elkaar moet worden geplakt, waarbij uiteraard de tussenliggende EOF's (data 77)

★ 18



Figuur 18. De subroutine ADJPNT zorgt ervoor dat de adreswijzer POINT met één wordt verlaagd en wordt in sommige gevallen aangeroepen tijdens de uitvoering van de GET-toetsroutine van TM.

moeten worden verwijderd. In hoofdstuk 11 (boek 3) is uitgebreid ingegaan op deze ID=FF-truuk. Zie met name figuur 11 van dat hoofdstuk. Uit de software van de GET-toetsroutine kunnen we opmaken dat het niet nodig is om, vóór het inlezen van het volgende datablok, iets op te geven. Immers: niet alleen de SA is automatisch op de juiste waarde gezet, óók de inhoud van de geheugenplaats ID is nog steeds FF. We hoeven dus voor het inlezen van een nieuw datablok (let wel: een "bij te plakken" datablok van een (stuk) ongeassembleerd programma!) uitsluitend opnieuw de toets GET in te drukken **èn ervoor te zorgen dat de eerste de beste komplete, door de band geleverde datazending ook daadwerkelijk** het gewenste, **"bij te plakken" datablok bevat!!** Want bij een ID FF wordt de eerste de beste komplete datazending geaccepteerd door de junior-computer. Maar dat wist u al van boek 3.

23 **De SAVE-toetsroutine** bestaat uit de vijf instrukties vanaf het label — hoe toepasselijk! — SAVE in figuur 17a. Zodra blijkt dat de SAVE-toets is ingedrukt volgt de aanroep van de subroutine DUMP. Zie hiervoor de punten 1 t/m 9 van dit hoofdstuk. Een vooraf gespecificeerde datazending wordt op de cassette-band gezet. Die hopelijk draait, waarbij de cassette-recorder in de opnamestand staat. Vooraf moeten de geheugenplaatsen ID, SAH, SAL, EAH en EAL met data zijn geladen die in overeenstemming is met de specifikatie.
Na afloop van de subroutine DUMP krijgt X de waarde nul toegekend. Dat betekent dat er na afloop, als het centrale label TPTXT via de TPI-omweg is bereikt, "id XX" op het display is te zien, waarbij XX gelijk is aan de door de gebruiker opgegeven ID. Net als bij de GET-toetsroutine (punt 22) is de TPI-omweg nodig om terug te schakelen van de DUMP-I/O op de TM-I/O.

24 **De PAR-toetsroutine** bestaat uit de instrukties vanaf het label PLUS in figuur 17b. Indien de PAR-toets blijkt te zijn ingedrukt wordt de inhoud van de parameterteller DISCNT met één verhoogd. Geeft dit aanleiding tot een inhoud $Ø9, dan krijgt DISCNT alsnog de waarde nul toegekend. Het indrukken van de PAR-toets heeft dus tot gevolg dat de volgende van de negen TM-geheugenplaatsen aan de beurt is om met naam en inhoud zichtbaar te worden gemaakt op het display. Voor de volgorde: zie punt 20. Na "ENDLXX" en het indrukken van toets PAR volgt dus "id XX".

25 **De SEF-toetsroutine** bestaat uit de instrukties vanaf het label FILES, linksboven in figuur 17b. Na het indrukken van de toets SEF wordt SA gelijk gemaakt aan BEGAD en het eindadres EA gelijk gemaakt aan CEND, de variabele eindadreswijzer van het op de cassette-band te schrijven (JSR-DUMP), ongeassembleerde programma. Er wordt vanuit gegaan dat vóór het indrukken van toets SEF een ID is gespecificeerd.
Na afloop van het "DUMPen" wordt TM verlaten, richting editor (warme startingang). Maar voor het zóver is volgt nog de subroutine BEGIN: de displaywijzer CURAD, op vele plaatsen in dit boek ook wel "instruktiewijzer" genoemd, wordt gericht op het adres BEGAD, tevens het beginadres SA. Verder wordt PBDD in overeenstemming gebracht met de

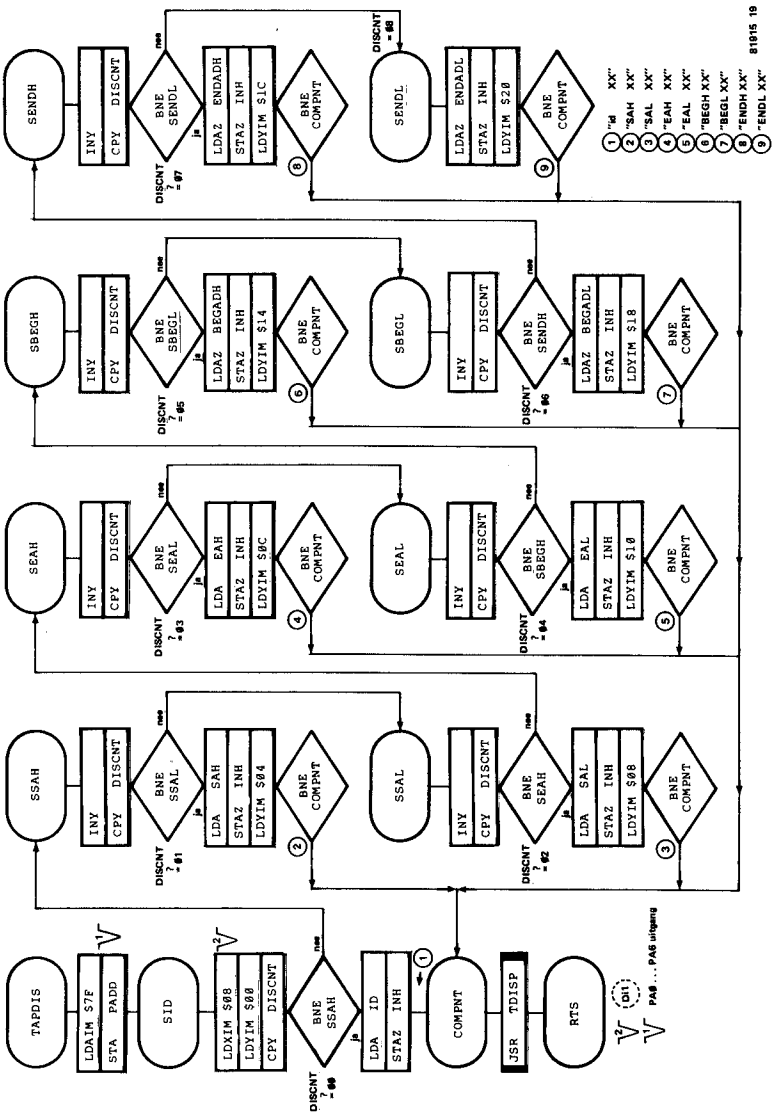
situatie zoals die is direkt na de RESET-voorroutine van de standaard-monitor. Zie hoofdstuk 7 (boek 2). Tevens wordt eerst nog de stapelwijzer (stack pointer) op zijn nummer ($FF) gezet. En pas dàn is het zover, die warme start.
De EDIT-toetsroutine (figuur 17b, label DAT) bestaat óók uit een sprong naar de editor, maar dan richting koude startingang. Meer valt er niet over te zeggen. Dat doen we dan ook maar niet.

26 Nu zijn alle niet-numerieke toetsen "uitgefilterd". Blijven over de numerieke toetsen Ø . . . F. Dus **de datatoetsroutine**. Dat is de hele rest van de TM-hoofdroutine. Dus alles vanaf het label SHIFT in figuur 17b. Dus de gehele rechter "kolom" van figuur 17b èn figuur 17c.
De gang van zaken: Een ingedrukte numerieke toets wordt verwerkt tot het nieuwe rechter nibble van de databuffer INH. Het oude rechter nibble gaat voortaan dienst doen als het nieuwe linker nibble. De manier waarop dit tot stand komt is al op verscheidene plaatsen, elders in de junior-computer-boekenserie beschreven, vandaar dat we dat nu niet doen.
De gewijzigde inhoud van de databuffer INH moet in één van de negen geheugenplaatsen worden gezet. Welke van de negen dat is, hangt af van de stand van de parameterteller. Dus hangt af van de inhoud van de geheugenplaats DISCNT. De data wordt geladen in geheugenplaats:

ID	(label SHIFT)	indien DISCNT=ØØ;
SAH	(label STSAH)	indien DISCNT=Ø1;
SAL	(label STSAL)	indien DISCNT=Ø2;
EAH	(label STEAH)	indien DISCNT=Ø3;
EAL	(label STEAL)	indien DISCNT=Ø4;
BEGADH	(label STBEGH)	indien DISCNT=Ø5;
BEGADL	(label STBEGL)	indien DISCNT=Ø6;
ENDADH	(label STENDH)	indien DISCNT=Ø7;
ENDADL	(label STENDL)	indien DISCNT=Ø8.

Nadat dit laden is gebeurd gaan we terug naar het centrale label TPTXT van TM, teneinde op het display de gevolgen te kunnen zien van het indrukken van één of twee numerieke toetsen.
Hiermee is de bespreking van de PM-hoofdroutine afgerond. Nu nog één subroutine en één "sub-subroutine".

27 **De subroutine TAPDIS** van figuur 19 zorgt voor de weergave van één van de negen geheugenplaatsen ID . . . ENDADL op het display. De displays Di1 . . . Di4 zorgen voor de weergave van de naam van de geheugenplaats, de displays Di5 en Di6 laten de inhoud van die geheugenplaats zien. We herhalen nu nog even in het kort een stukje hoofdstuk 7 en stellen vast dat, afhankelijk van X, het volgende display aan de beurt is om met nul, 1, 2, 3, 4, 5, 6 of alle segmenten op te lichten:
X=Ø8 → display Di1
X=ØA → display Di2
X=ØC → display Di3
X=ØE → display Di4
X=1Ø → display Di5
X=12 → display Di6
Direkt na het label TAPDIS van figuur 19 worden de poortlijnen

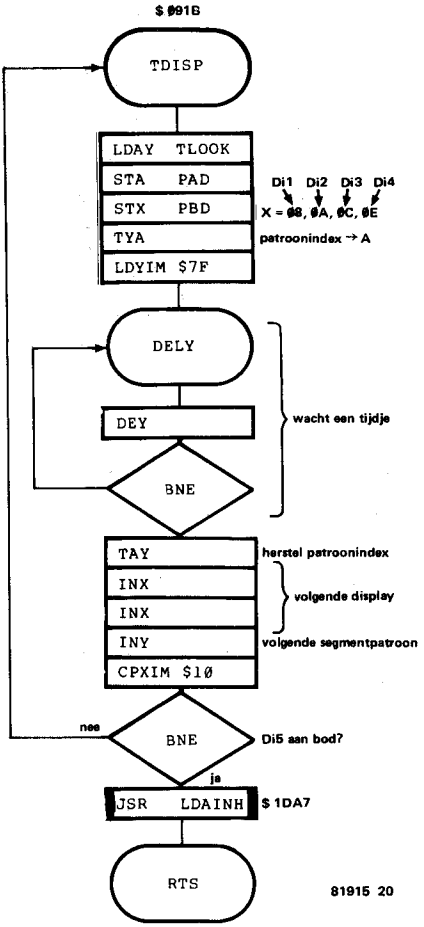


Figuur 19. De TM-subroutine TAPDIS stelt vast welke van de negen TM-geheugenplaatsen met naam en inhoud op het display moet worden weergegeven. Het eigenlijke werk wordt verricht in de subroutine TDISP van figuur 20.

PAØ . . . PA6 tot uitgang verklaard. Op deze poortlijnen komt immers het weer te geven segmentpatroon (zie figuur 15!) te staan. Direkt na het label SID van figuur 19 krijgt X de waarde $Ø8. Dus het display Di1 staat gereed om (tijdens de subroutine TDISP, die spoedig volgt) op te lichten. Maar voordat TDISP wordt aangeroepen moet eerst bekend zijn welke geheugenplaats zichtbaar moet worden gemaakt. En dat hangt af van de inhoud van de geheugenplaats DISCNT. Dus volgt er een soort "software-enquête", die het grootste deel van figuur 19 beslaat. Het Y-register speelt daarbij de rol van "enquêteur", en krijgt, nadat bekend is om welke van de negen geheugenplaatsen het gaat, meteen ander werk. Hij doet dan dienst als index voor een tijdens de subroutine TDISP gebruikte opzoektabel.
Zodra het label COMPNT van figuur 19 is bereikt, dus zodra de subroutine TDISP wordt aangeroepen, is er sprake van de volgende toestand:

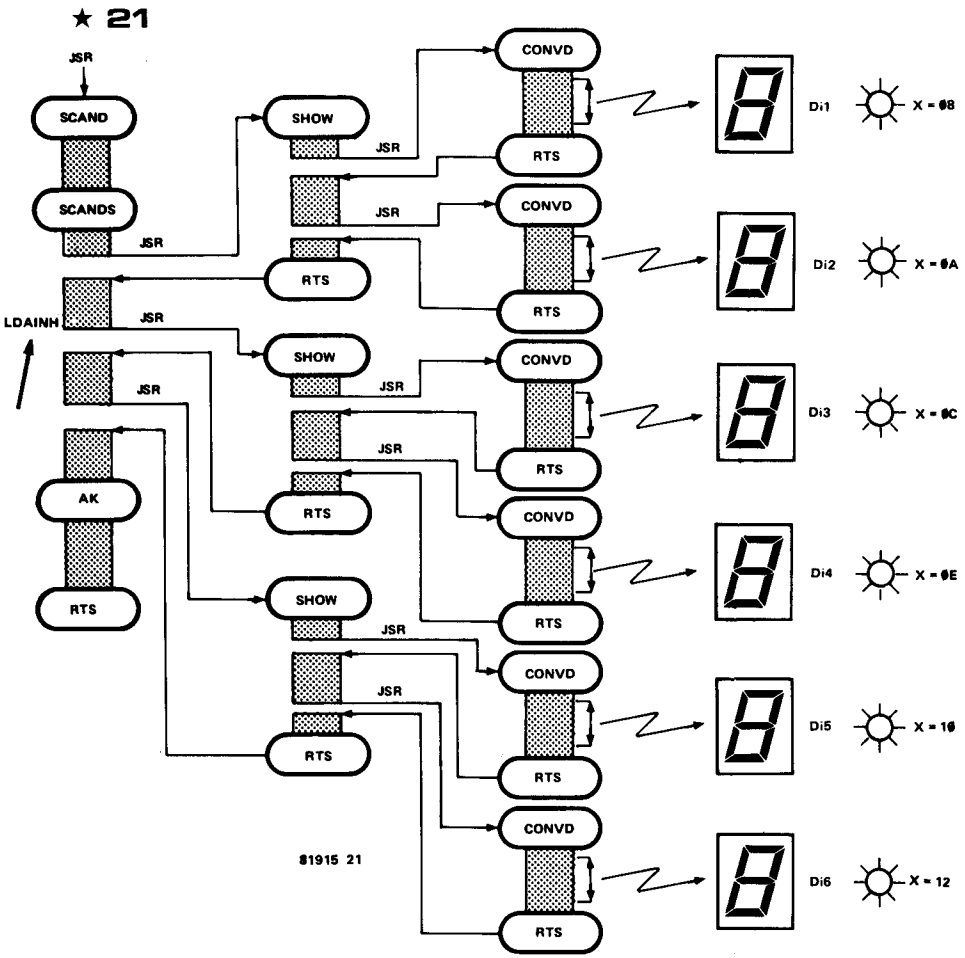
DISCNT=ØØ; INH ← inhoud ID; index Y=ØØ;
DISCNT=Ø1; INH ← inhoud SAH; index Y=Ø4;
DISCNT=Ø2; INH ← inhoud SAL; index Y=Ø8;
DISCNT=Ø3; INH ← inhoud EAH; index Y=ØC;
DISCNT=Ø4; INH ← inhoud EAL; index Y=1Ø;
DISCNT=Ø5; INH ← inhoud BEGADH; index Y=14;
DISCNT=Ø6; INH ← inhoud BEGADL; index Y=18;
DISCNT=Ø7; INH ← inhoud ENDADH; index Y=1C;
DISCNT=Ø8; INH ← inhoud ENDADL; index Y=2Ø.

28 **De subroutine TDISP** van figuur 20 zorgt voor de weergave op het display van de naam (Di1 . . . Di4) en de inhoud (Di5 & Di6) van de door DISCNT vastgelegde geheugenplaats. Verder stelt hij vast of er een nieuwe toets is ingedrukt of niet.
Eén doorloop van de programmalus rond het label TDISP is te vergelijken met de subroutine CONVD van de standaard-monitor. Zie figuur 14 van hoofdstuk 7. Boek 2 dus. Eerst wordt de accu geladen met data, afkomstig uit de opzoektabel TLOOK. Welke data, hangt af van de patroonindex Y vlak voor de aanroep van TDISP. Zie hiervoor punt 27. Deze data doet dienst als segmentpatroon en wordt dus in PBD gezet. Welk segmentpatroon wàt zichtbaar maakt volgt uit een onderzoek van de opzoektabel TLOOK (zie het betreffende deel van de PM/TM-listing, dus Aanhangsel 3, achter in dit boek) en uit figuur 15, het overzicht van alle 128 mogelijke verschillende segmentpatronen.
Na het bewaren van de patroonindex Y in de accu (TYA) wordt Y gebruikt voor een wachtlus. Gedurende de wachttijd licht Di1, Di2, Di3 of Di4 op. Na afloop gaat de patroonindex weer naar Y (TAY), die met één wordt verhoogd (INY), voor het ophalen van het volgende weer te geven segmentpatroon. Inmiddels is ook X voorbereid voor het aktiveren van het volgende display, één nummertje hoger. Tenminste: zolang Di5 niet aan bod is (CPX # 1Ø). Want dan besluiten we de doorloop van TDISP met de gang naar weer een andere subroutine, te weten LDAINH.
Wat is dàt nou weer voor een subroutine, die LDAINH? Nóóit van gehoord! Klopt. Neem boek 2 voor u. Zoek pagina 184 op. Bekijk het adres $1DA7 van de listing. We blijken midden in de subroutine SCAND/SCANDS te zitten. Gaat u nu een lichtje op? Bekijk figuur 21 (dat is tevens figuur 11 van hoofdstuk 7) en u ziet dat JSR-LDAINH inhoudt dat we bij



Figuur 20. De subroutine TDISP zorgt voor het om de beurt een tijdje oplichten van de zes displays. In aansluiting daarop wordt er gekeken of er een nieuwe toets is ingedrukt.

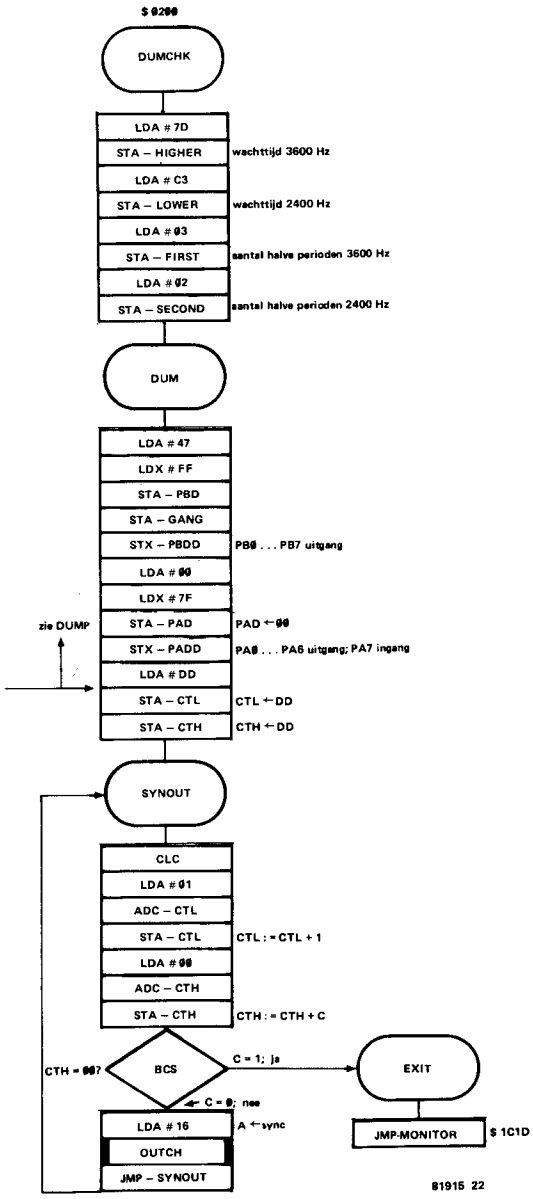
SCAND/SCANDS instappen op het moment dat begonnen wordt aan de weergave op het display van de databuffer INH. Dus de weergave op de displays Di5 en Di6. En in díe fase waren we nou juist aangeland tijdens TDISP: de weergave van de inhoud van één van de negen geheugenplaatsen ID . . . ENDADL op de displays Di5 en Di6.
Nadat LDAINH, alias SCAND/SCANDS is afgewerkt is aan de inhoud van de accu te zien of er een nieuwe toets is ingedrukt of niet: het laatste gedeelte (zie figuur 21) vanaf het label AK.
Hiermee is de bespreking van het systeemprogramma, en daarmee bijna dit hoofdstuk, afgerond.



Figuur 21. De tijdens TDISP (figuur 20) aangeroepen subroutine LDAINH bestaat uit het laatste gedeelte van de standaard-monitor-subroutine SCAND/SCANDS, vanaf het oplichten van display Di5.

