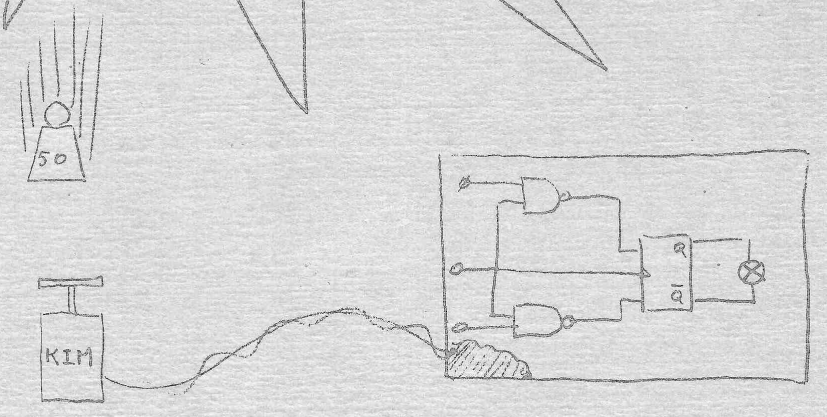
– KIM

PROGRAMMERING

– DIGITAAL SCHAKELEN

– SIEP DE VRIES



Hoofdstuk 1.

Inleiding:

Het doel van dit boek is om diegene, die een microcomputer KIM-1 aangeschaft heeft, een aantal min of meer kant-en-klare programma's te verschaffen, die gebruikt kunnen worden in de meeste toepassingen, waarbij het een of andere "ding" bestuurd of geregeld moet worden.

De programma's zijn in zoverre klaar, dat de gebruiker zelf nog moet coderen. Dit is gedaan om de programma's algemeen bruikbaar te houden. De meeste programma's zijn als subroutine geschreven en kunnen met een JSR aangeroepen worden.

Rond ieder programma zijn een aantal zaken, die door de gebruiker bepaald moeten worden.

Dit zijn:

A - De plaats van het programma in de KIM.

De bepaling hiervan is een volgordekwestie. Als programma A, B en C alle drie tegelijkertijd in het geheugen aanwezig moeten zijn, is het belangrijk, dat ze alle drie hun eigen lokaties voor het programma bezetten.

Eén lokatie kan slechts een instruktie of gedeelte daarvan, die bij één van de drie programma's behoort, bezetten.

Waar het programma zich bevindt, is verder totaal onbelangrijk voor de werking.

Voorbeeld: We hebben drie programam's, genaamd SCAN,ZOEK en PLAATS. (Met de naam van een programma wordt meestal bedoeld de naam van de eerste lokatie van het programma, die een instruktie bevat).

SCAN is $83 lokaties lang, terwijl ZOEK $25 en PLAATS $4A lokaties in beslag neemt. Als er nu geen andere programma's benodigd zijn, zal SCAN kunnen beginnen bij $200, ZOEK bij $283 en PLAATS bij $2A8.

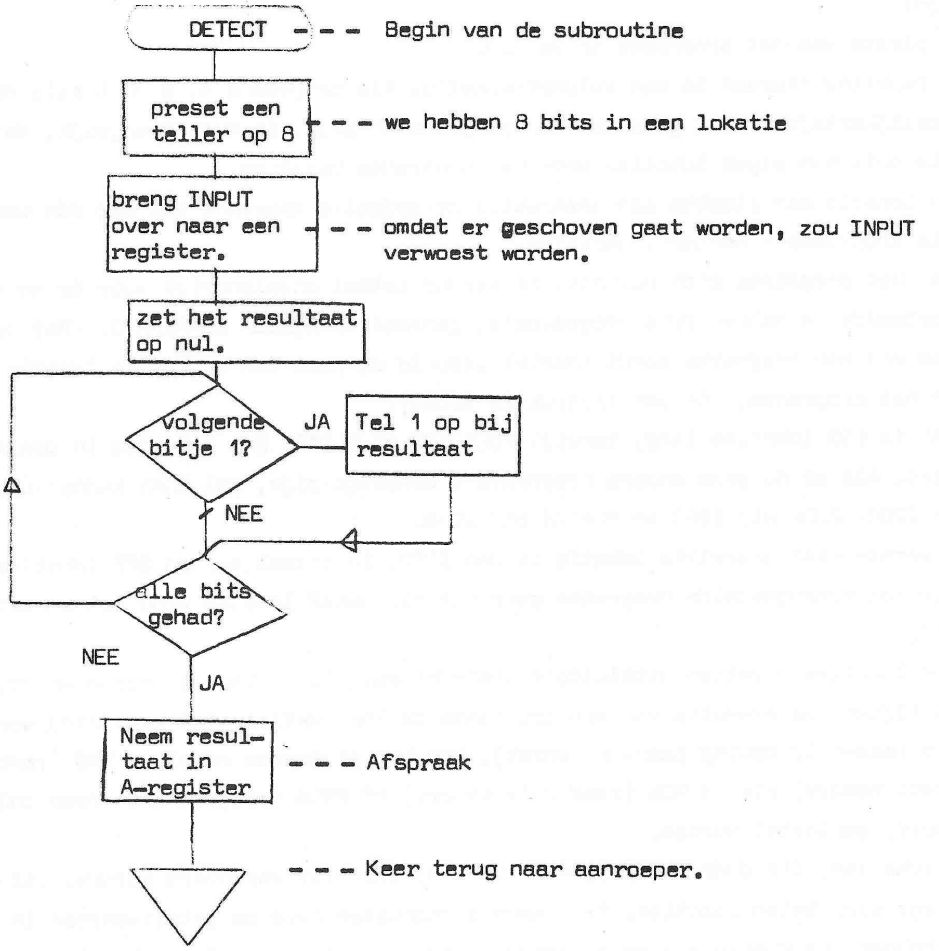
De eerste niet gebruikte lokatie is dan $2F2. In totaal worden $F2 lokaties door het samengestelde programma gebruikt nl. vanaf lokatie $200 tot en met $2F1.

Deze lokaties bevatten uitsluitend instrukties. Als er vanuit gegaan wordt, dat tijdens de executie van een programma de instrukties niet gewijzigd worden (dus read-only coding gebruikt wordt), kan het programma zowel in RAM (random access memory) als in ROM (read only memory) of PROM (programmable read only memory) geplaatst worden.

B - De lokaties, die door het programma tijdens executie veranderd worden. Dit zijn de zg: variabelen lokaties. Programma's gebruiken deze om getalswaardes in te onthouden. We kunnen 2 soorten onderscheiden nl: interne of lokale variabelen en externe of globale variabelen.

De interne variabelen zijn lokaties, die alleen van belang zijn voor het programma of de subroutine, zolang hij bezig is uitgevoerd te worden door de KIM-computer.

Deze variabelen kunnen gemeenschappelijk gebruikt worden door meerdere programmadelen, zolang deze elkaar niet aanroepen. De externe variabelen zijn lokaties, die getallen bevatten, waarvan meerdere programmadelen gebruik maken en waarvan de inhoud niet verwoest mag worden door willekeurig gebruik. Deze lokaties bevatten meestal getallen, die direkt iets te maken hebben met het doel van het totale samengestelde programma.
Deze beide soorten variabelen worden altijd gedefinieerd aan het begin of het eind van een programma of subroutine.
Voorbeeld: Een subroutine DETECT onderzoekt hoeveel bitjes van de lokatie INPUT een 1 bevatten. Bij terugkeer bevat het A-register dit aantal.



STROOMDIAGRAM VAN SUBROUTINE DETECT.

```
; - - - SUBROUTINE DETECT
;
;       DEZE SUBROUTINE TELT HOEVEEL BITS VAN DE LOKATIE INPUT
;       1 zijn. BIJ TERUGKEER STAAT HET AANTAL IN HET A-REGISTER.
;
; - - - INTERNE VARIABELEN:
;
 TELLER= ....        AANTAL TE ONDERZOEKEN BITS
 RESULT= ....        AANTAL ENEN
;
; - - - EXTERNE VARIABELEN:
;
 INPUT= ....         TE ONDERZOEKEN LOKATIE
;
; - - - BEGIN VAN DE SUBROUTINE
;
 DETECT LDA # 8      PRESET EEN TELLER OP 8
        STA TELLER
        LDA # 0      MAAK HET RESULTAAT
        STA RESULT NUL
        LDA INPUT    TE ONDERZOEKEN GETAL
 TEST   ROL          SCHUIF 1 PLAATS
        BCC NIETS    BITJE = 1?
        INC RESULT JA, TEL 1 BIJ RESULTAAT
 NIETS  DEC TELLER ALLE BITS GEHAD?
        BNE TEST     BRANCH ALS NEE
        LDA RESULT JA, RESULTAAT NAAR A
        RTS          KEER TERUG.
```

SUBROUTINE DETECT.

In dit voorbeeld zien we dat er twee interne variabelen zijn nl: TELLER en RESULT. De inhoud van deze lokaties is alleen van belang tijdens het uitvoeren (executie) van de subroutine. De werking van de subroutine wordt niet beïnvloed door de inhoud van deze lokaties voor en na de executie. Er is slechts 1 externe variabele nl: INPUT en deze bevat een getal, dat in de definitie van het probleem voorkomt.

Dit voorbeeld is een eigenschap van interne t.o.v. externe variabelen te zien: Ze worden aan het begin van een programma op een waarde gepreset. We noemen dit gedeelte van een programma het initialiseren. Ieder programmadeel of subroutine dient te beginnen met dit initialiseren.
De externe variabelen worden hoogstens gepreset direkt na het starten van het totale programma.

C - De benodigde randapparatuur voor een programma. Hiermee worden alle adressen bedoeld, die niet direkt een algemeen bruikbare RAM.lokatie beslaan en toch in het programma gebruikt worden. We noemen deze de attributen van een programma. Hieronder vallen: PIA-adressen, Timer-adressen, stacklokaties bij het geïndexte gebruik van de stack en eventueel subroutines in de KIM-monitor.
Alhoewel de gebruiker van een programma de adressen hiervan zelf moet invullen, zijn deze nooit verschuifbaar in een KIM.
Ze worden bepaald door de "bedrading" en worden harde adressen genoemd.

Hoofdstuk 2.

Het coderen van programma's.

Coderen van programma's is een bezigheid, die te vergelijken is met het aan elkaar solderen en bevestigen van een aantal elektronische componenten.
Het levert ook dezelfde soort ervaring op. Als het gecodeerde programma niet werkt, blijken er vaak codeerfouten in te zitten. Minder vaak blijkt, dat de gebruikte componenten (de programma's, die geschreven zijn) fouten vertonen.
Uiteraard zal een beginner erg veel codeerfouten maken en een ervaren codeur bijna niet meer. Voor deze laatste blijven dan uitsluitend de programmafouten over.
De instrukties coderen kan in twee fazen gedaan worden. In de eerste faze wordt van iedere regel (1 regel = 1 instruktie) bepaald hoeveel bytes deze instruktie in beslag neemt en welke numerieke (hexadecimale) waarde overeenkomt met een bepaalde naam (label). Tijdens deze eerste faze wordt dan een lijst gemaakt, waarin de naam staat met de waarde erbij. Links van het programma wordt steeds het adres van de lokatie, die deze instruktie bevat, gezet, gevolgd door een aantal . . voor iedere byte die door de instruktie gebruikt wordt. Zodra een regel een definitie van een label bevat, wordt deze met zijn waarde overgenomen op de lijst. In de tweede faze wordt de inhoud van iedere byte ingevuld. Hier vinden we op iedere te coderen regel een combinatie van getallen en namen, die alle zorgen voor één of meer bytes in de instruktie.
Op een regel tekst kunnen de volgende items voorkomen:

1. Commentaar. Een regel, die uitsluitend commentaar bevat, levert geen bytes geheugen-inhoud op en dient alleen ter verduidelijking van een programma.
   Een commentaar-regel begint altijd met ; zoals:

```
; DIT IS COMMENTAAR
; LDA #8 IS OOK COMMENTAAR
;
```

   Bij het coderen worden regels met commentaar totaal genegeerd.
2. Definities. Deze geven adressen van interne en externe variabelen en van attributen aan. Ze bestaan uit een "woord", dat begint met een letter en verder bestaat uit maximaal nog vijf letters en/of cijfers, gevolgd door een =, gevolgd door een getal (decimaal of hexadecimaal) of een expressie. Een expressie bestaat uit getallen en labels, waarvan de waarde al bekend is, gescheiden door + (optellen),-(aftrekken), * (vermenigvuldigen) of/(delen).
   Voorbeeld:

```
A=10
B=$10
C=A+B-3
D=C*A/2
```

De label A krijgt de waarde $000A
" " B " " " $0010
" " C " " " $0017
" " D " " " $0073

Alle berekeningen moeten bij voorkeur binair uitgevoerd worden in 16 bits. Alles, wat bij optellen en vermenigvuldigen in het zeventiende bit of verder terecht komt, wordt vergeten.
Bij aftrekken mag altijd vanuit het zeventiende bit geleend worden.
Bij delen wordt de rest van de deling vergeten.
De berekening wordt altijd van links naar rechts uitgevoerd zonder prioriteiten van operaties.
Een tweede vorm van definitie is de label, die voor een instruktie staat. Deze label is een naam, die gehecht wordt aan het adres van de eerste byte van de instruktie.
Als deze ontbreekt, is de ruimte van een label blanco. Dus de eigenlijke instruktie begint verder naar rechts, dan het begin van de regel.
Voorbeeld:

```
  BEGIN   LDA  # 8
          STA  TEST
          LDA  # 6
  WEER    SBC  # 3
          BNE  WEER

          BEGIN VAN DE INSTRUKTIE
  BEGIN VAN DE REGEL.
```

De eerste en vierde regel bevatten een label, de overige regels niet.
De vraag is dan: welke waarde hoort bij een dergelijke naam?
Dit kan alleen bepaald worden door van alle voorgaande instukties te tellen hoeveel bytes er in beslag genomen worden en op welk adres iedere instruktie terechtkomt.
Voorbeeld: Neem het hiervoor behandelde programma, neem aan dat het begint op lokatie $2A7, zet bij iedere instruktie op welke lokatie hij zich bevindt en noteer een label met het adres, waar zich dat voordoet. We hebben dan twee papieren.
Het eerste bevat:

```
02A7 . . . .  BEGIN  LDA # 8
02A9 . . . . . .     STA  TEST
02AC . . . .         LDA # 6
02AE . . . .  WEER   SBC # 3
02B0 . . . .         BNE  WEER
```

Het tweede blad bevat:

BEGIN = 02A7

WEER = 02AE

(Voor de tweede instruktie, STA TEST, is een 16-bits adres aangenomen voor test, zodat de instruktie 3 bytes beslaat).

Iedere instruktie heeft nu zijn geheugenadres gekregen en alle labels hebben een waarde.