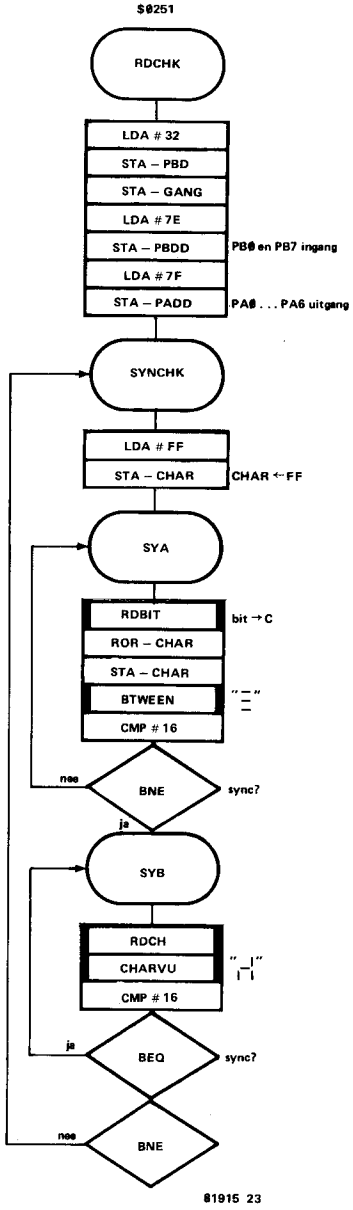
29 In hoofdstuk 11 (boek 3) zijn twee metoden besproken voor de afregeling van de PLL. In beide gevallen is een testprogrammaatje nodig. Bij de PLL-afregelmetode via het display is zowel een schrijf- als een leesprogrammaatje nodig, bij de PLL-afregelmetode via de oscilloskoop hoeft er vooraf uitsluitend wat op de band te worden geschreven. Voor de details verwijzen we naar hoofdstuk 11 (boek 3), de pagina's 110 . . . 118. Bij de PLL-afregeling via het display is het de bedoeling dat er eerst gedurende een zekere tijd achter elkaar en ononderbroken syncs naar de band worden gestuurd. Dat gebeurt met het programma DUMCHK van figuur 22, waarvan u de hex dump al bent tegengekomen in hoofdstuk 11,

★ 22



Figuur 22. Het schrijf-hulpprogramma DUMCHK hoort bij de PLL-afregelmetode via het display.

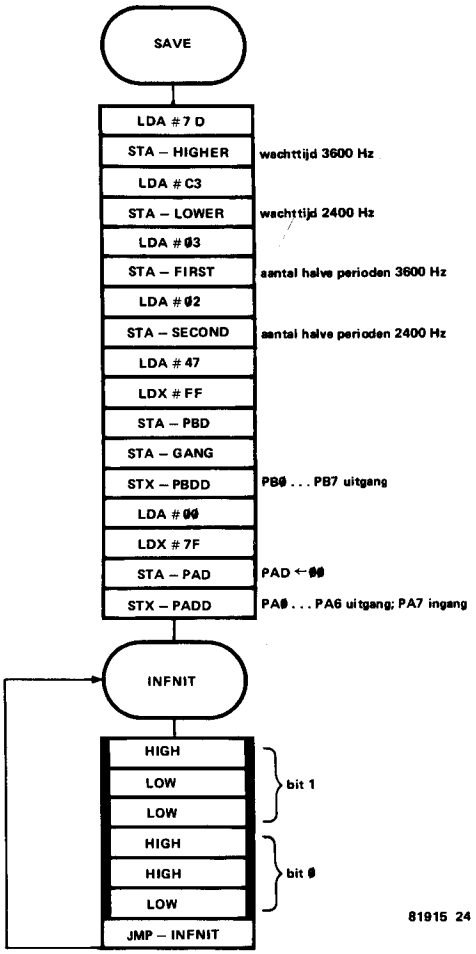
★ 23



Figuur 23. Het lees-hulpprogramma RDCHK moet worden doorlopen tijdens de eigenlijke afregeling van de PLL volgens de display-metode.

namelijk de eerste helft van tabel 2, op pagina 110. Het startadres is $Ø2ØØ. De eerste instrukties van figuur 22 zijn identiek aan de eerste instrukties van DUMP/DUMPT. Pas twee instrukties vóór het label SYNOUT gebeurt er iets dat we nog niet wisten: de geheugenplaatsen CTH en CTL krijgen beide een inhoud $DD toegekend. Het 16-bits getal, gevormd door de inhoud van CTH (linker byte) en CTL (rechter byte) wordt telkens gedurende een omloop rond het label SYNOUT met één verhoogd. Zolang het resultaat van de optelling een carry nul oplevert wordt de optelling gevolgd door het schrijven op de band van een sync (LDA # 16 plus JSR-OUTCH) en een nieuwe optelling (JMP-SYNOUT). Levert de optelling een carry één op, dan volgt de sprong naar het label EXIT: er wordt teruggesprongen naar de standaard-monitor. We zien dan het startadres ($Ø2ØØ) op het display verschijnen.
Wanneer wordt de carry eigenlijk één? Dat gebeurt zodra de inhoud van de geheugenplaats CTH nul is geworden, dus zodra CTL vanuit de stand FF met één wordt verhoogd èn zodra daardoor CTH vanuit de stand FF met één wordt verhoogd, resulterend in een CTH en een CTL nul en een carry één. Dus vlak voor de eerste doorgang van SYNOUT is (CTH, CTL) gelijk aan $DDDD en na afloop is er sprake van het 17 bits getal $1ØØØØ, oftewel decimaal 65.536. Nu is $DDDD gelijk aan het decimale getal 4096x13+256x13+16x13+13=56.797. Met andere woorden: De gang door het label SYNOUT vindt 65.536–56.797=8739 keer plaats. Er worden dus 8738 syncs op de band geschreven. Elke sync duurt 8 bits en elk bit duurt 1250 μs. Hoe we daar aan komen heeft u kunnen lezen in punt 3. Er worden dus gedurende 1250x8x8738 μs = ruim 87 sekonden oftewel ongeveer anderhalve minuut syncs op de band gezet.
Het bijbehorende leesprogramma heet RDCHK en staat in figuur 23. De hex dump vinden we in de tweede helft van tabel 2 (hoofdstuk 11). Het gedeelte van figuur 23 vanaf het label RDCHK (startadres $Ø251) tot aan het label SYB is vrijwel gelijk aan het eerste gedeelte van RDTAPE (figuur 6a), vanaf het label RDTAPE tot aan het label TENSYN. Maar nu dan de instrukties van figuur 23 vanaf het label SYB. Er wordt een karakter ingelezen (JSR-RDCH); de ASCII-kode staat in de accu. Vervolgens gaan we na of dat karakter een sync is. Zoja: terug naar het label SYB. Zo nee: begin opnieuw, bij het label SYNCHK.
De programma's van de figuren 22 en 23 horen bij de PLL-afregelmetode via het display. De vraag rijst wat er wanneer is te zien. Twee instrukties van figuur 23 zijn daarop van invloed. Direkt na afloop van JSR-BTWEEN is het "zoekplaatje" (de drie horizontale segmenten lichten op) te zien. En direkt na afloop van JSR-CHARVU doet de op bit b7 na geïnverteerde accu-inhoud dienst als segmentpatroon. Dus de op bit b7 na geïnverteerde ASCII-kode van het zojuist van de band ingelezen ASCII-karakter. Worden er dus syncs vastgesteld èn is de PLL goed afgeregeld, dan wordt de programmalus rond het label SYB doorlopen. Zolang dàt het geval is toont het display het sync-plaatje. Er kunnen echter nog meer situaties optreden. We lezen een stuk cassette-band waarop (nog) geen syncs (meer) staan. Of we lezen wèl syncs van de band, maar niet in de accu, omdat de PLL niet goed is afgeregeld. In de beide laatste gevallen wordt er, na JSR-CHARVU, een willekeurige, dat wil zeggen foute accu-inhoud gebruikt als segmentpatroon. **Dat duurt echter maar kort en is niet op het display**

★ 24



Figuur 24. Het schrijf-hulpprogramma SAVE, ten behoeve van de PLL-afregelmetode via de oscilloskoop.

zichtbaar, omdat een handvol mikrosekonden later, na de sprong terug naar het label SYNCHK, de subroutine BTWEEN wordt aangeroepen. En dat heeft het displayen van het "zoekplaatje" tot gevolg. Overigens: de zojuist geschetste situatie treedt op in het geval dat er ooit minstens één sync is vastgesteld en dat er later iets anders werd ingelezen. Immers: zolang er niet minstens één sync is vastgesteld wordt de wachtlus rond het label SYA van figuur 23 niet doorbroken. Precies dezelfde wachtlus komt óók voor in RDTAPE.
Vandaar dat er altijd sprake is van òf het "zoekplaatje", òf het syncplaatje. En nóóit van iets anders.

30 De software van het programma SAVE van figuur 24 hoort bij de hulpsoftware die nodig is om de PLL af te regelen volgens de oscilloskoop-metode. De hex dump van deze software bent u in hoofdstuk 11 (boek 3) al tegengekomen; het is tabel 3 op pagina 112.
Het programma SAVE moet ervoor zorgen dat er afwisselend enen en nullen naar de band gaan. Het zal u dan ook niet verbazen dat a) de instrukties van figuur 24 vanaf het label SAVE tot aan het label INFNIT dezelfde zijn als de eerste instrukties van DUMP/DUMPT, en b) er sprake is van een oneindig lange programmalus rond het label INFNIT, met als gevolg dat er een "ja/nee-bittrein" wordt afgeleverd; een kwestie van de juiste volgorde van en hoeveelheden HIGH en LOW kiezen. Verder valt er eigenlijk niets over die figuur 24 te vertellen. Of het moet zijn dat "oneindig lang durende" programmalussen natuurlijk maar een betrekkelijke zaak is: na een paar minuten op de band schrijven drukt men gewoon de RST-toets in. Met als gevolg dat het schrijven ophoudt. Wat nu overigens óók van toepassing is voor dit hoofdstuk 16. Want alles wat bespreekbaar is, is besproken.

De listing van het systeemprogramma PME

De nu volgende pagina's vormen een kombinatie van machinetaal en Engels, die bekend staat als de listing.
Op pagina 189/190 vindt men de instrukties van de routine BINAR en PMBINA, die nodig zijn om (tijdelijk) op binair rekenen terug te schakelen. Voor details: zie Aanhangsel 4 in dit boek.

Enige belangrijke adressen:

EDITC	$1500
BRK	$1533
EDITW	$153D
SEMIW	$1667
SEACND	$17C5
BINAR	$17F6
PMBINA	$17FA

```
0001: 14D8                ORG   $14D8                    14F8 !
0002:
0003:
0004:                     SOURCE LISTING OF THE PM EDITOR
0005:
0006:
0007:                     WRITTEN BY G. NACHBAR
0008:
0009:                     DATE : 25 JUNE 1981
0010:
0011:
0012:                     THE PM EDITOR USES HEXADECIMAL LABELS
0013:
0014:                     ********************************
0015:                     POINTERS AND TEMPS IN PAGE ZERO
0016:                     ********************************
0017:
0018: 14D8                BEGADL *      $00E2
0019: 14D8                BEGADH *      $00E3
0020: 14D8                ENDADL *      $00E4
0021: 14D8                ENDADH *      $00E5
0022: 14D8                CURADL *      $00E6
0023: 14D8                CURADH *      $00E7
0024: 14D8                CENDL  *      $00E8
0025: 14D8                CENDH  *      $00E9
0026: 14D8                TABLEL *      $00EC
0027: 14D8                TABLEH *      $00ED
0028: 14D8                LABELS *      $00EE
0029: 14D8                BYTES  *      $00F6
0030: 14D8                COUNT  *      $00F7
0031: 14D8                INH    *      $00F9
0032: 14D8                POINTL *      $00FA
0033: 14D8                POINTH *      $00FB
0034: 14D8                TEMPX  *      $00FD
0035: 14D8                NIBBLE *      $00FE
0036:
0037:                     ************************************
0038:                     PME'S POINTERS AND TEMPS IN PAGE 1A
0039:                     ************************************
0040:
0041: 14D8                PARAL  *      $1A63
0042: 14D8                PARAH  *      $1A64
0043: 14D8                PARBL  *      $1A65
0044: 14D8                PARBH  *      $1A66
0045: 14D8                NMIL   *      $1A7A
0046: 14D8                NMIH   *      $1A7B
0047: 14D8                BRKTL  *      $1A7C
0048: 14D8                BRKTH  *      $1A7D
0049: 14D8                PBDD   *      $1A83
0050:
0051:
0052:                     ********************************
0053:                     ADDRESSES IN THE STANDARD EPROM
0054:                     ********************************
0055:
0056:
0057: 14D8                SAVE   *      $1C00
0058: 14D8                BEGIN  *      $1ED3
0059: 14D8                OPLEN  *      $1E5C
0060: 14D8                LENACC *      $1E60
0061: 14D8                ADCEND *      $1EDC
0062: 14D8                RECEND *      $1EEA
0063: 14D8                UP     *      $1E83
0064: 14D8                FILLWS *      $1E47
0065: 14D8                NEXT   *      $1EF8
0066: 14D8                ASSEMB *      $1F51
0067:
0068:
```

```
0069:                   ****************
0070:                   SUBROUTINES IN PM
0071:                   ****************
0072:
0073:
0074: 14D8              INPAR  *      $1387
0075: 14D8              RECCHA *      $12AE
0076: 14D8              PRCHA  *      $1334
0077: 14D8              PRBYT  *      $128F
0078: 14D8              PRSP   *      $11F3
0079: 14D8              ASHETT *      $141E
0080: 14D8              CRLF   *      $11E8
0081: 14D8              RESIN  *      $1268
0082: 14D8              RESTTY *      $14BC
0083:
0084: 14D8              STEP   *      $14CF
0085:
0086:
0087:
0088:                   IOCORR: ADAPTS THE PBDD TO THE VERSION D SITUATION,
0089:                   PRIOR TO ASSEMBLING
0090:
0091: 14D8 A9 1E        IOCORR LDAIM $1E
0092: 14DA 8D 83 1A           STA   PBDD   PB0 IS INPUT
0093: 14DD 4C 51 1F           JMP   ASSEMB ASSEMBLE THE PRCGRAM
0094:
0095:                   EDITC: COLD START ENTRY OF PME
0096:
0097: 14E0 20 68 12  EDITC  JSR   RESIN  RESET INPUT BUFFERS
0098: 14E3 A0 00           LDYIM $00
0099: 14E5 20 1E 17        JSR   MESSA  "BEGAD,ENDAD:"
0100: 14E8 20 87 13        JSR   INPAR  GET PARAMETERS
0101: 14EB 30 F3           BMI   EDITC  TRY IT AGAIN, IF NOT DONE PROPERLY
0102: 14ED AD 63 1A        LDA   PARAL  LOAD BEGADL
0103: 14F0 85 E2           STAZ  BEGADL
0104: 14F2 18              CLC
0105: 14F3 69 01           ADCIM $01
0106: 14F5 85 E8           STAZ  CENDL  CENDL = BEGADL+1
0107: 14F7 AD 64 1A        LDA   PARAH
0108: 14FA 85 E3           STAZ  BEGADH LOAD BEGADH
0109: 14FC 69 00           ADCIM $00
0110: 14FE 85 E9           STAZ  CENDH  CENDH = BEGADH+AARRY
0111: 1500 AD 65 1A        LDA   PARBL
0112: 1503 85 E4           STAZ  ENDADL LOAD ENDAD
0113: 1505 AD 66 1A        LDA   PARBH
0114: 1508 85 E5           STAZ  ENDADH LOAD ENDADH
0115: 150A 20 D3 1E        JSR   BEGIN  PRINT POINTER CURAD
0116: 150D A9 77           LDAIM $77    EQUALS BEGAD
0117: 150F A0 00           LDYIM $00
0118: 1511 91 E6           STAIY CURADL EOF 77 ON FIRST ADDRESS BEGAD
0119:
0120:                   BRK: WARM START ENTRY OF PME, INCLUDING THE SPECIFI-
0121:                   CATION OF THE BRK JUMP VECTOR
0122:
0123: 1513 A9 3D        BRK    LDAIM $3D
0124: 1515 8D 7C 1A           STA   BRKTL
0125: 1518 A9 15              LDAIM $15
0126: 151A 8D 7D 1A           STA   BRKTH
0127:
0128:                   EDITW: MAY BE USED AS A WARM START ENTRY OF PME
0129:                   PROVIDED THE BRK JUMP VECTOR HAS BEEN SPECIFIED
0130:                   BEFOREHAND.
0131: 151D A0 20        EDITW  LDYIM $20
0132: 151F 20 1E 17           JSR   MESSA  "PM EDITOR"
0133: 1522 A0 FF              LDYIM $FF
0134: 1524 9A                 TXS          RESET STACK POINTER
0135:
0136:                   THE CENTRAL LABEL "WARM" OF PME STARTS STARTS
0137:                   WITH THE K-KEY ROUTINE
```

```
                        ("DELETE")
                        
1525 20 EB 16   WARM    JSR   PRINS  PRINT CURRENT INSTRUCTION
1528 20 AE 12           JSR   RECCHA WAIT FOR A DEPRESSED KEY
152B C9 4B              CMPIM 'K     IS IT THE K-KEY?
152D D0 09              BNE   LIST   CONTINUE IF NOT
152F 20 83 1E           JSR   UP     ADAPT WORKSPACE
1532 20 EA 1E           JSR   RECEND ADAPT CURRENT END ADDRESS POINTER
1535 4C 25 15           JMP   WARM   READY
                    LABEL LIST: L-KEY ROUTINE
                        
1538 C9 4C      LIST    CMPIM 'L     L-KEY DEPRESSED?
153A D0 12              BNE   SKIP   IF NOT CONTINUE
153C 20 D3 1E           JSR   BEGIN  START AT BEGAD
153F 20 EB 16   LST     JSR   PRINS  PRINT CURRENT INSTRUCTION
1542 20 F8 1E           JSR   NEXT   ADAPT INSTRUCTION POINTER
1545 30 F8              BMI   LST
1547 A0 1F      DONE    LDYIM $1F
1549 20 1E 17   MESS    JSR   MESSA  PRINT "DONE"
154C F0 D7              BEQ   WARM   READY
                        
                    LABEL SKIP: SPACE BAR KEY ROUTINE
                    LABEL INSERT: I-KEY ROUTINE
                        
154E C9 20      SKIP    CMPIM        SPACE BAR DEPRESSED?
1550 D0 07              BNE   INSERT CONTINUE IF NOT
1552 20 F8 1E           JSR   NEXT   ADAPT INSTRUCTION POINTER
1555 30 CE              BMI   WARM   READY
1557 10 EE              BPL   DONE   PRINT "DONE" IF NO FURTHER INSTR. AVAILABLE
1559 C9 49      INSERT  CMPIM 'I     I-KEY DEPRESSED?
155B D0 17              BNE   SEARCH IF NOT CONTINUE
155D 20 A9 16           JSR   READIN READ NEW INSTRUCTION
1560 30 0A              BMI   ILLKEY PRINT "ILLEGAL KEY" IF NEW ISTR. ISN'T VALID
1562 20 D6 16   MEM     JSR   CHECK
1565 90 09              BCC   FULL   PRINT "FULL" IF MEMORY IS FULL
1567 20 47 1E           JSR   FILLWS LOAD NEW INSTR. IN MEMORY
156A F0 B9              BEQ   WARM   READY
156C A0 0E      ILLKEY  LDYIM $0E
156E D0 D9              BNE   MESS   PRINT "ILLEGAL KEY"
1570 A0 1A      FULL    LDYIM $1A
1572 D0 D5              BNE   MESS   PRINT "FULL"
                        
                    LABEL SEARCH: S-KEY ROUTINE
                    ANY BYTE PATTERN (INSTRUCTION) MAY BE SEARCHED FOR.
                    IF THE INSTRUCTION SEARCHED FOR IS FOUND,
                    ONE MAY SEARCH FOR THE SAME INSTRUCTION ON A HIGHER
                    ADDRESS, BY PRESSING THE KEY "Y". THE SEARCHING PRO-
                    CESS ALWAYS HAS TO BE CONCLUDED BY THE MESSAGE "DONE".
                    THIS OCCURS EITHER BECAUSE ALL OR NO INSTRUCTIONS ARE
                    FOUND, OR BY PRESSING AN ARBITRARY SOFTWARE-KEY (I.E.
                    A KEY GENERATING AN ASCII CODE WHEN DEPRESSED),
                    WITH THE EXEPTION OF THE 'Y'-KEY.
1574 C9 53      SEARCH  CMPIM 'S     S-KEY DEPRESSED?
1576 D0 40              BNE   BACK    IF NOT CONTINUE
1578 20 A9 16           JSR   READIN READ INSTR. TO BE SEARCHED FOR
157B 30 EF              BMI   ILLKEY "ILLEGAL KEY" IF NOT PROPERLY DONE
157D 20 D3 1E           JSR   BEGIN  START AT BEGAD
1580 A5 FB      SCAN    LDAZ  POINTH GET OPCODE OF SEARCH INSTR.
1582 20 60 1E           JSR   LENACC GET LENGTH OF SEARCH INSTR.
1585 A0 00              LDYIM $00
1587 B1 E6              LDAIY CURADL GET OPCODE OF THE CURRENT INSTR.
1589 C5 FB              CMPZ  POINTH COMPARE IT AGAINST OPCODE OF THE SEARCH INST.
158B D0 20              BNE   AGAIN  IF NO MATCH THEN NEXT INSTR.
158D C6 F6              DECZ  BYTES
158F F0 12              BEQ   FOUND  CONTINUE, IF INSTR. LENGTH IS 2 OR 3
1591 C8                 INY
1592 B1 E6              LDAIY CURADL GET NEXT BYTE IN MEMORY
1594 C5 FA              CMPZ  POINTL COMP. IT AGAINST 1ST OPERAND BYTE
1596 D0 15              BNE   AGAIN  IF NO MATCH NEXT INSTR.
```

```
0207: 1598 C6 F6          DECZ  BYTES
0208: 159A F0 07          BEQ   FOUND  CONTINUE IF LENGTH OF CURR. INSTR. IS 3
0209: 159C C8             INY
0210: 159D B1 E6          LDAIY CURADL GET NEXT BYTE IN MEMORY
0211: 159F C5 F9          CMPZ  INH    COMP. IT AGAINST 2ND OPERAND BYTE
0212: 15A1 D0 0A          BNE   AGAIN  IF NO MATCH NEXT INSTR.
0213: 15A3 20 EB 16  FOUND JSR   PRINS  PRINT OUT THE SPECIFIED INSTRUCTION
0214: 15A6 20 AE 12       JSR   RECCHA WAIT FOR A DEPRESSED KEY
0215: 15A9 C9 59          CMPIM 'Y     Y-KEY DEPRESSED?
0216: 15AB D0 08          BNE   DNE     IF NOT PRINT "DONE"
0217: 15AD 20 5C 1E  AGAIN JSR   OPLEN  GET LENGTH OF THE CURR. INSTR.
0218: 15B0 20 F8 1E       JSR   NEXT   ADAPT INSTR. POINTER
0219: 15B3 30 CB          BMI   SCAN   CONTINUE SEARCH IF STILL INSTR. AVAILAB
0220: 15B5 4C 47 15  DNE  JMP   DONE   ELSE PRINT "DONE"
0221:
0222:                     LABEL BACK: Z-KEY ROUTINE
0223:
0224: 15B8 C9 5A     BACK CMPIM 'Z     Z-KEY DEPRESSED?
0225: 15BA D0 30          BNE   TOF    IF NOT CONTINUE
0226: 15BC A5 E2          LDAZ  BEGADL
0227: 15BE 85 EC          STAZ  TABLEL
0228: 15C0 A5 E3          LDAZ  BEGADH
0229: 15C2 85 ED          STAZ  TABLEH TABLE:=BEGAD
0230: 15C4 C5 E7          CMPZ  CURADH  TABLEH:=CURADH?
0231: 15C6 D0 08          BNE   BCKA   IF NOT INCREASE TABLE
0232: 15C8 A5 E2          LDAZ  BEGADL
0233: 15CA C5 E6          CMPZ  CURADL TABLEL:=CURADL?
0234: 15CC D0 02          BNE   BCKA   IF NOT INCREASE TABLE
0235: 15CE F0 19          BEQ   BCKB   ELSE PRINT FIRST INSTRUCTION
0236: 15D0 A0 00     BCKA LDYIM $00
0237: 15D2 B1 EC          LDAIY TABLEL GET LENGTH OF INSTR. AT
0238: 15D4 20 60 1E       JSR   LENACC WHICH POINT IS POINTED
0239: 15D7 20 80 16       JSR   INCTAB INCREASE POINT BY BYTES
0240: 15DA A5 E6          LDAZ  CURADL
0241: 15DC C5 EC          CMPZ  TABLEL TABLEL:=CURADL?
0242: 15DE D0 F0          BNE   BCKA   IF NOT INCREASE TABLE
0243: 15E0 A5 E7          LDAZ  CURADH
0244: 15E2 C5 ED          CMPZ  TABLEH TABLEH:=CURADH
0245: 15E4 D0 EA          BNE   BCKA   IF NOT INCREASE TABLE
0246: 15E6 20 72 16       JSR   DECURA DECREASE INSTRUCTION POINTER BY BYTES
0247: 15E9 4C 25 15  BCKB JMP   WARM
0248:
0249:                     LABEL TOF: T-KEY ROUTINE
0250:                     LABEL SXTEEN: P-KEY ROUTINE
0251:
0252: 15EC C9 54     TOF  CMPIM 'T     T-KEY DEPRESSED?
0253: 15EE D0 06          BNE   SXTEEN IF NOT CONTINUE
0254: 15F0 20 D3 1E       JSR   BEGIN  INSTR. POINTER = BEGAD
0255: 15F3 4C 25 15  TOFEND JMP   WARM   READY
0256: 15F6 C9 50     SXTEEN CMPIM 'P     P-KEY DEPRESSED?
0257: 15F8 D0 1C          BNE   ASMBLR IF NOT CONTINUE
0258: 15FA A9 0F          LDAIM $0F    15 INSTR. TO BE PRINTED
0259: 15FC 85 F7          STAZ  COUNT
0260: 15FE 20 5C 1E  LINES JSR   OPLEN  GET LENGTH OF CURRENT INSTR.
0261: 1601 20 F8 1E       JSR   NEXT   ADAPT INSTR. POINTER
0262: 1604 C6 F7          DECZ  COUNT
0263: 1606 F0 EB          BEQ   TOFEND READY IF 15 INSTR. HAVE BEEN PRINTED
0264: 1608 A0 00          LDYIM $00
0265: 160A B1 E6          LDAIY CURADL
0266: 160C C9 77          CMPIM $77
0267: 160E F0 E3          BEQ   TOFEND READY IF OPCODE IS 77
0268: 1610 20 EB 16       JSR   PRINS  PRINT INSTRUCTION
0269: 1613 4C FE 15       JMP   LINES  NEXT INSTRUCTION
0270:                     LABEL ASMBLR: X-KEY ROUTINE
0271:                     LABEL ASSEND: RESTORE I/O AFTER ASSEMBLING
0272:
0273: 1616 C9 58     ASMBLR CMPIM 'X     X-KEY DEPRESSED?
0274: 1618 D0 13          BNE   INPUT  IF NOT CONTINUE
0275: 161A A9 27          LDAIM ASSEND SPECIFY NMI JUMP VECTOR
```

```
0276: 161C 8D 7A 1A          STA   NMIL
0277: 161F A9 16             LDAIM ASSEND /256
0278: 1621 8D 7B 1A          STA   NMIH
0279: 1624 4C D8 14          JMP   IOCORR PREPARE I/O PRIOR TO ASSEMBLING
0280: 1627 20 BC 14  ASSEND  JSR   RESTTY RESTORE I/O: PM SITUATION
0281: 162A 4C 68 17          JMP   LABLST LIST THE HEXADECIMAL LABELS
0282:
0283:                        LABEL INPUT: I-KEY ROUTINE
0284:                        NOTE: NO NON-NUMERICAL KEY HAS TO BE DEPRESSED
0285:                        PME AUTOMATICALLY ASSUMES THE INPUT KEY FUNCTION
0286:                        HAS BEEN OPTED FOR AS SOON AS THE DATA BELONGING
0287:                        TO THE NEW INSTR. HAS BEEN SPECIFIED
0288:
0289: 162D 20 91 16  INPUT   JSR   BYT    GET THE 2ND NIBBLE OF THE 1ST BYTE
0290: 1630 30 12             BMI   WRONG
0291: 1632 20 AE 16          JSR   READ   GET THE OTHER BYTE(S)
0292: 1635 30 0D             BMI   WRONG
0293: 1637 20 5C 1E          JSR   OPLEN  GET LENGTH OF THE CURRRENT INSTR.
0294: 163A 20 F8 1E          JSR   NEXT   ADAPT INSTR. POINTER
0295: 163D A5 FD             LDAZ  TEMPX  GET LENGTH OF NEW INSTR.
0296: 163F 85 F6             STAZ  BYTES  AND STORE IT IN BYTES
0297: 1641 4C 62 15          JMP   MEM    LOAD MEMORY WITH NEW INSTRUCTION
0298: 1644 4C 6C 15  WRONG   JMP   ILLKEY PRINT "ILLEGAL KEY"
0299:
0300:                        END OF THE PME MAIN ROUTINE
0301:
0302:
0303:
0304:                        SEMI WARM START ("LUKE WARM") ENTRY OF PME
0305:                        FOLLOWING THE SPECIFICATION BY THE USER
0306:                        OF BEGAD AND ENDAD THE INSTRUCTION POINTER
0307:                        CURAD IS POINTED AT BEGAD AND THE CURRENT
0308:                        END ADDRESS POINTER CEND IS POINTED AT ENDAD.
0309:                        AFTER THIS PME IS ENTERED BY THE WARM START ENTRY
0310:
0311:
0312: 1647 20 68 12  SEMIW   JSR   RESIN  RESET INPUT BUFFERS
0313: 164A A0 00             LDYIM $00
0314: 164C 20 1E 17          JSR   MESSA  "BEGAD,ENDAD:"
0315: 164F 20 87 13          JSR   INPAR  GET BEGAD AND ENDAD
0316: 1652 30 F3             BMI   SEMIW  TRY IT AGAIN IF NOT DONE PROPERLY
0317: 1654 AD 63 1A          LDA   PARAL
0318: 1657 85 E2             STAZ  BEGADL
0319: 1659 AD 64 1A          LDA   PARAH
0320: 165C 85 E3             STAZ  BEGADH
0321: 165E AD 65 1A          LDA   PARBL
0322: 1661 85 E4             STAZ  ENDADL
0323: 1663 85 E8             STAZ  CENDL
0324: 1665 AD 66 1A          LDA   PARBH
0325: 1668 85 E5             STAZ  ENDADH
0326: 166A 85 E9             STAZ  CENDH  CEND = ENDAD
0327: 166C 20 D3 1E          JSR   BEGIN  CURAD = BEGAD
0328: 166F 4C 13 15          JMP   BRK    JUMP TO WARM START
0329:
0330:
0331:
0332:                        ******************
0333:                        SUBROUTINES OF PME
0334:                        ******************
0335:
0336:
0337:                        DECURA
0338:                        THE INSTRUCTION POINTER IS DECREASED BY THE
0339:                        NUMBER OF BYTES - BEING THE LENGTH OF THE
0340:                        INSTRUCTION - WHICH PRECEEDS THE CURRENT
0341:                        INSTRUCTION IN MEMORY.
0342:
0343: 1672 38              DECURA SEC
0344: 1673 A5 E6                 LDAZ  CURADL
```

```
0345: 1675 E5 F6            SBCZ  BYTES
0346: 1677 85 E6            STAZ  CURADL CURADL:=CURADL-BYTES
0347: 1679 A5 E7            LDAZ  CURADH
0348: 167B E9 00            SBCIM $00
0349: 167D 85 E7            STAZ  CURADH CURADH:=CURADH-BORROW
0350: 167F 60               RTS
0351:
0352:
0353:                  SUBROUTINR INCTAB
0354:                  INCRESES THE POINTER TABLE BY THE AMOUNT DETERMINED
0355:                  BY THE CONTENTS OF BYTES (INSTRUCTION LENGTH)
0356:
0357: 1680 18         INCTAB CLC
0358: 1681 A5 EC            LDAZ  TABLEL
0359: 1683 65 F6            ADCZ  BYTES
0360: 1685 85 EC            STAZ  TABLEL TABLE:=TABLE + BYTES
0361: 1687 A5 ED            LDAZ  TABLEH
0362: 1689 69 00            ADCIM $00
0363: 168B 85 ED            STAZ  TABLEH
0364: 168D 60               RTS
0365:
0366:                  THE SUBROUTINE BYTIN  LOADS THE ACCU WITH
0367:                  WITH DATA WHICH BELONGS TO TWO NUMERICAL KEYS
0368:                  DEPRESSED. RETURNING FROM SUBROUTINE N=1 AND
0369:                  Z=0 IF A NON-NUMERICAL KEY WAS DEPRESSED. TWO
0370:                  SUCCESIVELY DEPRESSED NUMERICAL KEYS WILL
0371:                  RESULT INTO N=0 AND Z=1
0372:
0373: 168E 20 AE 12  BYTIN  JSR   RECCHA WAIT FOR A DEPRESSED KEY
0374: 1691 20 1E 14  BYT    JSR   ASHETT CONVERT IT TO A DATA NIBBLE
0375: 1694 30 12            BMI   RETURN ERROR EXIT IF KEY <> 0...F
0376: 1696 0A               ASLA         ENTERED DATA IS NEW HIGHER DATA NIBB
0377: 1697 0A               ASLA
0378: 1698 0A               ASLA
0379: 1699 0A               ASLA
0380: 169A 85 FE            STAZ  NIBBLE SAVE HIGHER DATA NIBBLE
0381: 169C 20 AE 12         JSR   RECCHA WAIT FOR A DEPRESSED KEY
0382: 169F 20 1E 14         JSR   ASHETT CONVERT IT TO A DATA NIBBLE
0383: 16A2 30 04            BMI   RETURN ERROR EXIT IF KEY <> 0...F
0384: 16A4 05 FE            ORAZ  NIBBLE ISERT NEW LOWER DATA NIBBLE
0385: 16A6 A2 00            LDXIM $00    RESET N-FLAG, SET Z-FLAG
0386: 16A8 60          RETURN RTS
0387:
0388:
0389:                  THE SUBROUTINE READIN LOADS EITHER ONE,
0390:                  TWO OR THREE DATA BUFFERS DEPENDING
0391:                  OF THE INSTRUCTION LENGTH SPECIFIED BY
0392:                  THE OPCODE.
0393:                  NORMAL EXIT: Z=1, N=0
0394:                  ERROR EXIT: Z=0, N=1
0395:
0396: 16A9 20 8E 16  READIN JSR   BYTIN  WAIT FOR OPCODE
0397: 16AC 30 27            BMI   RDB    ERROR IF KEY <> 0...F
0398: 16AE 85 FB       READ  STAZ  POINTH STORE OPCODE IN POINTH
0399: 16B0 20 60 1E         JSR   LENACC GET INSTRUCTION LENGTH
0400: 16B3 84 F7            STYZ  COUNT  AND COPY IT INTO COUNT
0401: 16B5 84 FD            STYZ  TEMPX  AS WELL AS IN TEMPX
0402: 16B7 C6 F7            DECZ  COUNT
0403: 16B9 F0 18            BEQ   RDA    RETURN IF ONE BYTE INSTR.
0404: 16BB 20 F3 11         JSR   PRSP   PRINT A SPACE
0405: 16BE 20 8E 16         JSR   BYTIN  WAIT FOR (1ST) OPERAND
0406: 16C1 30 12            BMI   RDB    ERROR IF KEY <> 0...F
0407: 16C3 85 FA            STAZ  POINTL STORE (1ST) OPERND IN POINTL
0408: 16C5 C6 F7            DECZ  COUNT
0409: 16C7 F0 0A            BEQ   RDA    RETURN IF TWO BYTE INSTR.
0410: 16C9 20 F3 11         JSR   PRSP   PRINT A SPACE
0411: 16CC 20 8E 16         JSR   BYTIN  WAIT FOR 2ND OPERAND
0412: 16CF 30 04            BMI   RDB    ERROR IF KEY <> 0...F
0413: 16D1 85 F9            STAZ  INH    STORE 2ND OPERAND IN INH
```

```
0414: 16D3 A2 00      RDA    LDXIM $00     Z=1, N=0
0415: 16D5 60         RDB    RTS
0416:
0417:
0418:                        THE SUBROUTINE CHECK VERIFIES IF THERE IS STILL
0419:                        ROOM IN THE WORKSPACE AREA FOR A NEW INSTRUCTION.
0420:                        THE WORKSPACE IS LIMITED BY BEGAD AND ENDAD.
0421:                        CHECK RETURNS WITH:
0422:                        C=0 IF THERE IS NO ROOM FOR THE INSTR.
0423:                        C=1 IF THERE IS ROOM FOR THE NEW INSTRUCTION
0424:
0425: 16D6 20 DC 1E  CHECK  JSR    ADCEND ADJUST CURRENT ENDADDRESS POINTER
0426: 16D9 38                 SEC
0427: 16DA A5 E4              LDAZ   ENDADL
0428: 16DC E9 02              SBCIM  $02
0429: 16DE E5 E8              SBCZ   CENDL
0430: 16E0 A5 E5              LDAZ   ENDADH CARRY DETERMINED BY ENDAD-2-CEND
0431: 16E2 E5 E9              SBCZ   CENDH
0432: 16E4 90 04              BCC    CHKEND EXIT IF NO ROOM ANYMORE
0433: 16E6 20 EA 1E           JSR    RECEND READJUST CURRENT END ADDRESS POINTER
0434: 16E9 38                 SEC
0435: 16EA 60         CHKEND  RTS
0436:
0437:
0438:                        THE SUBROUTINE PRINS PRINTS AN INSTRUCTION
0439:                        SPECIFIED BY THE INSTRUCTION POINTER CURAD.
0440:                        THE CONTENTS OF THE INSTRUCTION POINTER IS ALSO
0441:                        PRINTED. BY PRINTING A CERTAIN NUMBER OF SPACES
0442:                        THE POSITION OF THE CARRIAGE IS IDENTICAL TO THE
0443:                        FIRST POSITION OF THE SYSTEM COMMAND COLUMN.
0444:
0445:
0446: 16EB 20 E8 11  PRINS  JSR    CRLF   PRINT A NEW LINE
0447: 16EE 20 5C 1E           JSR    OPLEN  GET LENGTH OF INSTRUCTION
0448: 16F1 A6 F6              LDXZ   BYTES  TO BE PRINTED
0449: 16F3 A5 E7              LDAZ   CURADH
0450: 16F5 20 8F 12           JSR    PRBYT  PRINT HIGHER ADDRESS BYTE
0451: 16F8 A5 E6              LDAZ   CURADL
0452: 16FA 20 8F 12           JSR    PRBYT  PRINT LOWER ADDRESS BYTE
0453: 16FD A9 0F              LDAIM  $0F
0454: 16FF 85 EE              STAZ   LABELS MAXIMUM NUMBER OF SPACES
0455: 1701 A0 00              LDYIM  $00
0456: 1703 20 F3 11  PRT      JSR    PRSP   PRINT A SPACE
0457: 1706 B1 E6              LDAIY  CURADL GET BYTE TO BE PRINTED
0458: 1708 20 8F 12           JSR    PRBYT  AND PRINT IT
0459: 170B 38                 SEC
0460: 170C A5 EE              LDAZ   LABELS DECREASE NUMBER OF SPACES
0461: 170E E9 03              SBCIM  $03    TO BE PRINTED
0462: 1710 85 EE              STAZ   LABELS BY 3
0463: 1712 C8                 INY           SET UP FOR NEXT BYTE
0464: 1713 CA                 DEX           TO BE PRINTED
0465: 1714 D0 ED              BNE    PRT    IF THERE IS ANYONE
0466: 1716 20 F3 11  SP       JSR    PRSP   PRINT A NUMBER OF SPACES
0467: 1719 C6 EE              DECZ   LABELS
0468: 171B D0 F9              BNE    SP
0469: 171D 60                 RTS           RETURN
0470:
0471:
0472:                        THE SUBROUTINE MESSA IS IDENTICAL WITH THE
0473:                        PM SUBROUTINE MESSY, EXEPT FOR A DIFFERENT
0474:                        LOOKUP TABLE
0475:
0476:
0477: 171E 20 E8 11  MESSA  JSR    CRLF   PRINT A NEW LINE
0478: 1721 B9 30 17  MA       LDAY   TXT    GET CHARACTER TO BE PRINTED
0479: 1724 C9 03              CMPIM  $03    EOT CHARACTER?
0480: 1726 F0 07              BEQ    TXTEND IF YES RETURN
0481: 1728 20 34 13           JSR    PRCHA  PRINT A CHARACTER
0482: 172B C8                 INY           SET UP FOR NEXT CHARACTER
```

```
0483: 172C 4C 21 17          JMP   MA
0484: 172F 60          TXTEND RTS                READY
0485:
0486:
0487:                        LOOKUP TABLE TXT
0488:
0489: 1730 42          TXT    =      'B          Y=00
0490: 1731 45                 =      'E
0491: 1732 47                 =      'G
0492: 1733 41                 =      'A
0493: 1734 44                 =      'D
0494: 1735 2C                 =      ',
0495: 1736 45                 =      'E
0496: 1737 4E                 =      'N
0497: 1738 44                 =      'D
0498: 1739 41                 =      'A
0499: 173A 44                 =      'D
0500: 173B 3A                 =      ':
0501: 173C 20                 =      '
0502: 173D 03                 =      $03
0503: 173E 49                 =      'I          Y=0E
0504: 173F 4C                 =      'L
0505: 1740 4C                 =      'L
0506: 1741 45                 =      'E
0507: 1742 47                 =      'G
0508: 1743 41                 =      'A
0509: 1744 4C                 =      'L
0510: 1745 20                 =      '
0511: 1746 4B                 =      'K
0512: 1747 45                 =      'E
0513: 1748 59                 =      'Y
0514: 1749 03                 =      $03
0515: 174A 46                 =      'F          Y=1A
0516: 174B 55                 =      'U
0517: 174C 4C                 =      'L
0518: 174D 4C                 =      'L
0519: 174E 03                 =      $03
0520: 174F 44                 =      'D          Y=1F
0521: 1750 4F                 =      'O
0522: 1751 4E                 =      'N
0523: 1752 45                 =      'E
0524: 1753 03                 =      $03
0525: 1754 50                 =      'P          Y=24
0526: 1755 4D                 =      'M
0527: 1756 20                 =      '
0528: 1757 45                 =      'E
0529: 1758 44                 =      'D
0530: 1759 49                 =      'I
0531: 175A 54                 =      'T
0532: 175B 4F                 =      'O
0533: 175C 52                 =      'R
0534: 175D 03                 =      $03
0535: 175E 4C                 =      'L          Y=2E
0536: 175F 41                 =      'A
0537: 1760 42                 =      'B
0538: 1761 20                 =      '
0539: 1762 24                 =      '$
0540: 1763 03                 =      $03
0541: 1764 3A                 =      ':          Y=34
0542: 1765 20                 =      '
0543: 1766 24                 =      '$
0544: 1767 03                 =      $03
0545:
0546:                        THE INSTRUCTIONS FOLLOWING THE LABEL LABLST
0547:                        DEAL WITH THE PRINTING OF ALL HEXADECIMAL LABELS
0548:                        AFTER THE PROGRAM HAS BEEN ASSEMBLED
0549:
0550:
0551: 1768 A0 FF     LABLST LDYIM $FF
```

```
0552: 176A 20 E8 11  LBLSTA JSR   CRLF    PRINT ON A NEW LINE
0553: 176D A2 04             LDXIM $04    4 LABELS ON EACH LINE
0554: 176F 84 FD     LBLSTB STYZ  TEMPX   SAVE Y
0555: 1771 A0 2E             LDYIM $2E    "LAB $"
0556: 1773 20 21 17          JSR   MA
0557: 1776 A4 FD             LDYZ  TEMPX  RESTORE Y
0558: 1778 B1 EC             LDAIY TABLEL GET LABEL NUMBER
0559: 177A 20 8F 12          JSR   PRBYT  PRINT LABEL NUMBER
0560: 177D 88                DEY
0561: 177E 84 FD             STYZ  TEMPX  SAVE Y
0562: 1780 A0 34             LDYIM $34    ": $"
0563: 1782 20 21 17          JSR   MA
0564: 1785 A4 FD             LDYZ  TEMPX  RESTORE Y
0565: 1787 B1 EC             LDAIY TABLEL GET HIGHER ADDRESS BYTE
0566: 1789 20 8F 12          JSR   PRBYT  AND PRINT IT
0567: 178C 88                DEY
0568: 178D B1 EC             LDAIY TABLEL GET LOWER ADDRESS BYTE
0569: 178F 20 8F 12          JSR   PRBYT  AND PRINT IT
0570: 1792 20 F3 11          JSR   PRSP   PRINT A SPACE
0571: 1795 88                DEY          SETUP FOR NEXT LABEL
0572: 1796 C4 EE             CPYZ  LABELS ALL LABELS PRINTED
0573: 1798 F0 05             BEQ   LBLSTC IF NOT SET UP FOR NEXT LABEL
0574: 179A CA                DEX
0575: 179B D0 D2             BNE   LBLSTB ON THE SAME LINE
0576: 179D F0 CB             BEQ   LBLSTA OR ON A NEW LINE
0577: 179F 20 D3 1E  LBLSTC JSR   BEGIN   INSTRUCTION POINTER = BEGAD
0578: 17A2 4C 1D 15          JMP   EDITW  WARM START ENTRY OF PME WITHOUT
0579:                                     BRK JUMP VECTOR SPECIFICATION
0580:
0581:
0582:
0583:                        THE WARM CEND ENTRY (LABEL SEACND) OF PME
0584:                        RESTORES THE POSITION OF THE CURRENT END
0585:                        ADDRESS POINTER CEND, AFTER SPECIFICATION,
0586:                        BY THE USER, OF BEGAD AND ENDAD, AND AFTER FINDING A
0587:                        PSEUDO OPCODE 77, I.E. AN EOF CHARACTER. THIS IS
0588:                        FOLLOWED BY A JUMP TO THE WARM START ENTRY OF PME.
0589:
0590:
0591: 17A5 20 68 12  SEACND JSR   RESIN   RESET INPUT BUFFERS
0592: 17A8 A0 00             LDYIM $00
0593: 17AA 20 1E 17          JSR   MESSA  "BEGAD,ENDAD"
0594: 17AD 20 87 13          JSR   INPAR  GET BEGAD AND ENDAD
0595: 17B0 30 F3             BMI   SEACND TRY IT AGAIN IF NOT PROPERLY DONE
0596: 17B2 20 D3 1E          JSR   BEGIN   INSTR. POINTER=BEGAD
0597: 17B5 A0 00     SCNDA   LDYIM $00
0598: 17B7 B1 E6             LDAIY CURADL GET CURRENT OPCODE
0599: 17B9 C9 77             CMPIM $77    IS IT OPCODE 77?
0600: 17BB F0 09             BEQ   SCNDB  IF NOT CONTINUE
0601: 17BD 20 60 1E          JSR   LENACC GET CURRENT INSTR. LENGTH
0602: 17C0 20 F8 1E          JSR   NEXT   ADJUST INSTR. POINTER
0603: 17C3 4C B5 17          JMP   SCNDA  AND CHECK AGAIN FOR OPCODE 77
0604: 17C6 18        SCNDB   CLC
0605: 17C7 A5 E6             LDAZ  CURADL
0606: 17C9 69 01             ADCIM $01
0607: 17CB 85 E8             STAZ  CENDL  CENDL:=CURADL+1
0608: 17CD A5 E7             LDAZ  CURADH
0609: 17CF 69 00             ADCIM $00
0610: 17D1 85 E9             STAZ  CENDH  CENDH:=CURADH+CARRY
0611: 17D3 4C 13 15          JMP   BRK    WARM START ENTRY OF PME
0612:
0613:
0614:                        THE INSTRUCTIONS FOLLOWING THE LABEL BINAR
0615:                        AND THE INSTRUCTIONS FOLLOWING THE LABEL PMBINA ARE
0616:                        AN EXTENSION OF THE STANDARD MONITOR AND OF PM
0617:                        RESPECTIVELY. IN CASE OF SINGLE STEPPING THROUGH
0618:                        AN USER PROGRAM WITH DECIMAL ARITHMETIC, THERE
0619:                        WILL BE A TEMPORARY SWTCH BACK ON BINARY ARITHMETIC
0620:                        PROVIDED THE NMI JUMP VECTOR HAS BEEN SPECIFIED
```

```
0621:                   ON 17F6 OR 17FA RESPECTIVELY.
0622:
0623:
0624: 17D6 D8           BINAR  CLD           BINARY ARITHMETIC
0625: 17D7 4C 00 1C            JMP   SAVE    SAVE INPUT PROPER
0626: 17DA D8           PMBINA CLD           BINARY ARITHMETIC
0627: 17DB 4C CF 14            JMP   STEP    SAVE INPUT PROPER
0628:
0629:
0630:                   ******* END OF THE PME PROGRAM *******
0631:
0632:
0633:                   VICINUS ELABORABAT PROGRAMMAM XXV VI MCMLXXXI
0634:                   VIR NOCTIS IMPOSIT PROGRMMAM IN MACHINAM
0635:
ID=

-T

    SYMBOL TABLE 3000 3276

    ADCEND 1EDC    AGAIN  15AD    ASHETT 141E    ASMBLR 1616
    ASSEMB 1F51    ASSEND 1627    BACK   15B8    BCKA   15D0
    BCKB   15E9    BEGADH 00E3    BEGADL 00E2    BEGIN  1ED3
    BINAR  17D6    BRKTH  1A7D    BRKTL  1A7C    BRK    1513
    BYTES  00F6    BYTIN  168E    BYT    1691    CENDH  00E9
    CENDL  00E8    CHECK  16D6    CHKEND 16EA    COUNT  00F7
    CRLF   11E8    CURADH 00E7    CURADL 00E6    DECURA 1672
    DNE    15B5    DONE   1547    EDITC  14E0    EDITW  151D
    ENDADH 00E5    ENDADL 00E4    FILLWS 1E47    FOUND  15A3
    FULL   1570    ILLKEY 156C    INCTAB 1680    INH    00F9
    INPAR  1387    INPUT  162D    INSERT 1559    IOCORR 14D8
    LABELS 00EE    LABLST 1768    LBLSTA 176A    LBLSTB 176F
    LBLSTC 179F    LENACC 1E60    LINES  15FE    LIST   1538
    LST    153F    MA     1721    MEM    1562    MESS   1549
    MESSA  171E    NEXT   1EF8    NIBBLE 00FE    NMIH   1A7B
    NMIL   1A7A    OPLEN  1E5C    PARAH  1A64    PARAL  1A63
    PARBH  1A66    PARBL  1A65    PBDD   1A83    PMBINA 17DA
    POINTH 00FB    POINTL 00FA    PRBYT  128F    PRCHA  1334
    PRINS  16EB    PRSP   11F3    PRT    1703    RDA    16D3
    RDB    16D5    READ   16AE    READIN 16A9    RECCHA 12AE
    RECEND 1EEA    RESIN  1268    RESTTY 14BC    RETURN 16A8
    SAVE   1C00    SCAN   1580    SCNDA  17B5    SCNDB  17C6
    SEACND 17A5    SEARCH 1574    SEMIW  1647    SKIP   154E
    SP     1716    STEP   14CF    SXTEEN 15F6    TABLEH 00ED
    TABLEL 00EC    TEMPX  00FD    TOFEND 15F3    TOF    15EC
    TXTEND 172F    TXT    1730    UP     1E83    WARM   1525
    WRONG  1644
```