3. Instrukties. Een instruktie bestaat uit een gedeelte, dat aangeeft wat er moet gebeuren, de opcode, en eventueel waarmee, de operand.
De operand bepaalt van een instruktie, hoeveel bytes deze in beslag neemt.
Als er geen operand is, is het een 1-byte instruktie.
Als de operand een zeropage adres is, (tussen $0000 en $0100) of als immediate adressing (aangegeven door #) toegepast wordt, beslaat de instruktie 2 bytes.
Als de operand een adres groter dan $00FF is, worden 3 bytes in beslag genomen.
Voorbeelden:

PHA	1 Byte
ROL	1 Byte
LDA # 8	2 Bytes
LDA 8	2 Bytes
LDA (8,X)	2 Bytes
LDA $300	3 Bytes
LDA $300,Y	3 Bytes

4. Dataomschrijvingen. Deze zien eruit als opcodes, maar zijn geen instrukties.
Ze dienen uitsluitend om lokaties te vullen met data. Ze beginnen alle met een .
en kunnen al dan niet voorafgegaan worden door een label.

- .BYTE gevolgd door één of meer getallen en/of expressies, gescheiden door komma's.
  De expressies moeten een resultaat in 8 bits opleveren.
  Voorbeeld:
  ```
  DATA  .BYTE 1,2,3,4+6
  ```
  Dit vult 4 opeenvolgende bytes, waarvan de eerste "DATA" heet.
- .WORD gevold door één of meer getallen en/of expressies, gescheiden door komma's.
  De expressies en getallen vullen ieder 2 bytes. De laagste 8 bits van het 16 bits resultaat komen in de eerste (laagste adres) en de hoogste 8 bits komen in de tweede byte.
  Voorbeeld:
  ```
  DATA  .WORD  1,2,$7F00
  ```
  vult in totaal 6 Bytes.
- .ASCII gevolgd door een teken, dat als afsluiting gebruikt wordt, gevolgd door enige tekst (willekeurige tekens), gevold door het afsluitteken.

Ieder van de tekens tussen de afsluittekens vult 1 lokatie.

Voorbeeld:

```
DATA  .ASCII   *AAP NOOT MIES*
```

vult 13 opeenvolgende lokaties beginnende met DATA.

- .END Dit is een speciale manier om het einde van een te coderen programma aan te geven. Hierna volgt niets meer.

In de tweede faze volgt het eigenlijke coderen.
Hierbij wordt niet gekeken naar definities van labels. Deze moeten volledig bekend zijn. Mocht in de tweede faze een naam opduiken, die noch in de lijst van instrukties voorkomt, noch op het tijdens de eerste faze gemaakte papier, noch één van de dataomschrijvingen zijn, dan is dit een fout in het oorspronkelijke programma.
Het is niet volledig.
Als opcode moet ten eerste gekeken worden naar de naam van de opcode (links op het instruktiekaartje), zoals LDA en ten tweede naar de addressingmode. De addressingmode is te zien aan de operand.
De operand in zijn meest algemene vorm bestaat uit een expressie. Deze vormt na uitrekenen een getal. Dit getal kan 8 of 16 bits zijn. Als de instruktie immediate addressing heeft,mag het antwoord uitsluitend 8 bits zijn.
Komen we hier op meer dan 8 bits, dan is dat een fout in het programma.
Twee speciale tekens worden bij immediate addressing gebruikt om aan te geven of de linker 8 of de rechter 8 bits van een 16 bits getal genomen moet worden nl:
< en >

```
   LDA    #<$7F01   betekent:  LDA # 01
   LDA    #>$7F01        "      LDA # $7F
```

Voorbeeld van een programma, dat gecodeerd wordt

```
; - - - SUBROUTINE OM X GETALLEN BIJ ELKAAR OP TE TELLEN
;
;       BIJ AANROEP STAAT X (>0) IN A, TERWIJL X EN Y SAMEN MET ADRES VAN DE
;       EERSTE LOKATIE BEVATTEN (X HOOGSTE (8BITS)
;
;       BIJ TERUGKEER BEVAT A DE SOM VAN DE GETALLEN
;
;       X EN Y WORDEN HERSTELD
;
; - - - INTERNE VARIABELEN
 ADRES = ....                  2 LOKATIES OP PAGE ZERO
;
;
```

```
; - - -  GEEN EXTERNE VARIABELEN
;
 ADDX   STX    ADRES           ZET VOLLEDIG ADRES
        STX    ADRES + 1       OP PAGE ZERO
        TAY                    Y = INDEX EN TELLER
        LDA    #0              DE BEGINSOM IS 0
        CLC
        DEY                    Y IS 1 TE HOOG
 WEER   ADC    (ADRES),Y       TEL VOLGENDE ERBIJ OP
        DEY                    AL KLAAR?
        BPL    WEER            BRANCH = NEE
        LDY    ADRES+1         JA, HERSTEL Y
        RTS                    KEER TERUG
       .END
```

Dit programma moet gecodeerd worden vanaf lokatie $0280. Op page zero mogen vanaf $C3 lokaties gebruikt worden. Na afloop van faze 1 van het coderen, hebben we een papier, dat het volgende bevat:

```
                ; - - - SUBROUTINE OM X GETALLEN BIJ ELKAAR OP TE TELLEN
                ;
                ;       BIJ AANROEP STAAT X (>0) IN A? TERWIJL X EN Y SAMEN
                ;       HET ADRES VAN DE EERSTE LOKATIE BEVATTEN.
                ;       (X HOOGSTE 8 BITS)
                ;
                ;       BIJ TERUGKEER BEVAT A DE SOM VAN DE GETALLEN
                ;
                ;       X EN Y WORDEN HERSTELD
                ;
                ; - - - INTERNE VARIABELEN
                 ADRES = $C3                2 LOKATIES OP PAGE ZERO
                ;
                ; - - - GEEN EXTERNE VARIABELEN
                ;
0280 . . . .  ADDX      STY     ADRES       . ZET VOLLEDIG ADRES
0282 . . . .            STX     ADRES+1      OP PAGE ZERO
0284 . .                TAY                  Y = INDEX EN TELLER
0285 . . . .            LDA     # 0          DE BEGINSOM IS 0
0287 . .                CLC
0288 . .                DEY                  Y is 1 te hoog
0289 . . . .  WEER      ADC     (ADRES),Y    TEL VOLGENDE ERBIJ OP
```

```
028B . .          DEY                 AL KLAAR?
028C . . . .      BPL     WEER        BRANCH = NEE
028E . . . .      LDY     ADRES+1     JA, HERSTEL Y
02C0 . .          RTS                 KEER TERUG
                  .END
```

Het bladpapier met de labels bevat:

ADRES = 00C3

ADDX = 0280

WEER = 0289

Na de tweede faze van het coderen ziet ons papier er als volgt uit:

```
                  ; - - -  SUBROUTINE OM X GETALLEN BIJ ELKAAR OP TE TELLEN
                  ;
                  ;        BIJ AANROEP STAAT X(>0) in A, TERWIJL X EN Y SAMEN
                  ;        HET ADRES VAN DE EERSTE LOKATIE BEVATTEN
                  ;        (X HOOGSTE 8 BITS)
                  ;
                  ;        BIJ TERUGKEER BEVAT A DE SOM VAN DE GETALLEN
                  ;
                  ;        X EN Y WORDEN HERSTELD
                  ;
                  ; - - -  INTERNE VARIABELEN
                   ADRES = $C3              2 LOKATIES OP PAGE ZERO
                  ;
                  ; - - -  GEEN EXTERNE VARIABELEN
                  ;
0280  84C3  ADDX  STY   ADRES       ZET   VOLLEDIG   ADRES
0282  86C4        STX   ADRES+1     OP PAGE ZERO
0284  A8          TAY               Y = INDEX EN TELLER
0285  A900        LDA   # 0         DE BEGINSOM IS 0
0287  18          CLC
0288  88          DEY               Y IS 1 TE HOOG
0289  71C3  WEER  ADC   (ADRES),Y   TEL VOLGENDE ERBIJ OP
028B  88          DEY               AL KLAAR?
028C  10FB        BPL   WEER        BRANCH = NEE
028E  A4C3        LDY   ADRES+1     JA, HERSTEL Y
02C0  60          RTS               KEER TERUG
                 .END
```

Het coderen heeft tot resultaat, dat we nu de inhoud van de lokatie $0280 tot $02C1 weten. Als deze getallen in het Kim-geheugen ingevuld worden, zit de subroutine "ADDX" in de computer en kan aangeroepen worden.
De ervaring leert, dat de moeilijkst te coderen instrukties de BRANCH-instrukties zijn. Hierbij worden de meeste vergissingen gemaakt.
Allereerst is het van belang te weten dat een branch-instruktie een adresgedeelte heeft, dat de afstand aangeeft in bytes vanaf de lokatie, die volgt op de branch tot aan de lokatie, waarnaartoe gesprongen moet worden.
In het volgende voorbeeld:

```
EEN    LDA   X
TWEE   BPL   VERDER
DRIE   LDA   # 3
VERDER STA   Y
```

moet de afstand tussen de lokatie genaamd "DRIE" en de lokatie "VERDER" bepaald worden. Deze afstand is 2 bytes.
De branch-instruktie wordt dus gecodeerd als:

```
1002  TWEE   BPL   VERDER
```

Merk op, dat de codering volledig onafhankelijk is van de plaats waar het programma staat. Aangezien de branch-instruktie niet alleen vooruit, maar ook achteruit kan, nog een voorbeeld:

```
X = $200
EEN    LDA   X
TWEE   BPL   EEN
DRIE   LDA   # 3
```

Evenals bij het vorige voorbeeld, wordt ook hier de afstand gerekend vanaf de lokatie die volgt op de branch, rekenen we vanaf de lokatie "DRIE".
De afstand is nu vanaf "DRIE" tot aan "EEN". Hiertussen liggen 5 bytes, 2 van BPL EEN en 3 van LDA X. Aangezien terug i.p.v. vooruit gegaan wordt, moet dit getal 5 negatief gemaakt worden in 8 bits, dus 0-5=FB.
De codering wordt nu:

```
10FB   TWEE    BPL    EEN
```

De simpelste wijze om de afstand tussen twee lokaties uit te rekenen is het tellen. We tellen de bytes die in faze 1 van het coderen al op papier gezet worden en als de branch terugwaarts is, maken we dit getal negatief. In de praktijk blijkt, dat alhoewel dit tellen vrij veel werk is, het de zekerste resultaten oplevert.

Hoofdstuk 3.

Logische besturingen I.

Logische besturingen kunnen omschreven worden met formules in BOOLEAANSE notatie. Het hierbeschreven programma biedt deze mogelijkheid voor iedere combinatie van 8 input- en 6 outputbits. De inputsignalen moeten aangesloten worden op de A-zijde van de applicatie PIA (pinnen PA0 t.e.m. PA7), terwijl de outputsignalen aangesloten moeten worden op de B-zijde van de applicatie PIA- (pinnen PB0 t/m PB5).
Het programma werkt als volgt:

- De inputbits worden ingelezen en neergezet op lokatie INPSTA+7. Iedere lokatie bevat $00 als het overeenkomstige bit de waarde 0 heeft en $FF als het bit de waarde 1 heeft. Lokatie INPSTA correspondeert met bit 0, lokatie INPSTA+1 met bit 1, lokatie INPSTA+2 met bit 2 enz. tot lokatie INPSTA+7, die overeenkomt met bit 7.
  Het inlezen gebeurt niet vaker dan eens per 10 millisecondes, zodat kontaktdender van de invoersignalen geen invloed heeft.
- De ingelezen bits worden vergeleken met de stand, die ze de vorige keer hadden en hieruit worden twee tabellen gegenereerd.
  De ene tabel bevat per lokatie $FF als het overeenkomstige bit van 0 naar 1 veranderd is, terwijl de andere tabellen alle veranderingen van 1 naar 0 bevat. Alle lokaties van beide tabellen, waarvan het signaal niet naar de gedefinieerde stand veranderd is, bevatten $00.
  De tabel met de veranderingen van 0 naar 1 staat op lokatie INP01 t.e.m. INP01+7.
  De veranderingen van 1 naar 0 staan in de lokaties INP10 t.e.m. INP10+7.
- Nadat de inputsignalen ingelezen zijn en vergeleken met de vorige stand om de veranderingen te detekteren, wordt een subroutine aangeroepen, waarin de gebruiker zijn geprogrammeerde logica aangebracht heeft. Deze subroutine zal een aantal outputsignalen genereren. Hiervoor zijn 2 tabellen beschikbaar eveneens met 1 lokatie per outputbit. De ene tabel bevat statische uitgangen, d.w.z. als hierin $00 staat, zal het bijbehorende outputbit op 0 gezet worden.
  Staat hierin een getal ongelijk aan $00, dan wordt het bijbehorende outputbit op 1 gezet. De tweede tabel bevat momentele outputkontakten d.w.z.: als in één van de bytes een getal ongelijk aan $00 gezet wordt, wordt het corresponderende outputbit gedurende tenminste 10 milliseconde 1 gemaakt.
  Als het gebruikersprogramma na 10 milliseconde (dus de volgende keer, dat de gebruikerssubroutine aangeroepen wordt) de inhoud van de corresponderende lokatie niet weer ongelijk aan $00 gemaakt heeft, wordt het outputbit op 0 gezet.

Ieder outputsignaal kan aldus als statisch signaal of als puls signaal gebruikt worden.

- Nadat de gebruikerssubroutine teruggekeerd is naar zijn aanroeper, worden de outputtabellen uitgestuurd naar de outputbits van de PIA. Hierbij worden de statische- en de pulstabel in een OR-bewerking samengevoegd.
De tabel met statische uitgangen staat op de lokaties OUTSTA t.e.m. OUTSTA+5, terwijl de pulsuitgangen OUTPLS t.e.m. OUTPLS+5 beslaan.
- Hierna wordt het geheel van voorafaan herhaald.

Als extra faciliteit houdt het programma een aantal timerlokaties bij. Iedere lokatie in deze tabel stelt een timer voor, die eens per seconde afgeteld wordt. Als het gebruikersprogramma dus het getal 100 in een dergelijke lokatie zet, zal de lokatie na 100 secondes nul bevatten. Zodra de lokatie nul geworden is, houdt het aftellen op. Aangezien een logische waarde 0 overeenkomt met $00 in een lokatie en de logische waarde 1 met niet $00 in een lokatie, kan een timer als logische variabele gehanteerd worden.
Een afgelopen timer heeft dan de logische waarde 0, terwijl een lopende timer de waarde 1 heeft. Er zijn 8 timers. Ze lopen vanaf TIMTAB t.e.m. TIMTAB+7.
Met het KIM-instruktiepakket kunnen nu op uiterst eenvoudige wijze logische bewerkingen gedaan worden.
De volgende lijst is hierbij als hulp bedoeld.