Hex dump van het systeemprogramma PME

De aanvullende software van ESS 507N (voormalige PM-EPROM, nu de PM/PME-EPROM) omvat de adressen $14F8 . . . $17FF. Het leeuwedeel daarvan, namelijk de adressen $14F8 . . . $17F5, staat in het teken van PME. De adressen $17F6 . . . $17FD horen bij de twee routines van Aanhangsel 4.
N.B. Voor de eerste helft van de hex dump van de PM/PME-EPROM: zie Aanhangsel 5 op de pagina's 206 en 207 van boek 3.Regel 14FØ is in beide gevallen weergegeven, zij het dat in Aanhangsel 5 van boek 3 uitsluitend de eerste ~~drie~~ PM-bytes zijn weergegeven.

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
14F0:	F9	4C	A2	13	FF	FF	FF	FF	A9	1E	8D	83	1A	4C	51	1F
1500:	20	68	12	A0	00	20	3E	17	20	87	13	30	F3	AD	63	1A
1510:	85	E2	18	69	01	85	E8	AD	64	1A	85	E3	69	00	85	E9
1520:	AD	65	1A	85	E4	AD	66	1A	85	E5	20	D3	1E	A9	77	A0
1530:	00	91	E6	A9	3D	8D	7C	1A	A9	15	8D	7D	1A	A0	24	20
1540:	3E	17	A2	FF	9A	20	0B	17	20	AE	12	C9	4B	D0	09	20
1550:	83	1E	20	EA	1E	4C	45	15	C9	4C	D0	12	20	D3	1E	20
1560:	0B	17	20	F8	1E	30	F8	A0	1F	20	3E	17	F0	D7	C9	20
1570:	D0	07	20	F8	1E	30	CE	10	EE	C9	49	D0	17	20	C9	16
1580:	30	0A	20	F6	16	90	09	20	47	1E	F0	B9	A0	0E	D0	D9
1590:	A0	1A	D0	D5	C9	53	D0	40	20	C9	16	30	EF	20	D3	1E
15A0:	A5	FB	20	60	1E	A0	00	B1	E6	C5	FB	D0	20	C6	F6	F0
15B0:	12	C8	B1	E6	C5	FA	D0	15	C6	F6	F0	07	C8	B1	E6	C5
15C0:	F9	D0	0A	20	0B	17	20	AE	12	C9	59	D0	08	20	5C	1E
15D0:	20	F8	1E	30	CB	4C	67	15	C9	5A	D0	30	A5	E2	85	EC
15E0:	A5	E3	85	ED	C5	E7	D0	08	A5	E2	C5	E6	D0	02	F0	19
15F0:	A0	00	B1	EC	20	60	1E	20	A0	16	A5	E6	C5	EC	D0	F0
1600:	A5	E7	C5	ED	D0	EA	20	92	16	4C	45	15	C9	54	D0	06
1610:	20	D3	1E	4C	45	15	C9	50	D0	1C	A9	0F	85	F7	20	5C
1620:	1E	20	F8	1E	C6	F7	F0	EB	A0	00	B1	E6	C9	77	F0	E3
1630:	20	0B	17	4C	1E	16	C9	58	D0	13	A9	47	8D	7A	1A	A9
1640:	16	8D	7B	1A	4C	F8	14	20	BC	14	4C	88	17	20	B1	16
1650:	30	12	20	CE	16	30	0D	20	5C	1E	20	F8	1E	A5	FD	85
1660:	F6	4C	82	15	4C	8C	15	20	68	12	A0	00	20	3E	17	20
1670:	87	13	30	F3	AD	63	1A	85	E2	AD	64	1A	85	E3	AD	65
1680:	1A	85	E4	85	E8	AD	66	1A	85	E5	85	E9	20	D3	1E	4C
1690:	33	15	38	A5	E6	E5	F6	85	E6	A5	E7	E9	00	85	E7	60

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
16A0: 18 A5 EC 65 F6 85 EC A5 ED 69 00 85 ED 60 20 AE
16B0: 12 20 1E 14 30 12 0A 0A 0A 0A 85 FE 20 AE 12 20
16C0: 1E 14 30 04 05 FE A2 00 60 20 AE 16 30 27 85 FB
16D0: 20 60 1E 84 F7 84 FD C6 F7 F0 18 20 F3 11 20 AE
16E0: 16 30 12 85 FA C6 F7 F0 0A 20 F3 11 20 AE 16 30
16F0: 04 85 F9 A2 00 60 20 DC 1E 38 A5 E4 E9 02 E5 E8
1700: A5 E5 E5 E9 90 04 20 EA 1E 38 60 20 E8 11 20 5C
1710: 1E A6 F6 A5 E7 20 8F 12 A5 E6 20 8F 12 A9 0F 85
1720: EE A0 00 20 F3 11 B1 E6 20 8F 12 38 A5 EE E9 03
1730: 85 EE C8 CA D0 ED 20 F3 11 C6 EE D0 F9 60 20 E8
1740: 11 B9 50 17 C9 03 F0 07 20 34 13 C8 4C 41 17 60
1750: 42 45 47 41 44 2C 45 4E 44 41 44 3A 20 03 49 4C
1760: 4C 45 47 41 4C 20 4B 45 59 03 46 55 4C 4C 03 44
1770: 4F 4E 45 03 50 4D 20 45 44 49 54 4F 52 03 4C 41
1780: 42 20 24 03 3A 20 24 03 A0 FF 20 E8 11 A2 04 84
1790: FD A0 2E 20 41 17 A4 FD B1 EC 20 8F 12 88 84 FD
17A0: A0 34 20 41 17 A4 FD B1 EC 20 8F 12 88 B1 EC 20
17B0: 8F 12 20 F3 11 88 C4 EE F0 05 CA D0 D2 F0 CB 20
17C0: D3 1E 4C 3D 15 20 68 12 A0 00 20 3E 17 20 87 13
17D0: 30 F3 20 D3 1E A0 00 B1 E6 C9 77 F0 09 20 60 1E
17E0: 20 F8 1E 4C D5 17 18 A5 E6 69 01 85 E8 A5 E7 69
17F0: 00 85 E9 4C 33 15 D8 4C 00 1C D8 4C CF 14 77 77
```

De listing van het systeemprogramma TM en de listing van het systeemprogramma PM

Elke boekpagina bevat 69 assembler-regels. Na het algemene gedeelte (deklaraties) komen achtereenvolgens aan de orde:

- Het hoofdprogramma TM
- De subroutines van TM
- De "super-subroutine" DUMP/DUMPT
- De subroutines van DUMP/DUMPT
- De "super-subroutine" RDTAPE
- De subroutines van RDTAPE
- Het hoofdprogramma PM
- De subroutines van PM
- De opzoektabel MESS van PM
- De STEP-voorroutine van PM
- De subroutine IPBRES van PM
- En tot slot een "symbol table", met een overzicht van alle labels en niet-anonieme geheugenplaatsen.

```
0001: 0800                      ORG   $0800  SOFTWARE OF JUNIOR COMPUTER 3,4
0002:
0003:
0004:                   SOURCE LISTING OF THE TAPE/PRINTER MONITOR
0005:
0006:                   WRITTEN BY A. NACHTMANN
0007:
0008:
0009:                   DATE: 17 DEC. 1980
0010:
0011:
0012:
0013:                   ************************************
0014:                   *** DEFINITIONS OF DUMPT & RDTAPE ***
0015:                   ************************************
0016:
0017:
0018:                   DUMPT STORES A PROGRAM ON TAPE
0019:                   1) STORE A PROGRAM ON TAPE
0020:                   2) ID IS THE IDENTITY NUMBER
0021:                   3) SA IS THE START ADDRESS
0022:                   4) EA IS THE END ADDRESS + 1
0023:                   5) ID = 00 & ID = FF SHOULD NOT BE USED
0024:
0025:                   RDTAPE READS A PROGRAM FROM TAPE
0026:                   1) READ A PROGRAM FROM TAPE WITH A CERTAIN ID
0027:                   2) DISCARD DATA IN MEMORY BEGINNING AT SA
0028:                   3) ID = 00 & ID = FF ARE SPECIAL IDS
0029:
0030:
0031:
0032:                   THE DATA FORMAT ON TAPE IS ASCII:
0033:
0034:                   255SYN<>*<>ID<>SAL<>SAH<>DATA<>/<>CHKL<>CHKH<>EOT<>EOT
0035:
0036:                   I/O DEFINITION
0037:
0038: 0800              PAD    *      $1A80  PORT A DATA REGISTER
0039: 0800              PADD   *      $1A81  PORT A DATA DIRECTION
0040: 0800              PBD    *      $1A82  PORT B DATA REGISTER
0041: 0800              PBDD   *      $1A83  PORT B DATA DIRECTION
0042:
0043:
0044:                   TIMER REGISTERS
0045:
0046: 0800              CNTA   *      $1AF4  CLK1T, DISABLE TIMER IRQ
0047: 0800              CNTC   *      $1AF6  CLK64T, DISABLE TIMER IRQ
0048: 0800              RDFLAG *      $1AD5  READ FLAG REGISTER, B7 IS TIMER FLAG
0049: 0800              RDTDIS *      $1AD4  READ CONTENTS OF TIMER CELL, IRQ DISABLE
0050:
0051:
0052:                   TEMPORARY DATA BUFFERS
0053:
0054: 0800              SY     *      $1A69  SYN COUNTER
0055: 0800              BYTE   *      $1A6A  BYTE FROM TAPE
0056: 0800              CHAR   *      $1A6B  CHARACTER FROM TAPE
0057: 0800              HIGHER *      $1A6C  3600 HZ HALF PERIODE DELAY
0058: 0800              LOWER  *      $1A6D  2400 HZ HALF PERIODE DELAY
0059: 0800              CHKL   *      $1A6E  CHECK SUM LOW
0060: 0800              CHKH   *      $1A6F  CHECK SUM HIGH
0061: 0800              SAL    *      $1A70  START ADDRESS LOW
0062: 0800              SAH    *      $1A71  START ADDRESS HIGH
0063: 0800              EAL    *      $1A72  END ADDRESS LOW + 1
0064: 0800              EAH    *      $1A73  END ADDRESS HIGH
0065: 0800              SYNCNT *      $1A74
0066: 0800              BITS   *      $1A75  AMOUNT OF BITS
0067: 0800              FIRST  *      $1A76  HALF PERIODE AMOUNT OF 3600 HZ
0068: 0800              SECOND *      $1A77  HALF PERIODE AMOUNT OF 2400 HZ
0069: 0800              GANG   *      $1A78  TEMP OF PBD-BITS
```

```
0070: 0800          ID     *     $1A79  ID OF THE DUMPT PROGRAM
0071: 0800          NMI    *     $1A7A  NMI VECTOR
0072:
0073:               ****************************************
0074:               *** DEFINITIONS OF THE TTY MONITOR ***
0075:               ****************************************
0076:
0077:
0078:               I/O DEFINITION: SEE DUMPT & RDTAPE
0079:
0080:
0081:               CPU REGISTERS
0082:
0083: 0800          PCL    *     $00EF  PROGRAM COUNTER
0084: 0800          PCH    *     $00F0
0085: 0800          PREG   *     $00F1  STATUS REGISTER
0086: 0800          SPUSER *     $00F2  USER STACK POINTER
0087: 0800          ACC    *     $00F3  ACCUMULATOR
0088: 0800          YREG   *     $00F4  INDEX Y
0089: 0800          XREG   *     $00F5  INDEX X
0090:
0091:
0092:               INPUT BUFFERS
0093:
0094: 0800          INL    *     $00F8  LOWER BYTE
0095: 0800          INH    *     $00F9  UPPER BYTE
0096:
0097:
0098:               ADDRESS POINTER
0099:
0100: 0800          POINTL *     $00FA
0101: 0800          POINTH *     $00FB
0102:
0103:
0104:               TEMPORARAY DATA BUFFERS
0105:
0106: 0800          TEMP   *     $00FC
0107: 0800          TEMPX  *     $00FD
0108: 0800          STPBIT *     $1A59  NUMBER OF STOP BITS + 1
0109: 0800          CNTL   *     $1A5A  BIT TIME BUFFER
0110: 0800          CNTH   *     $1A5B
0111: 0800          CNTHL  *     $1A5C  HALF BIT TIME BUFFER
0112: 0800          CNTHH  *     $1A5D
0113: 0800          TIML   *     $1A5E  COUNT DOWN BUFFER
0114: 0800          TIMH   *     $1A5F
0115: 0800          TEMPA  *     $1A60  TEMPS
0116: 0800          TEMPB  *     $1A61
0117: 0800          CHA    *     $1A62  CHARACTER BUFFER
0118: 0800          PARAL  *     $1A63  PARAMETER BUFFERS
0119: 0800          PARAH  *     $1A64
0120: 0800          PARBL  *     $1A65
0121: 0800          PARBH  *     $1A66
0122: 0800          PRTEMP *     $1A67  TTY BUFFER
0123: 0800          BRKT   *     $1A7C  BREAK TEST VECTOR
0124:
0125:
0126:
0127:
0128:               ************************************
0129:               *** BUFFERS & EXTERNAL ADDRESSES ***
0130:               ************************************
0131:
0132: 0800          DISCNT *     $1A68  DISPLAY COUNTER
0133: 0800          COLDST *     $1CB5  EDITOR COLD START
0134: 0800          WARMST *     $1CCA  EDITOR WARM START
0135: 0800          BEGIN  *     $1ED3  EDITOR SUBROUTINE
0136: 0800          RESET  *     $1C1D  RESET OF VERSION D
0137: 0800          GETKEY *     $1DF9  COMPUTE THE KEY VALUE
0138: 0800          LDAINH *     $1DA7  PART OF SCAND/SCANDS
```