1. Inverteren. EOR # $FF

   Voorbeeld: Geef de inverse stand van inputbit 3 door aan outputbit 1.

```
LDA   INPSTA+3
EOR   #$FF
STA   OUTSTA+1
```

2. Logisch EN LDA . . . .
   AND . . . .

   Voorbeeld: Geef de inverse logische EN van inputbit 2 en 3 door aan outputbit 1.

   formule: $\emptyset_1 = \overline{I_2 . I_3}$

   programma:

```
LDA   INPSTA+2
AND   INPSTA+3
EOR   #$FF
STA   OUTSTA+1
```

3. Logische OF. LDA . . . .
   ORA . . . .

   Voorbeeld:

   Formule: O1=I2.I3+I1

```
Programma:   LDA  INPSTA+2
             AND  INPSTA+3
             ORA  INPSTA+1
             STA  OUTSTA+1
```

4. Hulpvariabelen. Als hulpvariabele kan iedere lokatie in het RAM-geheugen gebruikt worden.

Voorbeeld:

Formule: $O1=I2.I3+I4.(I0+I1)+I5.I6+\overline{I7}$

```
Programma:  HULP= . . . .
            LDA    INPSTA+2
            AND    INPSTA+3
            STA    HULP
            LDA    INPSTA+1
            ORA    INPSTA
            AND    INPSTA+4
            ORA    HULP
            STA    HULP
            LDA    INPSTA+6
            AND    INPSTA+5
            ORA    HULP
            STA    HULP
            LDA    INPSTA+7
            EOR    #$FF
            ORA    HULP
            STA    OUTSTA+1
```

Speciale aandacht moet beschouwd worden aan "FLIPFLOP-lokaties". Als FLIPFLOP-lokatie geldt een lokatie die gezet wordt als inputbit A van 0 naar 1 gaat en gereset als inputbit B van 0 naar 1 gaat. Deze FLIPFLOP staat aan het begin in een ongedefinieerde toestand. Als dat gewenst is, kan de eerste keer, dat het programma loopt deze FLIPFLOP in een gedefinieerde stand gezet worden. De codering hiervoor is dan:

```
    LDA  #0          ZET HEM OP NUL
    STA FLIPFL
    -OF-
    LDA  #$FF        ZET HEM OP EEN
    STA FLIPFL
```

Deze codering kan ingevoegd worden na de lokatie "ENDINT" en voor de lokatie "OUTDO".

5. SR-flipflop. Hiervoor kan een statisch outputbit of een hulpvariabele gebruikt worden.

Voorbeeld: Outputbit 3 moet gezet worden, als inputbit 1 van 1 naar 0 gaat en gereset als inputbit 2 van 0 naar 1 gaat.

Waarheidstabel:

I1	I2	O3
0	0	X
0	1	X
1	0	X
1	1	X
↑	X	X
↓	X	1
X	↑	0
X	↓	X

```
Programma: LDA   INP10+1
           ORA   OUTSTA+3
           STA   OUTSTA+3
           LDA   INPO1+2
           EOR   #$FF
           AND   OUTSTA+3
           STA   OUTSTA+3
```

6. D-type flipflop.

Voorbeeld: Outputbit 4 moet geset worden overeenkomstig hulpvariabele "DUM", waarbij inputbit 1 als kloksignaal fungeert (0 → 1).

Programma:

```
           DUM = . . . .
                 |
                 |
           LDA   INPO1+1
           BEQ   NIET
           LDA   DUM
           STA   OUTSTA+4
    NIET   . . . . . . .
```

7. JK flipflop.

Voorbeeld: Outputbit 5 moet geset worden als hulpvariabele "DUM" 1 is en gereset worden als inputbit 0 1 is, waarbij inputbit 2 als klok fungeert (0 → 1)

```
Programma:DUM = . . . .
          |
          |
          LDA  INPO1+2
          BEQ  NIET
          LDA  DUM
          ORA  OUTSTA+5

          STA  OUTSTA+5
          LDA  INPSTA
          EOR  #$FF
          AND  OUTSTA+5
          STA  OUTSTA+5
   NIET   . . . . . .
```

8. Binaire deler.

   Voorbeeld: Outputbit 0 moet iedere keer als inputbit 3 van 0 naar 1 gaat omklappen.

```
Programma:  LDA  INPSTA+3
            BEQ  NIET
            LDA  OUTSTA
            EOR  #$FF
            STA  OUTSTA
    NIET    . . . . . .
```

9. Decimale deler.

   Voorbeeld: Outputbit 2 moet iedere keer als inputbit 1 voor de tiende keer van 0 naar 1 gegaan is een puls afgeven.

```
Programma:      HULP = . . . .   HULPVARIABELE
                  |
                  |
                  LDA   INPO1+1
                  BEQ   NOINC
                  INC   HULP
                  LDA   HULP
                  CMP   #10   WE TELLEN TOT 10
                  BNE   NOINC
                  LDA   #$FF
                  STA   OUTPLS
                  LDA   #0
                  STA   HULP
        NOINC     . . . . .
```

De variabele "HULP" moet de allereerste keer op 0 gezet worden.
Deze deler is uiteraard niet beperkt tot 10. De instruktie CMP-# 10 kan variëren tussen CMP #1 en CMP #255.

10. Integrating oneshot.
Voorbeeld: Zodra inputbit 0 van 0 naar 1 gaat, moet outputbit 3 1 worden en dit gedurende 20 secondes blijven.

```
Programma:   LDA  INP01
             BEQ  NOSTRT
             LDA  #20
             STA  TIMTAB
NOSTRT       LDA  TIMTAB
             STA  OUTSTA+3
```

11. Non integrating one shot.
Voorbeeld: Zodra inputbit 0 van 0 naar 1 gaat, moet outputbit 3 1 worden en dit gedurende 20 secondes blijven. Komt het ingangssignaal, terwijl het uitgangssignaal 1 is, dan wordt het ingangssignaal genegeerd.

```
Programma:   LDA    OUTSTA+3
             BNE    IGNORE
             LDA    INP01
             BEQ    IGNORE
             LDA    #20
             STA    TIMTAB
IGNORE       LDA    TIMTAB
             STA    OUTSTA
```