```
0139: 0800                BEGADL *      $00E2  BEGIN ADDRESS POINTER
0140: 0800                BEGADH *      $00E3
0141: 0800                ENDADL *      $00E4  END ADDRESS POINTER
0142: 0800                ENDADH *      $00E5
0143: 0800                CENDL  *      $00E8  CURRENT END ADDRESS POINTER
0144: 0800                CENDH  *      $00E9
0145:
0146:                     ************************
0147:                     *** TAPE MANAGEMENT ***
0148:                     ************************
0149:
0150:
0151:                     >PC KEY: READ A DATA BLOCK WITH ID=01...FE FROM TAPE
0152:                              1) IF ID = 00, NO SA MAY BE ENTERED
0153:                              2) IF ID = FF, SAH & SAL SHOULD BE ENTERED
0154:                     >AD KEY: SAVE A DATA BLOCK FROM SA TO EA-1 ON TAPE
0155:                              1) ENTER ID = 01...FE
0156:                              2) ENTER SAH & SAL
0157:                              3) ENTER EAH & EAL + 1
0158:                     >DA KEY: 1) EDITOR COLD START ENTRY
0159:                              2) FILE IS DEFINED BY BEGH,BEGL & ENDH,ENDL
0160:                     >+  KEY: DISPLAY ID-SAH-SAL-EAH-EAL-BEGH-BEGL-ENDH-
0161:                                        ENDL-ID-SAH...
0162:                     >GO KEY: SAVE AN EDITOR FILE BETWEEN BEGAD AND CEND
0163:                                ON TAPE; THEN JUMP TO WARM START ENTRY OF
0164:                                THE EDITOR AND DISPLAY THE FIRST INSTRUCTION
0165:
0166:
0167:
0168: 0800 20 72 0C  GETA   JSR   ADJPNT DECREMENT FILE POINTER
0169: 0803 A5 FA            LDAZ  POINTL POINT := SA
0170: 0805 8D 70 1A         STA   SAL
0171: 0808 A5 FB            LDAZ  POINTH
0172: 080A 8D 71 1A         STA   SAH
0173: 080D 4C 4E 08         JMP   GETB
0174:
0175: 0810 A2 00      TPINIT LDXIM $00
0176: 0812 8E 79 1A         STX   ID      RESET ALL TAPE PARAMETERS
0177: 0815 8E 70 1A         STX   SAL
0178: 0818 8E 71 1A         STX   SAH
0179: 081B 8E 72 1A         STX   EAL
0180: 081E 8E 73 1A         STX   EAH
0181:
0182: 0821 8E 68 1A  TPI    STX   DISCNT RESET DISPLAY COUNTER
0183: 0824 A9 06            LDAIM $06
0184: 0826 8D 82 1A         STA   PBD     DISPLAY OFF
0185: 0829 A9 1E            LDAIM $1E
0186: 082B 8D 83 1A         STA   PBDD
0187:
0188: 082E 20 36 09  TPTXT  JSR   TAPDIS DISPLAY TAPE PARAMETERS
0189: 0831 D0 FB            BNE   TPTXT   KEY RELEASED?
0190:
0191: 0833 20 36 09  TTXT   JSR   TAPDIS
0192: 0836 F0 FB            BEQ   TTXT    ANY KEY DEPRESSED?
0193: 0838 20 36 09         JSR   TAPDIS DEBOUNCE KEY
0194: 083B F0 F6            BEQ   TTXT    KEY STILL DEPRESSED?
0195: 083D 20 F9 1D         JSR   GETKEY RETURN WITH KEY IN ACCU
0196:
0197: 0840 C9 14      GET    CMPIM $14     PC KEY?
0198: 0842 D0 0E            BNE   SAVE
0199: 0844 20 02 0B         JSR   RDTAPE READ DATA FROM TAPE
0200: 0847 A2 FF            LDXIM $FF
0201: 0849 EC 79 1A         CPX   ID      CHECK, IF ID = FF
0202: 084C F0 B2            BEQ   GETA    IF YES, ADJUST FILE POINTER
0203:
0204: 084E E8         GETB   INX           RESET DISCNT
0205: 084F 4C 21 08         JMP   TPI     SHOW 'ID  XX'
0206:
0207: 0852 C9 10      SAVE   CMPIM $10     AD KEY?
```

```
0208: 0854 D0 08          BNE   PLUS
0209: 0856 20 DF 09       JSR   DUMP    WRITE DATA ON TAPE
0210: 0859 A2 00          LDXIM $00
0211: 085B 4C 21 08       JMP   TPI     SHOW 'ID  XX'
0212:
0213: 085E C9 12    PLUS  CMPIM $12     + KEY?
0214: 0860 D0 11          BNE   FILES
0215: 0862 EE 68 1A       INC   DISCNT  SET UP PARAMETER COUNTER
0216: 0865 A0 09          LDYIM $09     LIMIT COUNT
0217: 0867 CC 68 1A       CPY   DISCNT
0218: 086A D0 C2          BNE   TPTXT
0219: 086C A0 00          LDYIM $00     RESET PARAMETER COUNTER
0220: 086E 8C 68 1A       STY   DISCNT
0221: 0871 F0 BB          BEQ   TPTXT
0222:
0223: 0873 C9 13    FILES CMPIM $13     GO KEY?
0224: 0875 D0 25          BNE   DAT
0225: 0877 A5 E2          LDAZ  BEGADL
0226: 0879 8D 70 1A       STA   SAL     BEGAD := SA
0227: 087C A5 E3          LDAZ  BEGADH
0228: 087E 8D 71 1A       STA   SAH
0229: 0881 A5 E8          LDAZ  CENDL   CEND := EA
0230: 0883 8D 72 1A       STA   EAL
0231: 0886 A5 E9          LDAZ  CENDH
0232: 0888 8D 73 1A       STA   EAH
0233:
0234: 088B 20 DF 09       JSR   DUMP    WRITE DATA = FILE ON TAPE
0235: 088E 20 D3 1E       JSR   BEGIN   SHOW THE FIRST INSTRUKTION
0236: 0891 A9 1E          LDAIM $1E     DIFINE I/O
0237: 0893 8D 83 1A       STA   PBDD
0238: 0896 A2 FF          LDXIM $FF     RESET STACK
0239: 0898 9A             TXS
0240: 0899 4C CA 1C       JMP   WARMST  RETURN TO EDITOR
0241:
0242: 089C C9 11    DAT   CMPIM $11     DA KEY?
0243: 089E D0 03          BNE   SHIFT   IT WAS A DATA KEY
0244: 08A0 4C B5 1C       JMP   COLDST  EDITOR COLD START ENTRY
0245:
0246: 08A3 06 F9    SHIFT ASLZ  INH     SHIFT KEY INTO DISPLAY BUFFER
0247: 08A5 06 F9          ASLZ  INH
0248: 08A7 06 F9          ASLZ  INH
0249: 08A9 06 F9          ASLZ  INH
0250: 08AB 05 F9          ORAZ  INH
0251: 08AD 85 F9          STAZ  INH
0252: 08AF A0 00          LDYIM $00     RESET PARAMETER COUNTER
0253: 08B1 CC 68 1A       CPY   DISCNT  ID TO DISPLAY?
0254: 08B4 D0 06          BNE   STSAH
0255: 08B6 8D 79 1A       STA   ID      SAVE ID
0256: 08B9 4C 2E 08       JMP   TPTXT
0257:
0258: 08BC C8       STSAH INY
0259: 08BD CC 68 1A       CPY   DISCNT  SAH TO DISPLAY?
0260: 08C0 D0 06          BNE   STSAL
0261: 08C2 8D 71 1A       STA   SAH     SAVE SAH
0262: 08C5 4C 2E 08       JMP   TPTXT
0263:
0264: 08C8 C8       STSAL INY
0265: 08C9 CC 68 1A       CPY   DISCNT  SAL TO DISPLAY?
0266: 08CC D0 06          BNE   STEAH
0267: 08CE 8D 70 1A       STA   SAL     SAVE SAL
0268: 08D1 4C 2E 08       JMP   TPTXT
0269:
0270: 08D4 C8       STEAH INY
0271: 08D5 CC 68 1A       CPY   DISCNT  EAH TO DISPLAY?
0272: 08D8 D0 06          BNE   STEAL
0273: 08DA 8D 73 1A       STA   EAH     SAVE EAH
0274: 08DD 4C 2E 08       JMP   TPTXT
0275:
0276: 08E0 C8       STEAL INY
```

```
0277: 08E1 CC 68 1A          CPY   DISCNT EAL TO DISPLAY?
0278: 08E4 D0 06             BNE   STBEGH
0279: 08E6 8D 72 1A          STA   EAL    SAVE EAL
0280: 08E9 4C 2E 08          JMP   TPTXT
0281:
0282: 08EC C8          STBEGH INY
0283: 08ED CC 68 1A          CPY   DISCNT BEGH TO DISPLAY?
0284: 08F0 D0 05             BNE   STBEGL
0285: 08F2 85 E3             STAZ  BEGADH SAVE BEGADH
0286: 08F4 4C 2E 08          JMP   TPTXT
0287:
0288: 08F7 C8          STBEGL INY
0289: 08F8 CC 68 1A          CPY   DISCNT BEGADL TO DISPLAY?
0290: 08FB D0 05             BNE   STENDH
0291: 08FD 85 E2             STAZ  BEGADL SAVE BEGADL
0292: 08FF 4C 2E 08          JMP   TPTXT
0293:
0294: 0902 C8          STENDH INY
0295: 0903 CC 68 1A          CPY   DISCNT ENDADH TO DISPLAY?
0296: 0906 D0 05             BNE   STENDL
0297: 0908 85 E5             STAZ  ENDADH SAVE ENDADH
0298: 090A 4C 2E 08          JMP   TPTXT
0299:
0300: 090D C8          STENDL INY
0301: 090E CC 68 1A          CPY   DISCNT ENDADL TO DISPLAY?
0302: 0911 D0 05             BNE   TPVEC  ERROR EXIT
0303: 0913 85 E4             STAZ  ENDADL SAVE ENDADL
0304: 0915 4C 2E 08          JMP   TPTXT
0305:
0306:
0307: 0918 4C 10 08    TPVEC JMP   TPINIT
0308:
0309:                  SUBROUTINES OF 'TAPE MANAGEMENT'
0310:
0311: 091B B9 BB 09    TDISP LDAY  TLOOK  TEXT LOOKUP
0312: 091E 8D 80 1A          STA   PAD    OUTPUT TEXT PATTERN
0313: 0921 8E 82 1A          STX   PBD    DISPLAY ENABLE
0314: 0924 98                TYA         SAVE INDEX Y
0315: 0925 A0 7F             LDYIM $7F
0316:
0317: 0927 88          DELY  DEY         DELAY
0318: 0928 D0 FD             BNE   DELY
0319: 092A A8                TAY         RESTORE INDEX Y
0320: 092B E8                INX         SET UP FOR NEXT DISPLAY
0321: 092C E8                INX
0322: 092D C8                INY         ADJUST TEXT POINTER
0323: 092E E0 10             CPXIM $10   4 DISPLAYS SCANNED?
0324: 0930 D0 E9             BNE   TDISP
0325: 0932 20 A7 1D          JSR   LDAINH DISPLAY DATA BUFFER INH
0326: 0935 60                RTS
0327:
0328: 0936 A9 7F       TAPDIS LDAIM $7F
0329: 0938 8D 81 1A          STA   PADD   I/O DEFINITION
0330:
0331: 093B A2 08       SID   LDXIM $08    INIT DISPLAY SWITCH
0332: 093D A0 00             LDYIM $00
0333: 093F CC 68 1A          CPY   DISCNT ID TO DISPLAY?
0334: 0942 D0 09             BNE   SSAH
0335: 0944 AD 79 1A          LDA   ID     ID TO DISPLAY
0336: 0947 85 F9             STAZ  INH
0337:
0338: 0949 20 1B 09    COMPNT JSR  TDISP  SHOW ANY PARAMETER
0339: 094C 60                RTS
0340:
0341: 094D C8          SSAH  INY
0342: 094E CC 68 1A          CPY   DISCNT SAH TO DISPLAY?
0343: 0951 D0 09             BNE   SSAL
0344: 0953 AD 71 1A          LDA   SAH    SAH TO DISPLAY
0345: 0956 85 F9             STAZ  INH
```

```
0346: 0958 A0 04           LDYIM $04    LOOKUP POINTER
0347: 095A D0 ED           BNE   COMPNT SHOW SAH
0348:
0349: 095C C8        SSAL  INY
0350: 095D CC 68 1A        CPY   DISCNT SAL TO DISPLAY?
0351: 0960 D0 09           BNE   SEAH
0352: 0962 AD 70 1A        LDA   SAL    SAL TO DISPLAY
0353: 0965 85 F9           STAZ  INH
0354: 0967 A0 08           LDYIM $08    LOOKUP POINTER
0355: 0969 D0 DE           BNE   COMPNT SHOW SAL
0356:
0357: 096B C8        SEAH  INY
0358: 096C CC 68 1A        CPY   DISCNT EAH TO DISPLAY?
0359: 096F D0 09           BNE   SEAL
0360: 0971 AD 73 1A        LDA   EAH    EAH TO DISPLAY
0361: 0974 85 F9           STAZ  INH
0362: 0976 A0 0C           LDYIM $0C    LOOKUP POINTER
0363: 0978 D0 CF           BNE   COMPNT SHOW EAH
0364:
0365: 097A C8        SEAL  INY
0366: 097B CC 68 1A        CPY   DISCNT
0367: 097E D0 09           BNE   SBEGH
0368: 0980 AD 72 1A        LDA   EAL
0369: 0983 85 F9           STAZ  INH
0370: 0985 A0 10           LDYIM $10    LOOKUP POINTER
0371: 0987 D0 C0           BNE   COMPNT SHOW EAL
0372:
0373: 0989 C8        SBEGH INY
0374: 098A CC 68 1A        CPY   DISCNT
0375: 098D D0 08           BNE   SBEGL
0376: 098F A5 E3           LDAZ  BEGADH
0377: 0991 85 F9           STAZ  INH
0378: 0993 A0 14           LDYIM $14
0379: 0995 D0 B2           BNE   COMPNT
0380:
0381: 0997 C8        SBEGL INY
0382: 0998 CC 68 1A        CPY   DISCNT
0383: 099B D0 08           BNE   SENDH
0384: 099D A5 E2           LDAZ  BEGADL
0385: 099F 85 F9           STAZ  INH
0386: 09A1 A0 18           LDYIM $18
0387: 09A3 D0 A4           BNE   COMPNT
0388:
0389: 09A5 C8        SENDH INY
0390: 09A6 CC 68 1A        CPY   DISCNT
0391: 09A9 D0 08           BNE   SENDL
0392: 09AB A5 E5           LDAZ  ENDADH
0393: 09AD 85 F9           STAZ  INH
0394: 09AF A0 1C           LDYIM $1C
0395: 09B1 D0 96           BNE   COMPNT
0396:
0397: 09B3 A5 E4     SENDL LDAZ  ENDADL
0398: 09B5 85 F9           STAZ  INH
0399: 09B7 A0 20           LDYIM $20
0400: 09B9 D0 8E           BNE   COMPNT
0401:
0402:
0403:
0404:                 7 SEGMENT TEXT LOOKUP TABLE
0405:
0406: 09BB 66        TLOOK =     $66    ID PATTERN
0407: 09BC 21              =     $21
0408: 09BD 7F              =     $7F
0409: 09BE 7F              =     $7F
0410:
0411: 09BF 52              =     $52    SAH PATTERN
0412: 09C0 08              =     $08
0413: 09C1 09              =     $09
0414: 09C2 7F              =     $7F
```

```
0415:
0416: 09C3 52                =       $52     SAL PATTERN
0417: 09C4 08                =       $08
0418: 09C5 47                =       $47
0419: 09C6 7F                =       $7F
0420:
0421: 09C7 06                =       $06     EAH PATTERN
0422: 09C8 08                =       $08
0423: 09C9 09                =       $09
0424: 09CA 7F                =       $7F
0425:
0426: 09CB 06                =       $06     EAL PATTERN
0427: 09CC 08                =       $08
0428: 09CD 47                =       $47
0429: 09CE 7F                =       $7F
0430:
0431: 09CF 03                =       $03     BEGH PATTERN
0432: 09D0 06                =       $06
0433: 09D1 42                =       $42
0434: 09D2 09                =       $09
0435:
0436: 09D3 03                =       $03     BEGL PATTERN
0437: 09D4 06                =       $06
0438: 09D5 42                =       $42
0439: 09D6 47                =       $47
0440:
0441: 09D7 06                =       $06     ENDH PATTERN
0442: 09D8 2B                =       $2B
0443: 09D9 21                =       $21
0444: 09DA 09                =       $09
0445:
0446: 09DB 06                =       $06     ENDL PATTERN
0447: 09DC 2B                =       $2B
0448: 09DD 21                =       $21
0449: 09DE 47                =       $47
0450:                   ******************************
0451:                   * DUMP/DUMPT IS A SUBROUTINE *
0452:                   ******************************
0453:
0454: 09DF A9 7D      DUMP    LDAIM $7D     3600HZ HALF PERIODE DELAY
0455: 09E1 8D 6C 1A           STA   HIGHER
0456: 09E4 A9 C3              LDAIM $C3     2400 HZ HALF PERIODE DELAY
0457: 09E6 8D 6D 1A           STA   LOWER
0458: 09E9 A9 03              LDAIM $03     TOGGLE 3 TIMES AT 3600 HZ
0459: 09EB 8D 76 1A           STA   FIRST
0460: 09EE A9 02              LDAIM $02     TOGGLE 2 TIMES AT 2400 HZ
0461: 09F0 8D 77 1A           STA   SECOND
0462:
0463: 09F3 A9 47      DUMPT   LDAIM $47     PBD PATTERN
0464: 09F5 A2 FF              LDXIM $FF     PBDD PATTERN
0465: 09F7 8D 82 1A           STA   PBD     INPUT OFF,OUTPUT ON,STROBE DISABLED
0466: 09FA 8D 78 1A           STA   GANG    SAVE BITS OF PBD
0467: 09FD 8E 83 1A           STX   PBDD    PB7...PB0 IS OUTPUT
0468: 0A00 A9 00              LDAIM $00     PAD PATTERN
0469: 0A02 A2 7F              LDXIM $7F     PADD PATTERN
0470: 0A04 8D 80 1A           STA   PAD     SEGMENTS ON BUT DISABLED
0471: 0A07 8E 81 1A           STX   PADD    ALL LINES OUTPUT EXEPT PA7
0472: 0A0A 8D 6E 1A           STA   CHKL    RESET CHECK SUM
0473: 0A0D 8D 6F 1A           STA   CHKH
0474: 0A10 AD 70 1A           LDA   SAL     INITIALIZE DUMPT POINTER
0475: 0A13 85 FA              STAZ  POINTL
0476: 0A15 AD 71 1A           LDA   SAH
0477: 0A18 85 FB              STAZ  POINTH
0478: 0A1A A2 FF              LDXIM $FF     SET SYNC COUNTER
0479: 0A1C 8E 74 1A           STX   SYNCNT
0480:
0481: 0A1F A9 16      SYNCS   LDAIM '       SYN CHARACTER
0482: 0A21 20 A3 0A           JSR   OUTCH   OUTPUT 255 SYN CHARACTERS
0483: 0A24 CE 74 1A           DEC   SYNCNT
```

```
0484: 0A27 D0 F6            BNE   SYNCS
0485:
0486: 0A29 A9 2A            LDAIM '*      OUTPUT START CHARACTER
0487: 0A2B 20 A3 0A         JSR   OUTCH
0488: 0A2E AD 79 1A         LDA   ID      OUTPUT ID
0489: 0A31 20 8B 0A         JSR   OUTBT
0490: 0A34 AD 70 1A         LDA   SAL     OUTPUT START ADDRESS
0491: 0A37 20 7A 0A         JSR   OUTBTC  AND START CHECK SUM COMPUTATION
0492: 0A3A AD 71 1A         LDA   SAH
0493: 0A3D 20 7A 0A         JSR   OUTBTC
0494:
0495: 0A40 A5 FB      DATATR LDAZ  POINTH
0496: 0A42 CD 73 1A          CMP   EAH     ENTIRE FILE TRANSMITTED?
0497: 0A45 D0 23             BNE   HEXDAT
0498: 0A47 A5 FA             LDAZ  POINTL
0499: 0A49 CD 72 1A          CMP   EAL
0500: 0A4C D0 1C             BNE   HEXDAT
0501:
0502: 0A4E A9 2F             LDAIM '/      OUTPUT END OF DATA CHARACTER
0503: 0A50 20 A3 0A          JSR   OUTCH   STOP WITH CHECK SUM COMPUTATION
0504: 0A53 AD 6E 1A          LDA   CHKL    OUTPUT CHECK SUM
0505: 0A56 20 8B 0A          JSR   OUTBT
0506: 0A59 AD 6F 1A          LDA   CHKH
0507: 0A5C 20 8B 0A          JSR   OUTBT
0508: 0A5F A9 04             LDAIM '       EOT CHARACTER
0509: 0A61 20 A3 0A          JSR   OUTCH   OUTPUT EOT CHARACTER
0510: 0A64 A9 04             LDAIM '       EOT CHARACTER
0511: 0A66 20 A3 0A          JSR   OUTCH
0512:
0513: 0A69 60                RTS           RETURN TO CALLER
0514:
0515: 0A6A A0 00      HEXDAT LDYIM $00
0516: 0A6C B1 FA             LDAIY POINTL  FETCH CURRENT DATA BYTE
0517: 0A6E 20 7A 0A          JSR   OUTBTC  TRANSMIT CURRENT DATA BYTE
0518: 0A71 E6 FA             INCZ  POINTL  AND COMPUT CHECK SUM
0519: 0A73 D0 CB             BNE   DATATR  SET UP FOR NEXT DATA BYTE
0520: 0A75 E6 FB             INCZ  POINTH
0521: 0A77 4C 40 0A          JMP   DATATR
0522:
0523:                        *** END OF DUMP/DUMPT ***
0524:                        DUMP'S SUBROUTINES
0525:
0526:
0527:                        OUTBTC OUTPUTS A HEX BYTE AS TWO ASCII CHARACTERS
0528:                        TO THE TAPE RECORDER. ALSO THE CHECK SUM IS COM-
0529:                        PUTED.
0530:
0531:                        OUTBT OUTPUTS A HEX BYTE AS TWO ASCCI CHARACTERS
0532:                        TO THE TAPE RECORDER WITHOUT CHECK SUM COMPUTATION.
0533:
0534:                        NIBOUT CONVERTS A HEX NIBBLE TO AN ASCII CHARACTER
0535:                        AND TRANSMITS THE 8 BIT ASCII WORD TO THE TAPE.
0536:
0537:                        HIGH AND LOW PRODUCE THE DELAYS OF THE 3600 HZ AND
0538:                        THE 2400 HZ FREQUENCY.
0539:
0540:
0541:                        *** OUTBTC/OUTBT ***
0542:
0543: 0A7A A8           OUTBTC TAY           SAVE ACCU
0544: 0A7B 18                  CLC
0545: 0A7C 6D 6E 1A            ADC   CHKL    CHECK SUM COMPUTATION
0546: 0A7F 8D 6E 1A            STA   CHKL
0547: 0A82 AD 6F 1A            LDA   CHKH
0548: 0A85 69 00               ADCIM $00     CHK:= CHK + BYTE
0549: 0A87 8D 6F 1A            STA   CHKH
0550: 0A8A 98                  TYA           GET ACCU AGAIN
0551:
0552: 0A8B A8           OUTBT  TAY           SAVE ACCU TEMP
```