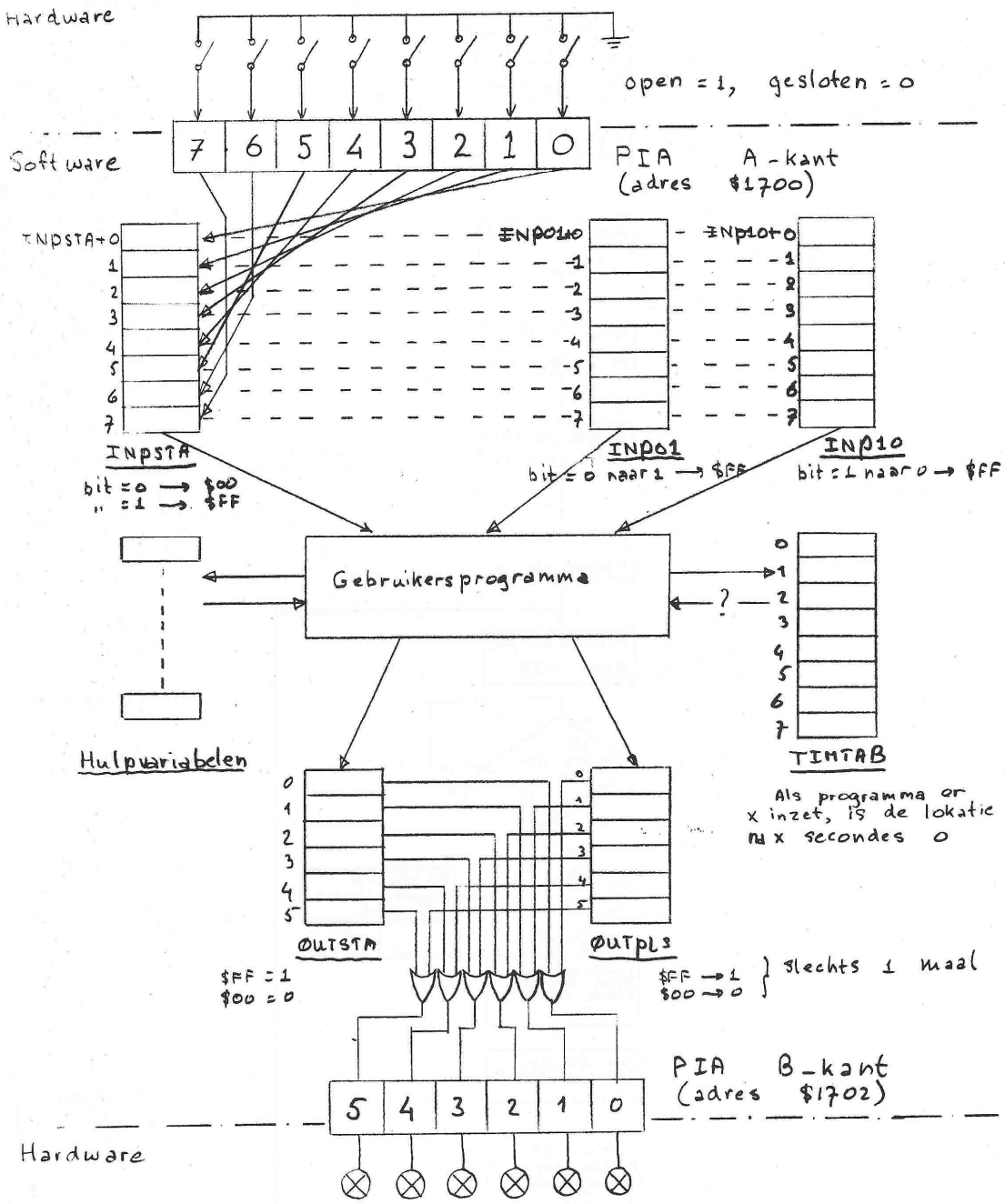
Enige algemene opmerkingen over de logische bewerkingen.

- Alhoewel de computer redelijk snel is, zal een logische schakeling, die geprogrammeerd wordt, vele factoren trager zijn dan zijn equivalent in hardware. Dit kan soms speciale problemen met timing opwerpen.
  De langste te verwachten reactietijd op een ingangssignaal bedraagt 10240 microsecondes.
  Deze tijd wordt bepaald met behulp van de hardwaretimer van de KIM.
- Het zal blijken, dat niet iedere seconde, die met behulp van dit programma gecreëerd wordt even lang is. De fout is niet accumulerend. De fout is maximaal +10243 microseconde. Hierbij wordt ervan uitgegaan, dat het gebruikersprogramma niet zolang is, dat het meer dan 10 millisecondes kost om in zijn geheel éénmaal doorlopen te worden.

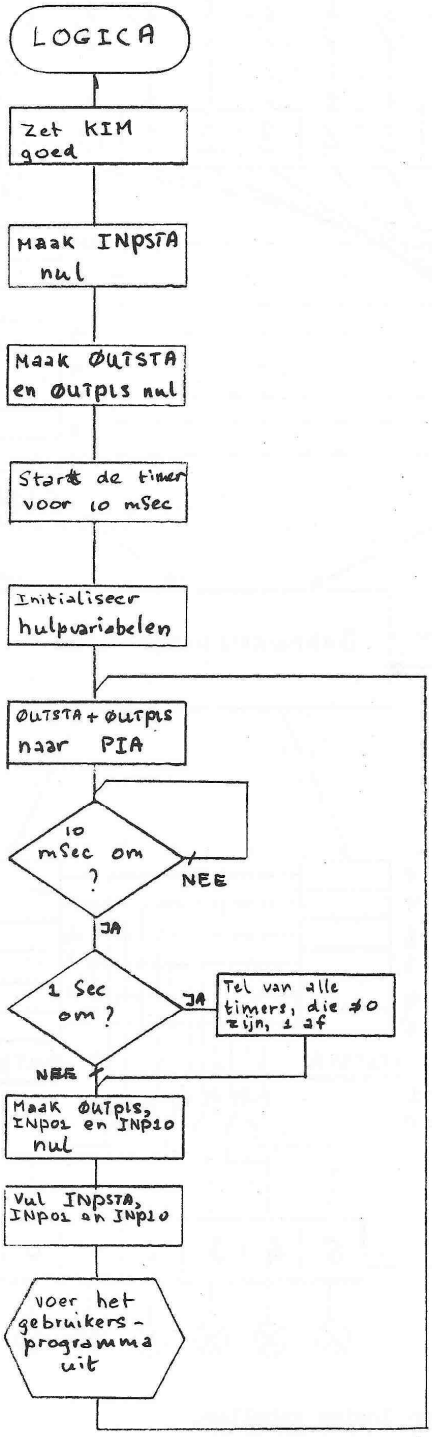
Teneinde een schatting te maken van de tijdsduur van een gebruikersprogramma, kan aangenomen worden, dat 1 instruktie gemiddeld 3 microsecondes duurt. In 10 millisecondes kunnen dus totaal ongeveer 10000:3=3333 instrukties doorlopen worden.
In het geval, dat twijfel rijst over de timing, kan het programma zodanig gewijzigd worden, dat het zelf test of het gebruikersprogramma binnen de vastgestelde tijd gereed komt en stopt zodra dit niet het geval is.
De volgende instrukties moeten dan geplaatst worden tussen "TOTO" en "WACHT".

```
LDA  TIMER   AL UITGETIMED?
BPL  WACHT   BRANCH=NEE
JMP  $1C00   JA, STOP MAAR
```

Als deze instrukties geplaatst zijn, is het niet meer mogelijk om single-instruktie door het programma heen te lopen.



Overzicht van logica tabellen.



Stroomdiagram programma logica.

```
; . . . . PROGRAMMA LOGICA
;
; . . . . EXTERNE VARIABELEN
;
INPSTA = . . . .                    STATISCHE INGANGSSIGNALEN
INPO1 = . . . .                     VERANDERINGEN VAN 0 ⟶ 1
INP10 = . . . .                          "    "    "  1 ⟶ 0
OUTSTA= . . . .                     STATISCHE UITGANGSSIGNALEN
OUTPLS= . . . .                     PULSUITGANGEN
TIMTAB= . . . .                     TABEL MET TIMERS
;
; . . . . INTERNE VARIABELEN
MASK = . . . .                      GEBRUIKT VOOR BITPOSITIES
TIME1= . . . .                      2 LOKATIES OM STEEDS
TIME2= . . . .                      1 SEKONDE AF TE TELLEN
;
; . . . . ATTRIBUTEN
PIA = $1700                         ADRES VAN APPLICATIE PIA
OUTPIA = PIA+2                      B-KANT
INPWRD=PIA                          A-KANT
TIMER=$1747                         1024 MICRO SECONDE TIMER
;
; . . . . DE NU VOLGENDE DEFINITIE IS HET STARTADRES WAAR DE USER SUBROUTINE
;         BEGINT
USER = . . . .
;
; . . . . BEGIN VAN HET PROGRAMMA
;
;         EERST ALLES GOED ZETTEN
;
START SEI
      CLD
      LDX   #$FF
      TXS
      LDA   #0                      DE NMI-VECTOR INVULLEN
      STA   $17FA
      LDA   #$1C
      STA   $17FB
      LDA   #0                      DE PIA
```

```
      STA   PIA+1                       A-KANT=INPUT
      STA   OUTPIA
      LDA   #$3F                        B-KANT=OUTPUT
      STA   PIA+3
; . . . . DE EERSTE KEER ALLE OUTPUTS 0, EN ALLE INPUTS 0
      LDX   #7
      LDA   #0
WRINT STA   INPSTA,X    INPUT  NUL
      CPX   #6
      BPL   NONT
      STA   OUTSTA,X    OUTPUT NUL
      STA   OUTPLS,X    PULS  OUTPUT  NUL
NONT  STA   TIMTAB,X    TIMERS  NUL
      DEX               AL KLAAR?
      BPL   WRINT       BRANCH=NEE
      LDA   #0          DE TIJD BEGINT OP
      STA   TIME1       0
      STA   TIME2
      LDA   #10         START DE TIMER
ENDINT STA  TIMER
; . . . . DOE ALLE ECHTE OUTPUT
OUTDO LDX   #5          ER ZIJN 6 OUTPUTBITS
      LDA   #$20        BEGINNEN MET BIT 5
      STA   MASK
      LDY   #0          Y is OUTPUT WAARDE
TSTOUT LDA  OUTSTA,X    ALS OUTSTA OF OUTPLS
      ORA   OUTPLS,X    NIET 0 IS, MOET DIT
      BEQ   NOTOUT      BIT GEZET
      TYA
      ORA   MASK
      TAY
NOTOUT LSR  MASK
      DEX               AL KLAAR?
      BPL  TSTOUT       BRANCH=NEE
TOTO  STY   OUTPIA      JA, NAAR UITGANGSSIGNAAL
; . . . . WACHT TOT DE 10 MILLISECONDES OM ZIJN
WACHT LDA   TIMER       TIMER KLAAR?
      BPL   WACHT       BRANCH=NEE
      LDA   #10         JA, PRESET HEM MAAR
      STA   TIMER       WEER VOOR ONGEVEER 10 MSC.
```

```
; . . . . BEREKEN DE SECONDE-TICK
         CLC
         CLD
         LDA   TIME1                  EERST TIENTALLEN
         ADC   #24                    MICROSECONDES
         STA   TIME1
         LDA   TIME2
         ADC   #1
         STA   TIME2
         CMP   #100                   AL 1 SECONDE OM?
         BMI   NOTYET
; . . . . ER IS 1 SECONDE OM. TEL DE TELLERS AF
SECOM    SEC
         SBC   #100
         STA   TIME?
         LDX   #7                     ER ZIJN 8 TIMERS
TIMWER   LDA   TIMTAB,X               IS DEZE NUL?
         BEQ   NOTIM                  BRANCH=JA
         DEC   TIMTAB,X               NEE, TEL AF
NOTIM    DEX
SECOUT   BPL   TIMWER
; . . . . MAAK ALLES NUL, WAT NUL GEMAAKT MOET WORDEN
NOTYET   LDX   #7                     8BITS/8LOKATIES
         LDA   #0
WECLER   STA   INPO1,X                ALLE VERANDERINGEN
         STA   INP10,X
         CPX   #6                     PULS OUTPUTBIT?
         BPL   NOCOUT                 BRANCH=NEE
         STA   OUTPLS,X               JA, PULS OUTPUT OOK
NOCOUT   DEX                          ALLES GEHAD?
         BPL   WECLER                 BRANCH=NEE
; . . . . DOE NU DE INPUTS EN DE VERANDERINGEN
DOEINP   LDX   #7                     8 INPUTBITS
         LDA   #$80                   BITPOSITIE
         STA   MASK
         LDA   INPWRD                 NEEM DE PIA-STAND
WERINP   PHA                          BEWAAR HEM
         LDY   #0                     Y wordt 0 OF $FF
         AND   MASK                   IS HET BITJE 1
```

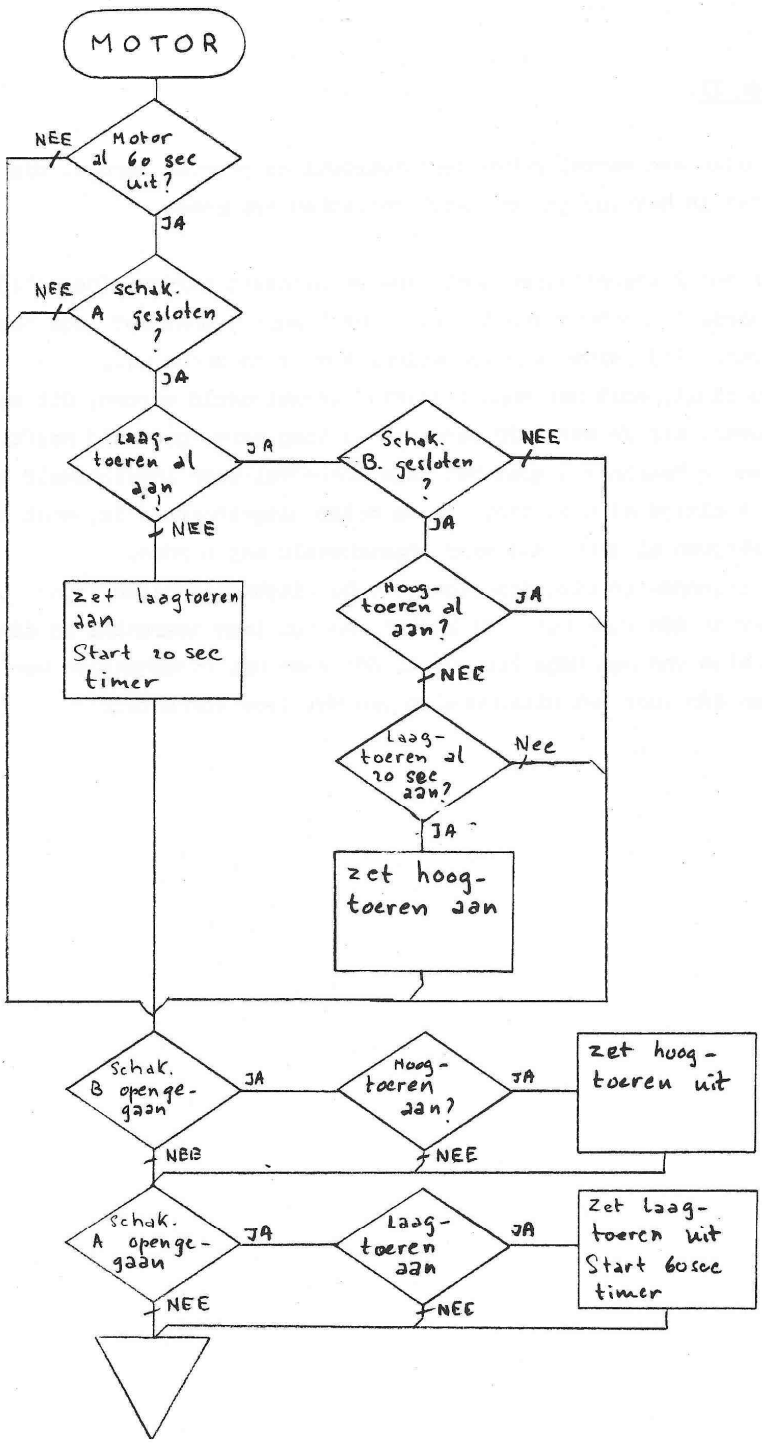
```
        BEQ   STINPS          BRANCH=NEE
        LDY   #$FF            JA, DUS $FF
STINPS  TYA
        EOR   INPSTA,X        IS HET BIT VERANDERD?
        BEQ   NOCHNG          BRANCH=NEE
        CPY   #0              JA, VAN 0 naar 1
        BEQ   TIS10           BRANCH=NEE
        STA   INP01,X         JA, NOTEER DIT IN DE
        LDA   #0              ENE OF IN DE ANDERE
TIS10   STA   INP10,X         TABEL
        STY   INPSTA,X        EN ZET DE NIEUWE STATUS
        PLA                   HAAL INPUT OP
        LSR   MASK            VOLGENDE BIT
        DEX                   AL KLAAR?
        BPL   WERINP          BRANCH=NEE
        JSR   USER            JA, GA NAAR USER
        JMP   OUTDO           OPNIEUW
        .END
```

Hoofdstuk 4.