```
0553: 0A8C 4A              LSRA          GET UPPER NIBBLE
0554: 0A8D 4A              LSRA
0555: 0A8E 4A              LSRA
0556: 0A8F 4A              LSRA
0557: 0A90 20 9A 0A        JSR   NIBOUT OUTPUT UPPER NIBBLE AS ASCII CHAR.
0558: 0A93 98              TYA           GET BYTE AGAIN
0559: 0A94 29 0F           ANDIM $0F     GET LOWER NIBBLE
0560: 0A96 20 9A 0A        JSR   NIBOUT OUTPUT LOWER NIBBLE AS ASCII CHAR.
0561: 0A99 60              RTS
0562:
0563:                 *** END OF OUTBTC/OUTBT ***
0564:
0565:
0566:                 *** NIBOUT/OUTCH ***
0567:
0568: 0A9A C9 0A     NIBOUT CMPIM $0A     CONVERT A NIBBLE TO AN ASCII CHAR.
0569: 0A9C 18               CLC
0570: 0A9D 30 02           BMI   NIB
0571: 0A9F 69 07           ADCIM $07
0572:
0573: 0AA1 69 30     NIB   ADCIM $30
0574:
0575: 0AA3 A2 08     OUTCH LDXIM $08     SET UP FOR 8 BITS
0576: 0AA5 8E 75 1A        STX   BITS
0577:
0578: 0AA8 4A        ONE   LSRA          SHIFT OUT BIT BY BIT
0579: 0AA9 48               PHA           SAVE CHARACTER
0580: 0AAA 90 0C            BCC   ZERO
0581: 0AAC 20 C8 0A         JSR   HIGH    START AT 3600 HZ
0582: 0AAF 20 E5 0A         JSR   LOW
0583: 0AB2 20 E5 0A         JSR   LOW     END AT 2400 HZ
0584: 0AB5 4C C1 0A         JMP   ZRO
0585:
0586: 0AB8 20 C8 0A  ZERO   JSR   HIGH    START AT 3600 HZ
0587: 0ABB 20 C8 0A         JSR   HIGH
0588: 0ABE 20 E5 0A         JSR   LOW     END AT 2400 HZ
0589:
0590: 0AC1 68        ZRO    PLA           GET CHARACTER AGAIN
0591: 0AC2 CE 75 1A         DEC   BITS    ALL BITS SHIFTED OUT
0592: 0AC5 D0 E1            BNE   ONE
0593: 0AC7 60               RTS
0594:
0595:                 *** END OF NIBOUT/OUTCH ***
0596:
0597:
0598:                 *** HIGH ***
0599:
0600: 0AC8 AE 76 1A  HIGH   LDX   FIRST   AMOUNT OF HALF PERIODES
0601:
0602: 0ACB 2C D5 1A  HIG    BIT   RDFLAG  TIME OUT?
0603: 0ACE 10 FB            BPL   HIG
0604: 0AD0 AD 6C 1A         LDA   HIGHER  SET UP TIMER FOR SHORT PERIODE
0605: 0AD3 8D F4 1A         STA   CNTA    DISABLE TIMER IRQ, CLK1T
0606: 0AD6 AD 78 1A         LDA   GANG
0607: 0AD9 49 80            EORIM $80     TOGGLE OUTPUT
0608: 0ADB 8D 82 1A         STA   PBD
0609: 0ADE 8D 78 1A         STA   GANG    SAVE STATUS QUO
0610: 0AE1 CA               DEX
0611: 0AE2 D0 E7            BNE   HIG
0612: 0AE4 60               RTS
0613:
0614:                 *** END OF HIGH ***
0615:
0616:
0617:                 *** LOW ***
0618:
0619: 0AE5 AE 77 1A  LOW    LDX   SECOND  AMOUNT OF HALF PERIODES
0620:
0621: 0AE8 2C D5 1A  LO     BIT   RDFLAG  TIME OUT?
```

```
0622: 0AEB 10 FB            BPL   LO
0623: 0AED AD 6D 1A         LDA   LOWER  SET UP TIMER FOR LONG PERIODE
0624: 0AF0 8D F4 1A         STA   CNTA   DISABLE TIMER IRQ, CLK1T
0625: 0AF3 AD 78 1A         LDA   GANG
0626: 0AF6 49 80            EORIM $80    TOGGLE OUTPUT
0627: 0AF8 8D 82 1A         STA   PBD
0628: 0AFB 8D 78 1A         STA   GANG   SAVE STATUS QUO
0629: 0AFE CA               DEX
0630: 0AFF D0 E7            BNE   LO
0631: 0B01 60               RTS
0632:
0633:                     *** END OF LOW ***
0634:
0635:                     ************************
0636:                     * RDTAPE IS A SUBROUTINE *
0637:                     ************************
0638:
0639:                     ID = 00: IGNORE ID ON TAPE; DISCARD DATA BLOCK
0640:                     AT SAL, SAH COMING FROM TAPE
0641:
0642:                     ID = FF: IGNORE ID ON TAPE; DISCARD DATA BLOCK
0643:                     AT SAL, SAH STORED IN JUNIORS MEMORY
0644:
0645: 0B02 A9 32      RDTAPE LDAIM $32    INPUT RECORDER IS ON
0646: 0B04 8D 82 1A         STA   PBD    OUTPUT RECORDER IS OFF, PB7 IS ENABLED
0647: 0B07 8D 78 1A         STA   GANG   STROBE '9' IS ENABLED
0648: 0B0A A9 7E            LDAIM $7E    PB0 IS INPUT
0649: 0B0C 8D 83 1A         STA   PBDD   PB7 IS INPUT
0650: 0B0F A9 7F            LDAIM $7F    PA6...PA0 IS OUTPUT
0651: 0B11 8D 81 1A         STA   PADD
0652: 0B14 A9 00            LDAIM $00    SEGMENTS ON
0653: 0B16 8D 6E 1A         STA   CHKL   RESET CHECK SUM
0654: 0B19 8D 6F 1A         STA   CHKH
0655:
0656: 0B1C A9 FF      SYNC  LDAIM $FF    RESET FOR SYN CHARACTER
0657: 0B1E 8D 6B 1A         STA   CHAR
0658:
0659: 0B21 20 C2 0B  SYNCA  JSR   RDBIT  READ A BIT FROM TAPE
0660: 0B24 6E 6B 1A         ROR   CHAR   RIGHT SHIFT
0661: 0B27 AD 6B 1A         LDA   CHAR   GET CURRENT CHARACTER
0662: 0B2A 20 E8 0B         JSR   BTWEEN DISPLAY NOT SYN CHARACTER
0663: 0B2D C9 16            CMPIM '      SYN CHARACTER?
0664: 0B2F D0 F0            BNE   SYNCA  IF NOT, RESYNC
0665: 0B31 A0 0A            LDYIM $0A    TRY IT FOR 10 SYNS AT LEAST
0666: 0B33 8C 69 1A         STY   SY     SYN COUNTER
0667:
0668: 0B36 20 36 0C  TENSYN JSR   RDCH
0669: 0B39 20 5D 0C         JSR   CHARVU DISPLAY SYN CHARACTER
0670: 0B3C C9 16            CMPIM '      STILL SYN CHARACTER?
0671: 0B3E D0 DC            BNE   SYNC   IF NOT, RETURN
0672: 0B40 CE 69 1A         DEC   SY     10 SYNS RECEIVED?
0729: 0BBA AD 71 1A         LDA   SAH
0730: 0BBD 85 FB            STAZ  POINTH
0731: 0BBF 4C 70 0B         JMP   FILMEM
0732:
0733:                     *** END OF RDTAPE ***
0734:
0735:                     SUBROUTINES OF RDTAPE
0736:
0737:                     *** RDBIT ***
0738:
0739:                     RDBIT READS 1 BIT FROM TAPE.
0740:                     LOG 1: IT RETURNS WITH C = 1
0741:                     LOG 0: IT RETUNS WITH C = 0
0742:
0743: 0BC2 2C 82 1A  RDBIT  BIT   PBD    3600 HZ?
0744: 0BC5 10 FB            BPL   RDBIT
0745: 0BC7 AD D4 1A         LDA   RDTDIS GET COUNT DOWN
0746: 0BCA A0 FF            LDYIM $FF
```

```
0747: 0BCC 8C F6 1A          STY   CNTC    START TIMER FOR 2400 HZ COUNT DOWN
0748: 0BCF A0 14             LDYIM $14
0749:
0750: 0BD1 88          RDBA  DEY           DELAY JITTER TIME
0751: 0BD2 D0 FD             BNE   RDBA
0752:
0753: 0BD4 2C 82 1A    RDBB  BIT   PBD     2400 HZ?
0754: 0BD7 30 FB             BMI   RDBB
0755: 0BD9 38                SEC
0756: 0BDA ED D4 1A          SBC   RDTDIS SET OR RESET C-FLAG
0757: 0BDD A0 FF             LDYIM $FF
0758: 0BDF 8C F6 1A          STY   CNTC    START TIMER FOR 3600 HZ COUNT DOWN
0759: 0BE2 A0 07             LDYIM $07     DELAY FOR JITTER
0760:
0761: 0BE4 88          RDBC  DEY
0762: 0BE5 D0 FD             BNE   RDBC
0763: 0BE7 60                RTS
0764:
0765:                  *** END OF RDBIT ***
0766:
0767:
0768:
0769:                  *** BTWEEN ***
0770:
0771:                  DISPLAY THE BETWEEN CHARACTER
0772:
0773: 0BE8 48          BTWEEN PHA          SAVE ACCU
0774: 0BE9 A9 36             LDAIM $36     OUTPUT BETWEEN CHARACTER
0775: 0BEB 8D 80 1A          STA   PAD
0776: 0BEE 68                PLA           GET ACCU AGAIN
0777: 0BEF 20 64 0C          JSR   VU      DON'T CHANGE BETWEEN
0778: 0BF2 60                RTS
0779:
0780:                  *** END OF BTWEEN ***
0781:                  *** RDBYT ***
0782:
0783:
0784:                  READ 1 HEX BYTE = 2 ASCII CHARACTERS FROM TAPE.
0785:                  COMPOSE THE HEX VALUES OF THE CHARACTERS IN ACCU
0786:                  AND RETURN.
0787:
0788:                  1) N = 1: A NOT VALID HEX CHARACTER WAS TRANSMITTED
0789:                  2) Z = 1: END OF DATA TRANSMISSION
0790:                  3) Z = 0: VALID HEX BYTE IN ACCU
0791:                  4) THE N-FLAG HAS ALWAYS PRIORITY
0792:
0793: 0BF3 20 36 0C    RDBYT JSR   RDCH    READ ANY ASCII CHARACTER FROM TAPE
0794: 0BF6 C9 2F             CMPIM '/      END OF DATA CHARACTER?
0795: 0BF8 D0 01             BNE   RBB
0796:
0797: 0BFA 60          RBA   RTS           ERROR EXIT
0798:
0799: 0BFB 20 19 0C    RBB   JSR   ASCHEX ASCII HEX CONVERSION
0800: 0BFE 30 FA             BMI   RBA     NOT VALID CHARACTER
0801: 0C00 0A                ASLA          SHIFT NIBBLE TO LEFT
0802: 0C01 0A                ASLA
0803: 0C02 0A                ASLA
0804: 0C03 0A                ASLA
0805: 0C04 8D 6A 1A          STA   BYTE    SAVE HIGH ORDER NIBBLE
0806: 0C07 20 36 0C          JSR   RDCH    READ NEXT CHARACTER
0807: 0C0A C9 2F             CMPIM '/      END OF DATA CHARACTER
0808: 0C0C F0 EC             BEQ   RBA
0809: 0C0E 20 19 0C          JSR   ASCHEX ASCII HEX CONVERSION
0810: 0C11 30 E7             BMI   RBA     NOT VALID CHARACTER
0811: 0C13 0D 6A 1A          ORA   BYTE    BYTE = HIGH ORDER AND LOW ORDER NIBBLE
0812: 0C16 A0 01             LDYIM $01     BE SHURE THAT CHARACTER
0813: 0C18 60                RTS           NORMAL EXIT
0814:
0815:                  *** END OF RDBYT ***
```

```
0816:                   *** ASCHEX ***
0817:
0818:
0819:                   CONVERT AN ASCII CHARACTER TO A HEX DATA NIBBLE.
0820:
0821:                   1) RETURN WITH CONVERTED HEX NUMBER IN ACCU
0822:                   2) N = 1, IF NOT VALID HEX NUMBER
0823:                   3) Z = 1, IF VALID HEX NUMBER
0824:                   4) *** ASCHEX IS ALSO USED IN THE PRINTER SOFTWARE ***
0825:
0826: 0C19 C9 30        ASCHEX CMPIM $30     IGNORE 00...2F
0827: 0C1B 30 0C               BMI   NOTVAL
0828: 0C1D C9 3A               CMPIM $3A
0829: 0C1F 30 0B               BMI   VALID
0830: 0C21 C9 41               CMPIM $41     IGNORE 3A...40
0831: 0C23 30 04               BMI   NOTVAL
0832: 0C25 C9 47               CMPIM $47     IGNORE 47...7F
0833: 0C27 30 03               BMI   VALID
0834:
0835: 0C29 A0 FF        NOTVAL LDYIM $FF     SET N-FLAG
0836: 0C2B 60                  RTS           ERROR EXIT
0837:
0838: 0C2C C9 40        VALID  CMPIM $40     ASCII HEX CONVERSION
0839: 0C2E 30 03               BMI   VAL
0840: 0C30 18                  CLC
0841: 0C31 69 09               ADCIM $09
0842:
0843: 0C33 29 0F        VAL    ANDIM $0F     HEX DATA IS LOW ORDER NIBBLE IN ACCU
0844: 0C35 60                  RTS
0845:
0846:                   *** END OF ASCHEX ***
0847:                   *** RDCH ***
0848:
0849:
0850:                   READ AN ASCII CHARACTER FROM TAPE AND
0851:                   STORE IT IN ACCU
0852:
0853: 0C36 A2 08        RDCH   LDXIM $08     SET UP FOR 8 BITS
0854:
0855: 0C38 20 C2 0B     READ   JSR   RDBIT   READ A BIT FROM TAPE
0856: 0C3B 6E 6B 1A            ROR   CHAR    SHIFT BIT INTO CHARACTER
0857: 0C3E CA                  DEX           ALL BITS READ?
0858: 0C3F D0 F7               BNE   READ
0859: 0C41 2E 6B 1A            ROL   CHAR    B7 MUST BE ZERO
0860: 0C44 4E 6B 1A            LSR   CHAR
0861: 0C47 AD 6B 1A            LDA   CHAR    RECEIVED CHARACTER TO ACCU
0862: 0C4A 60                  RTS
0863:
0864:                   *** END OF RDCH ***
0865:
0866:                   *** CHKSUM ***
0867:
0868:
0869:                   COMPUTE CHECK SUM OF RECEIVED DATA
0870:
0871: 0C4B 48           CHKSUM PHA           SAVE ACCU
0872: 0C4C 18                  CLC
0873: 0C4D 6D 6E 1A            ADC   CHKL    CHK := CHK + BYTE
0874: 0C50 8D 6E 1A            STA   CHKL
0875: 0C53 AD 6F 1A            LDA   CHKH
0876: 0C56 69 00               ADCIM $00
0877: 0C58 8D 6F 1A            STA   CHKH
0878: 0C5B 68                  PLA           GET ACCU AGAIN
0879: 0C5C 60                  RTS
0880:
0881:                   *** END OF CHKSUM ***
0882:
0883:
0884:                   *** CHARVU ***
```

```
0885:
0886:
0887:                   OUTPUT ANY CHARACTER TO 7 SEGMENT DISPLAY
0888:                   CHARACTER CHANGE IS ENABLED/DISABLED
0889:
0890: 0C5D 48            CHARVU PHA          SAVE ACCU
0891: 0C5E 49 7F               EORIM $7F     OUTPUT INVERTED CHAR. TO SEGMENTS
0892: 0C60 8D 80 1A            STA   PAD
0893: 0C63 68                  PLA           RESTORE ACCU
0894:
0895: 0C64 48            VU    PHA
0896: 0C65 AD 78 1A            LDA   GANG
0897: 0C68 49 02               EORIM $02     CHANGE DISPLAYS
0898: 0C6A 8D 82 1A            STA   PBD
0899: 0C6D 8D 78 1A            STA   GANG    SAVE STATUS QUO
0900: 0C70 68                  PLA
0901: 0C71 60                  RTS
0902:
0903:                   *** END OF CHARVU ***
0904:
0905:
0906:                   ADJPNT ADDRESS POINTER ADJUSTMENT
0907: 0C72 38           ADJPNT SEC          16 BIT SUBTRACTION
0908: 0C73 A5 FA               LDAZ  POINTL
0909: 0C75 E9 01               SBCIM $01
0910: 0C77 85 FA               STAZ  POINTL
0911: 0C79 A5 FB               LDAZ  POINTH
0912: 0C7B E9 00               SBCIM $00
0913: 0C7D 85 FB               STAZ  POINTH
0914: 0C7F 60                  RTS
0915:
0916: 0D                *** END OF ADJPNT ***
ID=B4 0E
R     0F
X

0001: 1000                     ORG   $1000
0002:
0003:
0004:                   ****************************************
0005:                   *** MAIN PROGRAM OF THE PRINTER MONITOR ***
0006:                   ****************************************
0007:
0008:
0009:                   COMMANDS OF THE PRINTER MONITOR:
0010:
0011:                   > '-'  DECREMENT THE CURRENT ADDRESS BY ONE
0012:                   > '+'  INCREMENT THE CURRENT ADDRESS BY ONE
0013:                   > 'SPACE'  1) PRINT THE ADDRESS IN THE INPUT BUFFER
0014:                              2) SHOW THE DATA OF THIS ADDRESS
0015:                   > '.'  STORE INPUT DATA AT THE CURRENT ADDRESS
0016:                   > 'R'   START PROGRAM EXECUTION AT THE CURRENT ADDRESS
0017:                   > 'L'  LIST THE CONTENTS OF ALL CPU REGISTERS
0018:                   > 'P'  PRINT OUT THE LAST PROGRAM CONTER
0019:                   > 'M'  PRINT A HEXDUMP SPECIFIED BY THE INPUT PARAMETER
0020:                   > 'G' READ A DATA BLOCK WITH A CERTAIN ID FROM TAPE
0021:                   > 'S' STORE A DATA BLOCK BETWEEN SA AND EA-1 ON TAPE
0022:
0023:
0024:
0025: 1000 D8           INITPR CLD          RESET SEQUENCE OF THE PRINTER PROGRAM
0026: 1001 78                  SEI           IRQ LINE IS DISABLED
0027: 1002 A9 67               LDAIM $67
0028: 1004 8D 82 1A            STA   PBD     PBD: 01100111
0029: 1007 A9 00               LDAIM $00
0030: 1009 8D 80 1A            STA   PAD     PAD: 00000000
0031: 100C A2 FE               LDXIM $FE     RESET BIT TIME COUNTER
0032: 100E 8E 5A 1A            STX   CNTL
0033: 1011 E8                  INX
```

```
0034: 1012 8E 5B 1A          STX   CNTH
0035: 1015 9A                TXS         RESET COMPUTER STACK POINTER
0036: 1016 86 F2             STXZ  SPUSER RESET USER STACK POINTER
0037: 1018 A9 7F             LDAIM $7F
0038: 101A 8D 81 1A          STA   PADD   PADD: 01111111
0039: 101D 8D 83 1A          STA   PBDD   PBDD: 01111111
0040: 1020 A2 02             LDXIM $02
0041: 1022 8E 59 1A          STX   STPBIT TRANSMIT NONE PARITY & ONE STOP BIT
0042: 1025 A9 5F             LDAIM LABJUN
0043: 1027 8D 7C 1A          STA   BRKT   SETUP BREAK VECTOR
0044: 102A A9 10             LDAIM LABJUN /256
0045: 102C 8D 7D 1A          STA   BRKT   +01
0046: 102F A9 CF             LDAIM STEP   SETUP STEP BY STEP VECTOR
0047: 1031 8D 7A 1A          STA   NMI
0048: 1034 A9 14             LDAIM STEP   /256
0049: 1036 8D 7B 1A          STA   NMI    +01
0050:
0051: 1039 2C 80 1A  STRTBT  BIT   PAD    WAIT FOR A START BIT
0052: 103C 30 FB             BMI   STRTBT
0053: 103E 20 E0 12          JSR   COMTIM COMPUTE THE START BIT TIME
0054: 1041 4E 5F 1A          LSR   TIMH   DIVIDE BY 2
0055: 1044 6E 5E 1A          ROR   TIML
0056: 1047 AD 5E 1A          LDA   TIML
0057: 104A 8D 5C 1A          STA   CNTHL  SAVE HALF START BIT TIME
0058: 104D AD 5F 1A          LDA   TIMH
0059: 1050 8D 5D 1A          STA   CNTHH
0060: 1053 A2 08             LDXIM $08
0061: 1055 20 03 13          JSR   DELHBT
0062: 1058 20 BE 12          JSR   RECD   GET THE REST OF THE CHARACTER
0063: 105B C9 7F             CMPIM $7F    WAS IT THE RUBOUT CHARACTER?
0064: 105D D0 A1             BNE   INITPR
0065:
0066: 105F 20 46 12  LABJUN  JSR   JUNIOR PRINT 'JUNIOR'
0067: 1062 20 E8 11          JSR   CRLF
0068: 1065 A2 FF             LDXIM $FF
0069: 1067 9A                TXS         RESET STACK POINTER WHEN A BREAK OCCURS
0070: 1068 86 F2             STXZ  SPUSER
0071:
0072: 106A 20 59 12  RESALL  JSR   RESPAR RESET PARAMETER BUFFER
0073: 106D 20 68 12          JSR   RESIN  RESET INPUT BUFFER
0074:
0075: 1070 20 AE 12  READCH  JSR   RECCHA WAIT FOR A CHARACTER
0076:
0077: 1073 C9 2B     PLU     CMPIM '+
0078: 1075 D0 09             BNE   MINUS
0079: 1077 20 13 12          JSR   INCPNT INCREMENT CURRENT ADDRESS
0080: 107A 20 F8 11          JSR   PRBUFS OPEN NEXT CELL
0081: 107D 4C 6A 10          JMP   RESALL
0082:
0083: 1080 C9 2D     MINUS   CMPIM '-
0084: 1082 D0 09             BNE   SPACE
0085: 1084 20 1A 12          JSR   DECPNT DECREMENT CURRENT ADDRESS
0086: 1087 20 F8 11          JSR   PRBUFS OPEN PREVIOUS CELL
0087: 108A 4C 6A 10          JMP   RESALL
0088:
0089: 108D C9 20     SPACE   CMPIM '
0090: 108F D0 0E             BNE   PNT
0091: 1091 A5 F8             LDAZ  INL    OUTPUT THE ADDRESS
0092: 1093 85 FA             STAZ  POINTL IN THE INPUT BUFFER
0093: 1095 A5 F9             LDAZ  INH
0094: 1097 85 FB             STAZ  POINTH
0095: 1099 20 F8 11          JSR   PRBUFS OPEN CURRENT CELL
0096: 109C 4C 6A 10          JMP   RESALL
0097:
0098: 109F C9 2E     PNT     CMPIM '.
0099: 10A1 D0 0F             BNE   RUN
0100: 10A3 A5 F8             LDAZ  INL
0101: 10A5 A0 00             LDYIM $00
0102: 10A7 91 FA             STAIY POINTL STORE CURRENT DATA BYTE IN MEMORY
```

```
0103: 10A9 20 13 12          JSR   INCPNT
0104: 10AC 20 F8 11          JSR   PRBUFS OPEN NEXT CELL
0105: 10AF 4C 6A 10          JMP   RESALL
0106:
0107: 10B2 C9 52      RUN    CMPIM 'R
0108: 10B4 D0 13             BNE   LIST
0109: 10B6 A6 F2             LDXZ  SPUSER START PROGRAM EXECUTION
0110: 10B8 9A                TXS         AT THE CURRENT DISPLAYED ADDRESS
0111: 10B9 A5 FB             LDAZ  POINTH
0112: 10BB 48                PHA
0113: 10BC A5 FA             LDAZ  POINTL
0114: 10BE 48                PHA
0115: 10BF A5 F1             LDAZ  PREG
0116: 10C1 48                PHA
0117: 10C2 A6 F5             LDXZ  XREG
0118: 10C4 A4 F4             LDYZ  YREG
0119: 10C6 A5 F3             LDAZ  ACC
0120: 10C8 40                RTI
0121:
0122: 10C9 C9 4C      LIST   CMPIM 'L
0123: 10CB D0 52             BNE   PC
0124: 10CD A0 14             LDYIM $14
0125: 10CF 20 D6 11          JSR   MESSY  ACC:
0126: 10D2 A5 F3             LDAZ  ACC
0127: 10D4 20 8F 12          JSR   PRBYT
0128: 10D7 A0 1A             LDYIM $1A
0129: 10D9 20 D6 11          JSR   MESSY  Y  :
0130: 10DC A5 F4             LDAZ  YREG
0131: 10DE 20 8F 12          JSR   PRBYT
0132: 10E1 A0 20             LDYIM $20
0133: 10E3 20 D6 11          JSR   MESSY  X  :
0134: 10E6 A5 F5             LDAZ  XREG
0135: 10E8 20 8F 12          JSR   PRBYT
0136: 10EB A0 26             LDYIM $26
0137: 10ED 20 D6 11          JSR   MESSY  PC :
0138: 10F0 A5 F0             LDAZ  PCH
0139: 10F2 20 8F 12          JSR   PRBYT
0140: 10F5 A5 EF             LDAZ  PCL
0141: 10F7 20 8F 12          JSR   PRBYT
0142: 10FA A0 2C             LDYIM $2C
0143: 10FC 20 D6 11          JSR   MESSY  SP :
0144: 10FF A9 01             LDAIM $01
0145: 1101 20 8F 12          JSR   PRBYT
0146: 1104 A5 F2             LDAZ  SPUSER
0147: 1106 20 8F 12          JSR   PRBYT
0148: 1109 A0 32             LDYIM $32
0149: 110B 20 D6 11          JSR   MESSY  PR :
0150: 110E 20 28 12          JSR   SHOWPR PRINT OUT FLAGS
0151: 1111 A0 38             LDYIM $38
0152: 1113 20 D6 11          JSR   MESSY  NV BDIZC
0153: 1116 20 F3 11          JSR   PRSP
0154: 1119 4C 6A 10          JMP   RESALL
0155:
0156: 111C 4C B0 11  CONTIN  JMP   GETTAP
0157:
0158:
0159: 111F C9 50      PC     CMPIM 'P
0160: 1121 D0 0E             BNE   MATRIX
0161: 1123 A5 EF             LDAZ  PCL    RESTORE LAST PROGRAM COUNTER
0162: 1125 85 FA             STAZ  POINTL
0163: 1127 A5 F0             LDAZ  PCH
0164: 1129 85 FB             STAZ  POINTH
0165: 112B 20 F8 11          JSR   PRBUFS OPEN CURRENT CELL
0166: 112E 4C 6A 10          JMP   RESALL
0167:
0168: 1131 C9 4D      MATRIX CMPIM 'M
0169: 1133 D0 E7             BNE   CONTIN
0170: 1135 A0 52             LDYIM $52
0171: 1137 20 D6 11          JSR   MESSY  HEXDUMP:
```

```
172: 113A 20 87 13          JSR   INPAR  READ PARAMETERS
173: 113D 10 03             BPL   MATF
174:
175: 113F 4C 5F 10  MATD    JMP   LABJUN RETURN, IF INVALID CHARACTER
176:
177: 1142 38        MATF    SEC          VALID INPUT PARAMETERS?
178: 1143 AD 65 1A          LDA   PARBL
179: 1146 ED 63 1A          SBC   PARAL  PARA < PARB?
180: 1149 AD 66 1A          LDA   PARBH
181: 114C ED 64 1A          SBC   PARAH
182: 114F 90 EE             BCC   MATD
183: 1151 20 E8 11          JSR   CRLF
184: 1154 A2 06             LDXIM $06
185:
186: 1156 20 F3 11  MATG    JSR   PRSP
187: 1159 CA                DEX
188: 115A D0 FA             BNE   MATG
189: 115C A0 00             LDYIM $00    RESET COLUMN COUNTER
190:
191: 115E 98        MATH    TYA
192: 115F 20 9B 12          JSR   PRNIBL OUTPUT COLUMNS
193: 1162 20 F3 11          JSR   PRSP
194: 1165 20 F3 11          JSR   PRSP
195: 1168 C8                INY
196: 1169 C0 10             CPYIM $10    PRINT COLUMS 0...F
197: 116B D0 F1             BNE   MATH
198: 116D AD 63 1A          LDA   PARAL
199: 1170 85 FA             STAZ  POINTL SETUP MATRIX POINTER
200: 1172 AD 64 1A          LDA   PARAH
201: 1175 85 FB             STAZ  POINTH
202:
203: 1177 20 E8 11  MATJ    JSR   CRLF
204: 117A A2 10             LDXIM $10
205: 117C A5 FB             LDAZ  POINTH
206: 117E 20 8F 12          JSR   PRBYT  OUTPUT CURRENT MATRIX ADDRESS
207: 1181 A5 FA             LDAZ  POINTL
208: 1183 20 8F 12          JSR   PRBYT
209: 1186 A0 17             LDYIM $17
210: 1188 20 D9 11          JSR   ME
211:
212: 118B 38        MATK    SEC
213: 118C AD 65 1A          LDA   PARBL  HEXDUMP FINISHED?
214: 118F E5 FA             SBCZ  POINTL
215: 1191 AD 66 1A          LDA   PARBH
216: 1194 E5 FB             SBCZ  POINTH
217: 1196 B0 06             BCS   MATL
218: 1198 20 E8 11          JSR   CRLF
219: 119B 4C 5F 10          JMP   LABJUN HEXDUMP IS FINISHED
220:
221: 119E A0 00     MATL    LDYIM $00
222: 11A0 B1 FA             LDAIY POINTL FETCH CURRENT DATA BYTE
223: 11A2 20 8F 12          JSR   PRBYT  OUTPUT CURRENT DATA BYTE
224: 11A5 20 F3 11          JSR   PRSP
225: 11A8 20 13 12          JSR   INCPNT
226: 11AB CA                DEX
227: 11AC D0 DD             BNE   MATK
228: 11AE F0 C7             BEQ   MATJ
229:
230: 11B0 C9 47     GETTAP CMPIM 'G
231: 11B2 D0 0B             BNE   SAVID
232: 11B4 20 8A 14          JSR   GETID  READ DATA FROM TAPE SPEC. BY ID
233: 11B7 30 03             BMI   GETERR ILLEGAL CHARACTER?
234: 11B9 4C 6A 10          JMP   RESALL NORMAL EXIT
235:
236: 11BC 4C 5F 10  GETERR JMP   LABJUN
237:
238: 11BF C9 53     SAVID  CMPIM 'S
239: 11C1 D0 08             BNE   VALNUM NO COMMAND KEY WAS DEPRESSED
240: 11C3 20 3B 14          JSR   SAID   STORE DATA ON TAPE SPECIFIED BY THE PARAMETERS
```

```
0241: 11C6 30 F4          BMI   GETERR ILLEGAL PARAMETER WAS ENTERED
0242: 11C8 4C 6A 10       JMP   RESALL
0243:
0244: 11CB 20 6F 12  VALNUM JSR   HEXNUM INPUT DATA TO BUFFER
0245: 11CE D0 03          BNE   VNA
0246: 11D0 4C 70 10       JMP   READCH
0247:
0248: 11D3 4C 6A 10  VNA    JMP   RESALL
0249:
0250:                     ****************************************
0251:                     *** SUBROUTINES OF THE PRINTER MONITOR ***
0252:                     ****************************************
0253:
0254:
0255:                     MESSY PRINTS A MESSAGE, POINTED BY Y REGISTER
0256:
0257: 11D6 20 E8 11  MESSY  JSR   CRLF    PRINT A CR&LF
0258:
0259: 11D9 B9 BD 13  ME     LDAY  MESS    LOAD CHARACTERS
0260: 11DC C9 03          CMPIM $03     ETX CHARACTER ?
0261: 11DE F0 07          BEQ   MESEND
0262: 11E0 20 34 13       JSR   PRCHA   CHARACTER TO TTY
0263: 11E3 C8             INY
0264: 11E4 4C D9 11       JMP   ME
0265:
0266: 11E7 60        MESEND RTS
0267:
0268:
0269:                     CRLF PRINT CARRIAGE RETURN & LINE FEED
0270:                     PRSP PRINT A SPACE
0271:
0272: 11E8 A9 0D     CRLF   LDAIM $0D
0273: 11EA 20 34 13       JSR   PRCHA   OUTPUT CR
0274: 11ED A9 0A          LDAIM $0A
0275:
0276: 11EF 20 34 13  CLEND  JSR   PRCHA   OUTPUT LF
0277: 11F2 60             RTS
0278:
0279: 11F3 A9 20     PRSP   LDAIM $20
0280: 11F5 4C EF 11       JMP   CLEND   OUTPUT A SPACE
0281:
0282:
0283:                     PRBUFS OUTPUT ADDRESS AND DATA
0284:
0285: 11F8 20 E8 11  PRBUFS JSR   CRLF
0286: 11FB A5 FB          LDAZ  POINTH
0287: 11FD 20 8F 12       JSR   PRBYT   OUTPUT HIGH ORDER ADDR
0288: 1200 A5 FA          LDAZ  POINTL
0289: 1202 20 8F 12       JSR   PRBYT   OUTPUT LOW ORDER ADDR
0290: 1205 20 F3 11       JSR   PRSP
0291: 1208 A0 00          LDYIM $00
0292: 120A B1 FA          LDAIY POINTL FETCH DATA FROM MEMORY
0293: 120C 20 8F 12       JSR   PRBYT
0294: 120F 20 F3 11       JSR   PRSP
0295: 1212 60             RTS
0296:
0297:
0298:                     INCPNT INCRMENT ADDR POINTER BY ONE
0299:
0300: 1213 E6 FA     INCPNT INCZ  POINTL
0301: 1215 D0 02          BNE   IP
0302: 1217 E6 FB          INCZ  POINTH
0303:
0304: 1219 60        IP     RTS
0305:
0306:
0307:                     DECPNT DECREMENT ADDR POINTER BY ONE
0308:
0309: 121A 38             DECPNT SEC
```

```
0310: 121B A5 FA          LDAZ  POINTL 16 BIT SUBTRACTION
0311: 121D E9 01          SBCIM $01
0312: 121F 85 FA          STAZ  POINTL
0313: 1221 A5 FB          LDAZ  POINTH
0314: 1223 E9 00          SBCIM $00
0315: 1225 85 FB          STAZ  POINTH
0316: 1227 60             RTS
0317:
0318:
0319:                SHOWPR SHOW THE CONTENTS OF THE P REGISTER
0320:
0321: 1228 A5 F1     SHOWPR LDAZ  PREG
0322: 122A 8D 67 1A        STA   PRTEMP
0323: 122D A2 08           LDXIM $08     BIT COUNTER
0324:
0325: 122F 0E 67 1A  SPRA   ASL   PRTEMP SHIFT OUT THE BITS
0326: 1232 90 09           BCC   SPRB    IS IT A 0 OR 1 ?
0327: 1234 A9 01           LDAIM $01
0328: 1236 20 9B 12        JSR   PRNIBL PRINT A '1'
0329: 1239 CA              DEX
0330: 123A D0 F3           BNE   SPRA    ALL BITS PRINTED ?
0331: 123C 60              RTS
0332:
0333: 123D A9 00     SPRB   LDAIM $00
0334: 123F 20 9B 12        JSR   PRNIBL PRINT A '0'
0335: 1242 CA              DEX
0336: 1243 D0 EA           BNE   SPRA    ALL BITS PRINTED ?
0337: 1245 60              RTS
0338:
0339:
0340:                JUNIOR PRINT 'JUNIOR'
0341:                EDITOR PRINT 'EDITOR'
0342:                ASSEM  PRINT 'ASSEMBLER'
0343:
0344: 1246 A0 00     JUNIOR LDYIM $00
0345:
0346: 1248 20 D6 11  JUN    JSR   MESSY
0347: 124B 20 E8 11        JSR   CRLF
0348: 124E 60              RTS
0349:
0350: 124F A0 07     EDITOR LDYIM $07
0351: 1251 4C 48 12        JMP   JUN
0352:
0353: 1254 A0 0E     ASSEM  LDYIM $0E
0354: 1256 4C 48 12        JMP   JUN
0355:
0356:
0357:                RESET SUBROUTINES
0358:
0359: 1259 A0 00     RESPAR LDYIM $00
0360: 125B 8C 63 1A        STY   PARAL
0361: 125E 8C 64 1A        STY   PARAH
0362: 1261 8C 65 1A        STY   PARBL
0363: 1264 8C 66 1A        STY   PARBH
0364: 1267 60              RTS
0365:
0366: 1268 A0 00     RESIN  LDYIM $00
0367: 126A 84 F8           STYZ  INL
0368: 126C 84 F9           STYZ  INH
0369: 126E 60              RTS
0370:
0371:                HEXNUM >CONVERT AN ASCII CHAR. TO A HEX NIBBLE
0372:                       >SHIFT HEX DATA NIBBLE INTO BUFFER
0373:                       >PRINT 'WHAT?' IF CHARACTER WAS NOT VALID
0374:                       >RETURN WITH Z=1, IF VALID CHARACTER
0375:                       >RETURN WITH N=1, IF NOT VALID CHARACTER
0376:
0377: 126F 20 1E 14  HEXNUM JSR   ASHETT ASCII HEX CONVERSION
0378: 1272 30 10           BMI   HNUB    NOT VALID ?
```

```
0379: 1274 A2 04            LDXIM $04     SET NIBBLE COUNTER
0380:
0381: 1276 06 F8      HNUA  ASLZ  INL
0382: 1278 26 F9            ROLZ  INH
0383: 127A CA               DEX
0384: 127B D0 F9            BNE   HNUA
0385: 127D 05 F8            ORAZ  INL     NIBBLE TO INPUT BUFFER
0386: 127F 85 F8            STAZ  INL
0387: 1281 A0 00            LDYIM $00     SET Z FLAG
0388: 1283 60               RTS
0389:
0390: 1284 A0 46      HNUB  LDYIM $46
0391: 1286 20 D6 11         JSR   MESSY   'WHAT?'
0392: 1289 20 E8 11         JSR   CRLF
0393: 128C A0 FF            LDYIM $FF     SET N FLAG
0394: 128E 60               RTS
0395:
0396:
0397:                 PRBYT >CONVERT AN BYTE STORED IN ACCU TO
0398:                       TWO ASCII CHARACTERS AND PRINT THEM
0399:
0400: 128F 48         PRBYT PHA           SAVE BYTE
0401: 1290 4A               LSRA          GET UPPER NIBBLE
0402: 1291 4A               LSRA
0403: 1292 4A               LSRA
0404: 1293 4A               LSRA
0405: 1294 20 A4 12         JSR   NIBASC  NIBBLE TO ASCII CONVERSION
0406: 1297 20 34 13         JSR   PRCHA   PRINT UPPER NIBBLE
0407: 129A 68               PLA           GET BYTE AGAIN
0408:
0409: 129B 29 0F      PRNIBL ANDIM $0F    GET LOWER NIBBLE
0410: 129D 20 A4 12         JSR   NIBASC
0411: 12A0 20 34 13         JSR   PRCHA   PRINT LOWER NIBBLE
0412: 12A3 60               RTS
0413:
0414:
0415:                 NIBASC >CONVERT A HEX DATA NIBBLE TO AN ASCII CHARACTER
0416:
0417: 12A4 C9 0A      NIBASC CMPIM $0A    NUMBER OR LETTER ?
0418: 12A6 18               CLC
0419: 12A7 30 02            BMI   NA
0420: 12A9 69 07            ADCIM $07
0421:
0422: 12AB 69 30      NA    ADCIM $30
0423: 12AD 60               RTS
0424:
0425:
0426:                 RECCHA >RECEIVE 1 ASCII CHARACTER FROM PRINTER
0427:                       >RETURN WITH ASCII CHARACTER IN ACCU
0428:                       >SAVE X REGISTER
0429:
0430: 12AE 2C 80 1A   RECCHA BIT   PAD     WAIT FOR START BIT
0431: 12B1 30 FB            BMI   RECCHA
0432: 12B3 8E 61 1A         STX   TEMPB   SAVE INDEX X
0433: 12B6 A2 08            LDXIM $08     BIT COUNTER IS 8
0434: 12B8 20 03 13         JSR   DELHBT  DELAY HALF BIT TIME
0435:
0436: 12BB 20 12 13   RECA  JSR   DELBIT  DELAY ONE BIT TIME
0437:
0438: 12BE 2C 80 1A   RECD  BIT   PAD     ONE/ZERO CHECK
0439: 12C1 10 0A            BPL   RECB    BRANCH IF ZERO
0440: 12C3 38               SEC           BIT IS '1'
0441: 12C4 6E 62 1A         ROR   CHA     ROTATE CARRY INTO CHARACTER
0442: 12C7 CA               DEX           SET UP FOR NEXT BIT
0443: 12C8 D0 F1            BNE   RECA    ALL BITS READ?
0444: 12CA 4C D4 12         JMP   RECC
0445:
0446: 12CD 18         RECB  CLC           BIT IS '0'
0447: 12CE 6E 62 1A         ROR   CHA
```

```
0448: 12D1 CA              DEX
0449: 12D2 DO E7           BNE   RECA
0450:
0451: 12D4 20 12 13  RECC  JSR   DELBIT WAIT FOR STOP BIT TIME
0452: 12D7 AD 62 1A        LDA   CHA
0453: 12DA 29 7F           ANDIM $7F     BIT 7 MUST BE ZERO
0454: 12DC AE 61 1A        LDX   TEMPB   RESTORE INDEX X
0455: 12DF 60              RTS
0456:
0457:
0458:                COMTIM >COMPUTE BIT TIME
0459:
0460: 12E0 18        COMTIM CLC
0461: 12E1 AD 5A 1A        LDA   CNTL    16 BIT ADD
0462: 12E4 69 01           ADCIM $01
0463: 12E6 8D 5A 1A        STA   CNTL
0464: 12E9 AD 5B 1A        LDA   CNTH
0465: 12EC 69 00           ADCIM $00
0466: 12EE 8D 5B 1A        STA   CNTH
0467: 12F1 2C 80 1A        BIT   PAD     START BIT FINISHED ?
0468: 12F4 10 EA           BPL   COMTIM
0469: 12F6 AD 5A 1A        LDA   CNTL    SET UP FOR
0470: 12F9 8D 5E 1A        STA   TIML    HALF BIT TIME COMPUTATION
0471: 12FC AD 5B 1A        LDA   CNTH
0472: 12FF 8D 5F 1A        STA   TIMH
0473: 1302 60              RTS
0474:
0475:
0476:                DELBIT >DELAY 1 BIT TIME
0477:                DELHBT >DELAY 1/2 BIT TIME
0478:
0479: 1303 AD 5C 1A  DELHBT LDA   CNTHL   FETCH 1/2 BIT TIME
0480: 1306 8D 5E 1A        STA   TIML
0481: 1309 AD 5D 1A        LDA   CNTHH
0482: 130C 8D 5F 1A        STA   TIMH
0483: 130F 4C 1E 13        JMP   CNTDN   START WITH BIT COUNT DOWN
0484:
0485: 1312 AD 5A 1A  DELBIT LDA   CNTL    FETCH 1 BIT TIME
0486: 1315 8D 5E 1A        STA   TIML
0487: 1318 AD 5B 1A        LDA   CNTH
0488: 131B 8D 5F 1A        STA   TIMH
0489:
0490: 131E 38        CNTDN  SEC
0491: 131F AD 5E 1A        LDA   TIML    16 BIT SUBTRACTION
0492: 1322 E9 01           SBCIM $01     COUNT DOWN
0493: 1324 8D 5E 1A        STA   TIML
0494: 1327 AD 5F 1A        LDA   TIMH
0495: 132A E9 00           SBCIM $00
0496: 132C 8D 5F 1A        STA   TIMH
0497: 132F EA              NOP           EQUALIZE 4 MICRO SEC
0498: 1330 EA              NOP
0499: 1331 B0 EB           BCS   CNTDN   COUNT DOWN FINISHED ?
0500: 1333 60              RTS
0501:
0502:                PRCHA >TRANSMIT AN ASCII CHARACTER STORED IN
0503:                       ACCU TO PRINTER
0504:                      >SAVE INDEX X
0505:
0506: 1334 8E 60 1A  PRCHA  STX   TEMPA   SAVE INDEX X
0507: 1337 8D 62 1A        STA   CHA
0508: 133A AD 82 1A        LDA   PBD
0509: 133D 29 FE           ANDIM $FE     TRNSMIT START BIT
0510: 133F 8D 82 1A        STA   PBD
0511: 1342 20 12 13        JSR   DELBIT  DELAY OF START BIT
0512: 1345 A2 07           LDXIM $07     SET UP FOR 7 DATA BITS
0513:
0514: 1347 4E 62 1A  PRA    LSR   CHA     SHIFT OUT CHARACTER
0515: 134A 90 30           BCC   PRC     BRANCH IF '0'
0516: 134C AD 82 1A        LDA   PBD
```

```
0517: 134F 09 01          ORAIM $01    OUTPUT A LOG '1'
0518: 1351 8D 82 1A       STA   PBD
0519:
0520: 1354 20 12 13  PRB  JSR   DELBIT DELAY 1 BIT TIME
0521: 1357 CA             DEX         SET UP FOR NEXT BIT
0522: 1358 D0 ED          BNE   PRA     ALL BITS TRANSMITTED ?
0523: 135A AE 59 1A       LDX   STPBIT X := AMOUNT OF STOP BITS + 1
0524:
0525: 135D AD 82 1A  PRD  LDA   PBD
0526: 1360 09 01          ORAIM $01    FIRST NONE PARITY
0527: 1362 8D 82 1A       STA   PBD    AND THEN 1 STOP BIT
0528: 1365 20 12 13       JSR   DELBIT
0529: 1368 CA             DEX
0530: 1369 D0 F2          BNE   PRD
0531: 136B 2C 80 1A       BIT   PAD    TEST FOR BREAK
0532: 136E 10 04          BPL   BRKTST
0533: 1370 AE 60 1A       LDX   TEMPA  RESTORE INDEX X
0534: 1373 60             RTS
0535:
0536: 1374 2C 80 1A  BRKTST BIT PAD    KEY RELEASED ?
0537: 1377 10 FB          BPL   BRKTST
0538: 1379 6C 7C 1A       JMI   BRKT   JUMP TO AN USER SELECTABLE VECTOR
0539:
0540: 137C AD 82 1A  PRC  LDA   PBD
0541: 137F 29 FE          ANDIM $FE    OUTPUT A LOG '0'
0542: 1381 8D 82 1A       STA   PBD
0543: 1384 4C 54 13       JMP   PRB
0544:
0545:                INPAR >PARAMETER INPUT OF MATRIX
0546:
0547: 1387 20 AE 12  INPAR JSR   RECCHA WAIT FOR A CHARACTER
0548: 138A C9 2C          CMPIM ',     IS IT A COLON ?
0549: 138C F0 07          BEQ   IPA
0550: 138E 20 6F 12       JSR   HEXNUM FILLUP INPUT BUFFER
0551: 1391 30 29          BMI   IPD    RETURN, IF NOT VALID
0552: 1393 F0 F2          BEQ   INPAR  ELSE CONTINUE
0553:
0554: 1395 A5 F8     IPA  LDAZ  INL    INPUT TO PARAMETER BUFFER
0555: 1397 8D 63 1A       STA   PARAL
0556: 139A A5 F9          LDAZ  INH
0557: 139C 8D 64 1A       STA   PARAH
0558: 139F 20 68 12       JSR   RESIN
0559:
0560: 13A2 20 AE 12  IPB  JSR   RECCHA WAIT FOR A CHARACTER
0561: 13A5 C9 0D          CMPIM $0D    WAS IT A CARRIAGE RETURN ?
0562: 13A7 F0 07          BEQ   IPC
0563: 13A9 20 6F 12       JSR   HEXNUM
0564: 13AC 30 0E          BMI   IPD    VALID CHARACTER ?
0565: 13AE F0 F2          BEQ   IPB
0566:
0567: 13B0 A5 F9     IPC  LDAZ  INH    INPUT TO PARAMETER BUFFER
0568: 13B2 8D 66 1A       STA   PARBH
0569: 13B5 A5 F8          LDAZ  INL
0570: 13B7 8D 65 1A       STA   PARBL
0571: 13BA A0 00          LDYIM $00
0572:
0573: 13BC 60        IPD  RTS
0574:
0575:
0576:                STRING LOOKUP TABLE
0577:
0578: 13BD 4A        MESS =     'J      Y = 00
0579: 13BE 55             =     'U
0580: 13BF 4E             =     'N
0581: 13C0 49             =     'I
0582: 13C1 4F             =     'O
0583: 13C2 52             =     'R
0584: 13C3 03             =     $03
0585: 13C4 45             =     'E      Y = 07
```

```
0586: 13C5 44           =     'D
0587: 13C6 49           =     'I
0588: 13C7 54           =     'T
0589: 13C8 4F           =     'O
0590: 13C9 52           =     'R
0591: 13CA 03           =     $03
0592: 13CB 41           =     'A        Y = 0E
0593: 13CC 53           =     'S
0594: 13CD 53           =     'S
0595: 13CE 45           =     'E
0596: 13CF 4D           =     'M
0597: 13D0 03           =     $03
0598: 13D1 41           =     'A        Y = 14
0599: 13D2 43           =     'C
0600: 13D3 43           =     'C
0601: 13D4 3A           =     ':        Y = 17
0602: 13D5 20           =     ' 
0603: 13D6 03           =     $03
0604: 13D7 59           =     'Y        Y = 1A
0605: 13D8 20           =     ' 
0606: 13D9 20           =     ' 
0607: 13DA 3A           =     ':
0608: 13DB 20           =     ' 
0609: 13DC 03           =     $03
0610: 13DD 58           =     'X        Y = 20
0611: 13DE 20           =     ' 
0612: 13DF 20           =     ' 
0613: 13E0 3A           =     ':
0614: 13E1 20           =     ' 
0615: 13E2 03           =     $03
0616: 13E3 50           =     'P        Y = 26
0617: 13E4 43           =     'C
0618: 13E5 20           =     ' 
0619: 13E6 3A           =     ':
0620: 13E7 20           =     ' 
0621: 13E8 03           =     $03
0622: 13E9 53           =     'S        Y = 2C
0623: 13EA 50           =     'P
0624: 13EB 20           =     ' 
0625: 13EC 3A           =     ':
0626: 13ED 20           =     ' 
0627: 13EE 03           =     $03
0628: 13EF 50           =     'P        Y = 32
0629: 13F0 52           =     'R
0630: 13F1 20           =     ' 
0631: 13F2 3A           =     ':
0632: 13F3 20           =     ' 
0633: 13F4 03           =     $03
0634: 13F5 20           =     '          Y = 38
0635: 13F6 20           =     ' 
0636: 13F7 20           =     ' 
0637: 13F8 20           =     ' 
0638: 13F9 20           =     ' 
0639: 13FA 4E           =     'N
0640: 13FB 56           =     'V
0641: 13FC 20           =     ' 
0642: 13FD 42           =     'B
0643: 13FE 44           =     'D
0644: 13FF 49           =     'I
0645: 1400 5A           =     'Z
0646: 1401 43           =     'C
0647: 1402 03           =     $03
0648: 1403 57           =     'W        Y = 46
0649: 1404 48           =     'H
0650: 1405 41           =     'A
0651: 1406 54           =     'T
0652: 1407 3F           =     '?
0653: 1408 03           =     $03
0654: 1409 52           =     'R        Y = 4C
```

```
0655: 140A 45                =       'E
0656: 140B 41                =       'A
0657: 140C 44                =       'D
0658: 140D 59                =       'Y
0659: 140E 03                =       $03
0660: 140F 48                =       'H      Y = 52
0661: 1410 45                =       'E
0662: 1411 58                =       'X
0663: 1412 44                =       'D
0664: 1413 55                =       'U
0665: 1414 4D                =       'M
0666: 1415 50                =       'P
0667: 1416 3A                =       ':
0668: 1417 20                =       ' 
0669: 1418 03                =       $03
0670: 1419 53                =       'S      Y = 5C
0671: 141A 41                =       'A
0672: 141B 3A                =       ':
0673: 141C 20                =       ' 
0674: 141D 03                =       $03
0675:
0676:                      ASHETT = ASCHEX
0677:
0678:
0679:                      CONVERT AN ASCII CHARACTER TO A HEX DATA NIBBLE.
0680:
0681:                      1) RETURN WITH CONVERTED HEX NUMBER IN ACCU
0682:                      2) N = 1, IF NOT VALID HEX NUMBER
0683:                      3) Z = 1, IF VALID HEX NUMBER
0684:
0685: 141E C9 30       ASHETT CMPIM $30     IGNORE 00...2F
0686: 1420 30 0C              BMI   NOTVAT
0687: 1422 C9 3A              CMPIM $3A
0688: 1424 30 0B              BMI   VALIT
0689: 1426 C9 41              CMPIM $41     IGNORE 3A...40
0690: 1428 30 04              BMI   NOTVAT
0691: 142A C9 47              CMPIM $47     IGNORE 47...7F
0692: 142C 30 03              BMI   VALIT
0693:
0694: 142E A0 FF       NOTVAT LDYIM $FF     SET N-FLAG
0695: 1430 60                 RTS           ERROR EXIT
0696:
0697: 1431 C9 40       VALIT  CMPIM $40     ASCII HEX CONVERSION
0698: 1433 30 03              BMI   VALT
0699: 1435 18                 CLC
0700: 1436 69 09              ADCIM $09
0701:
0702: 1438 29 0F       VALT   ANDIM $0F
0703: 143A 60                 RTS
0704:
0705: 143B 20 AE 12    SAID   JSR   RECCHA WAIT FOR A CHARACTER
0706: 143E C9 2C              CMPIM ',
0707: 1440 F0 07              BEQ   SIC    IT WAS A DELIMITER
0708: 1442 20 6F 12           JSR   HEXNUM READ PARAMETER = ID
0709: 1445 30 3F              BMI   SIB
0710: 1447 F0 F2              BEQ   SAID
0711:
0712: 1449 A5 F8       SIC    LDAZ  INL    SAVE ID
0713: 144B 8D 79 1A           STA   ID
0714: 144E C9 00              CMPIM $00     ID = 00 & FF ARE NOT VALID
0715: 1450 F0 35              BEQ   SIA
0716: 1452 C9 FF              CMPIM $FF
0717: 1454 F0 31              BEQ   SIA
0718: 1456 20 68 12           JSR   RESIN  RESET INPUT BUFFER
0719: 1459 20 87 13           JSR   INPAR  READ SA AND EA
0720: 145C 30 28              BMI   SIB    NOT VALID CHARACTER
0721: 145E AD 63 1A           LDA   PARAL  SAVE ALL PARAMETERS
0722: 1461 8D 70 1A           STA   SAL
0723: 1464 AD 64 1A           LDA   PARAH
```

```
0724: 1467 8D 71 1A          STA   SAH
0725: 146A AD 65 1A          LDA   PARBL
0726: 146D 8D 72 1A          STA   EAL
0727: 1470 AD 66 1A          LDA   PARBH
0728: 1473 8D 73 1A          STA   EAH
0729: 1476 20 DF 09          JSR   DUMP    STORE DATA ON TAPE
0730: 1479 20 BC 14          JSR   RESTTY I/O RESET
0731: 147C A0 4C             LDYIM $4C     'READY'
0732: 147E 20 D6 11          JSR   MESSY
0733: 1481 20 E8 11          JSR   CRLF
0734: 1484 A0 00             LDYIM $00     NORMAL EXIT
0735:
0736: 1486 60          SIB   RTS
0737:
0738: 1487 A0 FF       SIA   LDYIM $FF     ERROR EXIT
0739: 1489 60                RTS
0740:
0741:
0742: 148A 20 A2 13    GETID JSR   IPB     READ ID
0743: 148D 30 17             BMI   GA      NOT VALID PARAMETER
0744: 148F 8D 79 1A          STA   ID      SAVE ID
0745: 1492 C9 FF             CMPIM $FF     SPECIAL ID ?
0746: 1494 F0 11             BEQ   IDPAR
0747:
0748: 1496 20 02 0B    GB    JSR   RDTAPE READ DATA FROM TAPE
0749: 1499 20 BC 14          JSR   RESTTY I/O RESET
0750: 149C A0 4C             LDYIM $4C     'READY'
0751: 149E 20 D6 11          JSR   MESSY
0752: 14A1 20 E8 11          JSR   CRLF
0753: 14A4 A0 00             LDYIM $00     NORMAL EXIT
0754:
0755: 14A6 60          GA    RTS
0756:
0757: 14A7 A0 5C       IDPAR LDYIM $5C     'SA: '
0758: 14A9 20 D6 11          JSR   MESSY
0759: 14AC 20 EB 14          JSR   IPBRES READ A SPECIAL ID
0760: 14AF 30 F5             BMI   GA      NOT VALID PARAMETER
0761: 14B1 8D 70 1A          STA   SAL     SAVE START ADDRESS
0762: 14B4 A5 F9             LDAZ  INH
0763: 14B6 8D 71 1A          STA   SAH
0764: 14B9 4C 96 14          JMP   GB
0765:
0766:
0767:
0768: 14BC A9 67       RESTTY LDAIM $67
0769: 14BE 8D 82 1A          STA   PBD
0770: 14C1 A9 00             LDAIM $00
0771: 14C3 8D 80 1A          STA   PAD
0772: 14C6 A9 7F             LDAIM $7F
0773: 14C8 8D 81 1A          STA   PADD
0774: 14CB 8D 83 1A          STA   PBDD
0775: 14CE 60                RTS
0776:                  ************************************************
0777:                  *** STEP BY STEP PROGRAM OF THE PRINTER MONITOR ***
0778:                  ************************************************
0779:
0780:
0781:                  > THE NMI VECTOR FOR STEP BY STEP IS SET AUTOMATICALLY
0782:                  > STEP SWITCH: ON POSITION
0783:                  > K4 AND K5 DISABLE THE SYNC SIGNAL OF THE PROCESSOR
0784:                  > A HARDWARE MODIFICATION IS REQIRED (SEE BOOK 3)
0785:
0786:
0787: 14CF 85 F3     STEP  STAZ  ACC     SAVE ACCU
0788: 14D1 68                PLA         GET PREG
0789: 14D2 85 F1             STAZ  PREG
0790: 14D4 68                PLA         GET PCL
0791: 14D5 85 EF             STAZ  PCL
0792: 14D7 85 FA             STAZ  POINTL PC OF THE NEXT INSTRUCTION
```

```
0793: 14D9 68              PLA          GET PCH
0794: 14DA 85 F0           STAZ  PCH
0795: 14DC 85 FB           STAZ  POINTH
0796: 14DE 84 F4           STYZ  YREG
0797: 14E0 86 F5           STXZ  XREG
0798: 14E2 BA              TSX          GET OLD STACK POINTER
0799: 14E3 86 F2           STXZ  SPUSER AND SAVE IT
0800: 14E5 20 F8 11        JSR   PRBUFS OPEN NEXT CELL
0801: 14E8 4C 6A 10        JMP   RESALL BACK TO MONITOR
0802:
0803:
0804:
0805:                 *** SPECIAL ID ***
0806:
0807: 14EB A9 00      IPBRES LDAIM $00
0808: 14ED 85 F8           STAZ  INL    RESET INPUT BUFFER
0809: 14EF 85 F9           STAZ  INH
0810: 14F1 4C A2 13        JMP   IPB    CONTINUE
0811:
0812:
0813:                 **************************
0814:                 *** END OF THE PROGRAM ***
0815:                 **************************
ID=

-T
```

SYMBOL TABLE 3000 3594

ACC	00F3	ADJPNT	0C72	ASCHEX	0C19	ASHETT	141E
ASSEM	1254	BEGADH	00E3	BEGADL	00E2	BEGIN	1ED3
BITS	1A75	BRKT	1A7C	BRKTST	1374	BTWEEN	0BE8
BYTE	1A6A	CENDH	00E9	CENDL	00E8	CHAR	1A6B
CHARVU	0C5D	CHA	1A62	CHECK	0B8A	CHKH	1A6F
CHKID	0B9E	CHKL	1A6E	CHKSUM	0C4B	CLEND	11EF
CNTA	1AF4	CNTC	1AF6	CNTDN	131E	CNTH	1A5B
CNTHH	1A5D	CNTHL	1A5C	CNTL	1A5A	COLDST	1CB5
COMPNT	0949	COMTIM	12E0	CONTIN	111C	CRLF	11E8
DATATR	0A40	DAT	089C	DECPNT	121A	DELBIT	1312
DELHBT	1303	DELY	0927	DISCNT	1A68	DUMP	09DF
DUMPT	09F3	EAH	1A73	EAL	1A72	EDITOR	124F
ENDADH	00E5	ENDADL	00E4	FILES	0873	FILMEM	0B70
FIRST	1A76	FMA	0B84	GA	14A6	GANG	1A78
GB	1496	GETA	0800	GETB	084E	GETERR	11BC
GETID	148A	GETKEY	1DF9	GETTAP	11B0	GET	0840
HEXDAT	0A6A	HEXNUM	126F	HIGH	0AC8	HIGHER	1A6C
HIG	0ACB	HNUA	1276	HNUB	1284	ID	1A79
IDPAR	14A7	INCPNT	1213	INH	00F9	INITPR	1000
INL	00F8	INPAR	1387	IP	1219	IPA	1395
IPBRES	14EB	IPB	13A2	IPC	13B0	IPD	13BC
JUNIOR	1246	JUN	1248	LABJUN	105F	LDAINH	1DA7
LIST	10C9	LO	0AE8	LOWER	1A6D	LOW	0AE5
MATD	113F	MATF	1142	MATG	1156	MATH	115E
MATJ	1177	MATK	118B	MATL	119E	MATRIX	1131
ME	11D9	MESEND	11E7	MESS	13BD	MESSY	11D6
MINUS	1080	NA	12AB	NIBASC	12A4	NIBOUT	0A9A
NIB	0AA1	NMI	1A7A	NOTVAL	0C29	NOTVAT	142E
ONE	0AA8	OUTBT	0A8B	OUTBTC	0A7A	OUTCH	0AA3
PADD	1A81	PAD	1A80	PARAH	1A64	PARAL	1A63
PARBH	1A66	PARBL	1A65	PBDD	1A83	PBD	1A82
PC	111F	PCH	00F0	PCL	00EF	PLUS	085E
PLU	1073	PNT	109F	POINTH	00FB	POINTL	00FA
PRA	1347	PRBUFS	11F8	PRBYT	128F	PRB	1354
PRCHA	1334	PRC	137C	PRD	135D	PREG	00F1
PRNIBL	129B	PRSP	11F3	PRTEMP	1A67	RBA	0BFA
RBB	0BFB	RDBA	0BD1	RDBB	0BD4	RDBC	0BE4
RDBIT	0BC2	RDBYT	0BF3	RDCH	0C36	RDFLAG	1AD5
RDSA	0B60	RDTAPE	0B02	RDTDIS	1AD4	READ	0C38
READCH	1070	RECA	12BB	RECB	12CD	RECC	12D4
RECCHA	12AE	RECD	12BE	RESALL	106A	RESET	1C1D
RESIN	1268	RESPAR	1259	RESTTY	14BC	RUN	10B2
SAH	1A71	SAID	143B	SAL	1A70	SAVE	0852
SAVID	11BF	SBEGH	0989	SBEGL	0997	SEAH	096B
SEAL	097A	SECOND	1A77	SENDH	09A5	SENDL	09B3
SHIFT	08A3	SHOWPR	1228	SIA	1487	SIB	1486
SIC	1449	SID	093B	SPACE	108D	SPRA	122F
SPRB	123D	SPUSER	00F2	SSAH	094D	SSAL	095C
STAR	0B45	STARA	0B55	STBEGH	08EC	STBEGL	08F7
STEAH	08D4	STEAL	08E0	STENDH	0902	STENDL	090D
STEP	14CF	STPBIT	1A59	STRTBT	1039	STSAH	08BC
STSAL	08C8	SY	1A69	SYNC	0B1C	SYNCA	0B21
SYNCNT	1A74	SYNCS	0A1F	SYNVEC	0B9B	TAPDIS	0936
TDISP	091B	TEMP	00FC	TEMPA	1A60	TEMPB	1A61
TEMPX	00FD	TENSYN	0B36	TIMH	1A5F	TIML	1A5E
TLOOK	09BB	TPINIT	0810	TPI	0821	TPTXT	082E
TPVEC	0918	TTXT	0833	VALID	0C2C	VALIT	1431
VALNUM	11CB	VALT	1438	VAL	0C33	VNA	11D3
VU	0C64	WARMST	1CCA	XREG	00F5	YREG	00F4
ZERO	0AB8	ZRO	0AC1				

EPROM-uitbreiding van de standaard-monitor en EPROM-uitbreiding van PM

Op minstens twee plaatsen in boek 3 hebben we het gehad over de omweg die moet worden afgelegd om de standaardmonitor of PM te bereiken tijdens of na afloop van een gebruikersprogramma waarin decimaal wordt gerekend. Zie Aanhangsel 2 van boek 3 en hoofdstuk 12, pagina's 167 . . . 169, Die omweg bestond uit een stukje programma in PIA-RAM. Welnu: die omweg is nog steeds nodig in de besproken gevallen, maar de bijbehorende instrukties hoeft men nu niet telkens zelf in PIA-RAM te zetten, maar zijn permanent aanwezig in de PM/PME-EPROM (ESS 507N). Voor de details wordt verwezen naar de bijgaande figuren 1a en 1b. Omdat er nu geen gebruik meer wordt gemaakt van PIA-RAM kan de derde aansluiting K6 op de imperial-print vervallen (zie figuur 17 van hoofdstuk 12). De instrukties van BINAR en PMBINA bevinden zich nu in de PM/PME-EPROM; de aansluiting K4 op de imperialprint zorgt ervoor dat deze instrukties ononderbroken worden uitgevoerd. En we zeggen het hier nòg maar eens: het signaal K7 moet altijd op de imperial-print zijn aangesloten.