Logische besturingen II.

In dit hoofdstuk zullen een aantal gebruikers-subroutines gegeven worden, die gebruik maken van het in het vorige hoofdstuk besproken programma.
LOGICA.

A. Een electromotor met 2 toerentallen moet aan- en uitgezet worden afhankelijk van 2 niveauschakelaars. Als schakelaar A sluit (dus 0 wordt) moet het lage toerental ingeschakeld worden. Bij openen van schakelaar A moet de motor uit.
Als schakelaar B sluit, moet het hoge toerental ingeschakeld worden. Dit mag echter pas gebeuren, als de motor 20 secondes op laagtoeren gedraaid heeft.
Bij het openen van schakelaar B moet het lage toerental weer ingeschakeld worden.
Als B sluit, is A altijd al gesloten. Als de motor uitgeschakeld is, moet hij 60 secondes uitblijven alvorens hij weer ingeschakeld mag worden.
We hebben dus 2 ingangs- en uitgangs-signalen. De uitgangssignalen zijn pulskontakten. Er is één voor het inschakelen van het lage toerental en één voor het inschakelen van het hoge toerental, één voor het uitschakelen van het hoge toerental en één voor het uitschakelen van het lage toerental.


MOTOR
Motor al 60 sec uit?
NEE
JA
schak. A gesloten ?
NEE
JA
laag-toeren al aan?
JA
NEE
Schak. B. gesloten ?
NEE
JA
zet laagtoeren aan
Start 20 sec timer
Hoog-toeren al aan?
JA
NEE
Laag-toeren al 20 sec aan?
Nee
JA
zet hoog-toeren aan
Schak. B opengegaan
JA
NEE
Hoog-toeren aan?
JA
NEE
zet hoog-toeren uit
Schak. A opengegaan
JA
NEE
Laag-toeren aan
JA
NEE
Zet laag-toeren uit
Start 60sec timer

```
; . . . . SUBROUTINE MOTOR
;
; . . . . DEZE SUBROUTINE SCHAKELT HOOG EN LAAG TOERENTAL VAN EEN
;        MOTOR VOLGENS SPECIFICATIES
;
; . . . . EXTERNE VARIABELEN
SCHAKA = INPSTA                          SCHAKELAAR A
SCHAKB = INPSTA+1                            "      B
SCHAUI = INPO1                               "      A  OPENGEGAAN
SCHBUI = INPO1+1                             "      B   "     "
;        SCHAKELAAR A IS AANGESLOTEN OP BIT 0 VAN DE PIA, TERWIJL
;        SCHAKELAAR B OP BIT 1 ZIT
LGAAN = OUTPLS                           LAAG TOEREN AAN
LGUIT = OUTPLS+1                          "     "   UIT
HGAAN = OUTPLS+2                         HOOG   "   AAN
HGUIT = OUTPLS+3                          "     "   UIT
;
;        LAAG TOEREN AAN IS BIT 0  PIA = B
;         "    "     UIT "   "  1   "    "
;        HOOG TOEREN AAN "   "  2   "    "
;         "    "     UIT "   "  3   "    "
;
UITDEL = TIMTAB                          DELAY NA UITSCHAKELEN
AANDEL = TIMTAB+1                         "     "  INSCHAKELEN
;
; . . . . INTERNE VARIABELEN
LAAG = . . . .                           LAAG TOEREN IS AAN OF UIT
HOOG=  . . . .                           HOOG TOEREN IS AAN OF UIT
;
; . . . . BEGIN VAN HET PROGRAMMA
MOTOR    LDA   UITDEL                    MOTOR AL 60 SEC UIT?
         BNE   TWEE                      BRANCH=NEE
         LDA   SCHAKA                    JA, SCHAKELAAR A GESLOTEN?
         BNE   TWEE                      BRANCH=NEE
         LDA   LAAG                      JA, LAAGTOEREN AL AAN?
         BNE   DRIE                      BRANCH=JA
         LDA   #$FF                      NEE, ZET LAAGTOEREN AAN
         STA   LAAG
         STA   LGAAN
```

```
        LDA   #20                 START 20 SEC TIMER
        STA   AANDEL
        IMP   TWEE
DRIE    LDA   SCHAKB              SCHAKELAAR B GESLOTEN?
        BNE   TWEE                BRANCH=NEE
        LDA   HOOG                JA, HOOGTOEREN AL AAN?
        BNE   TWEE                BRANCH=JA
        LDA   AANDEL              NEE, LAAG AL 20 SEC AAN?
        BNE   TWEE                BRANCH=NEE
        LDA   #$FF                JA, ZET HOOGTOEREN AAN
        STA   HOOG
        STA   HGAAN
TWEE    LDA   SCHBUI              SCHAKELAAR B OPENGEGAAN?
        BEQ   VIER                BRANCH=NEE
        LDA   HOOG                JA, HOOGTOEREN AAN?
        BEQ   VIER                BRANCH=NEE
        LDA   #0                  JA, ZET HOOGTOEREN UIT
        STA   HOOG
        LDA   #$FF
        STA   HGUIT
VIER    LDA   SCHAUI              SCHAKELAAR A OPENGEGAAN?
        BEQ   KLAAR               BRANCH=NEE
        LDA   LAAG                JA, LAAGTOEREN AAN?
        BEQ   KLAAR               BRANCH=NEE
        LDA   #$FF                JA, ZET LAAGTOEREN UIT
        STA   LGUIT
        LDA   #60                 START 60 SECONDE TIMER
        STA   UITDEL
KLAAR   RTS                       KEER TERUG NAAR AANROEPER
        .END
```

In het initialiseringsgedeelte van LOGICA moeten de hulpvariabelen "HOOG" en "LAAG" op nul gezet worden met de volgende coding:

```
        LDA   #0
        STA   LAAG
        STA   HOOG
```

8. Ter beveiliging van een brandkast wordt met behulp van een KIM een elektronisch cijferslot gemaakt. Het cijferslot bestaat uit 4 drukknopjes, waarop een combinatie ingesteld kan worden door één of meer malen op de knopjes te drukken. Verder is er nog een drukknop voor het openen en één voor het sluiten van de brandkast.

Er is een hersteldrukknop voor het geval tijdens het indrukken van een combinatie een vergissing gemaakt is en een drukknop om het alarm af te zetten.

Er is een uitgangssignaal voor het openen en sluiten van de brandkast en één voor het alarm.

Als de brandkast open is, kan hij gesloten worden door op de vier drukknoppen van het geheime getal een code in te drukken.

Dit gebeurt, door ieder van de knopjes tenminste éénmaal in te drukken.

Maximaal mag een knopje 255 maal ingedrukt worden.

Na het intoetsen van de combinatie moet de knop sluiten ingedrukt worden.

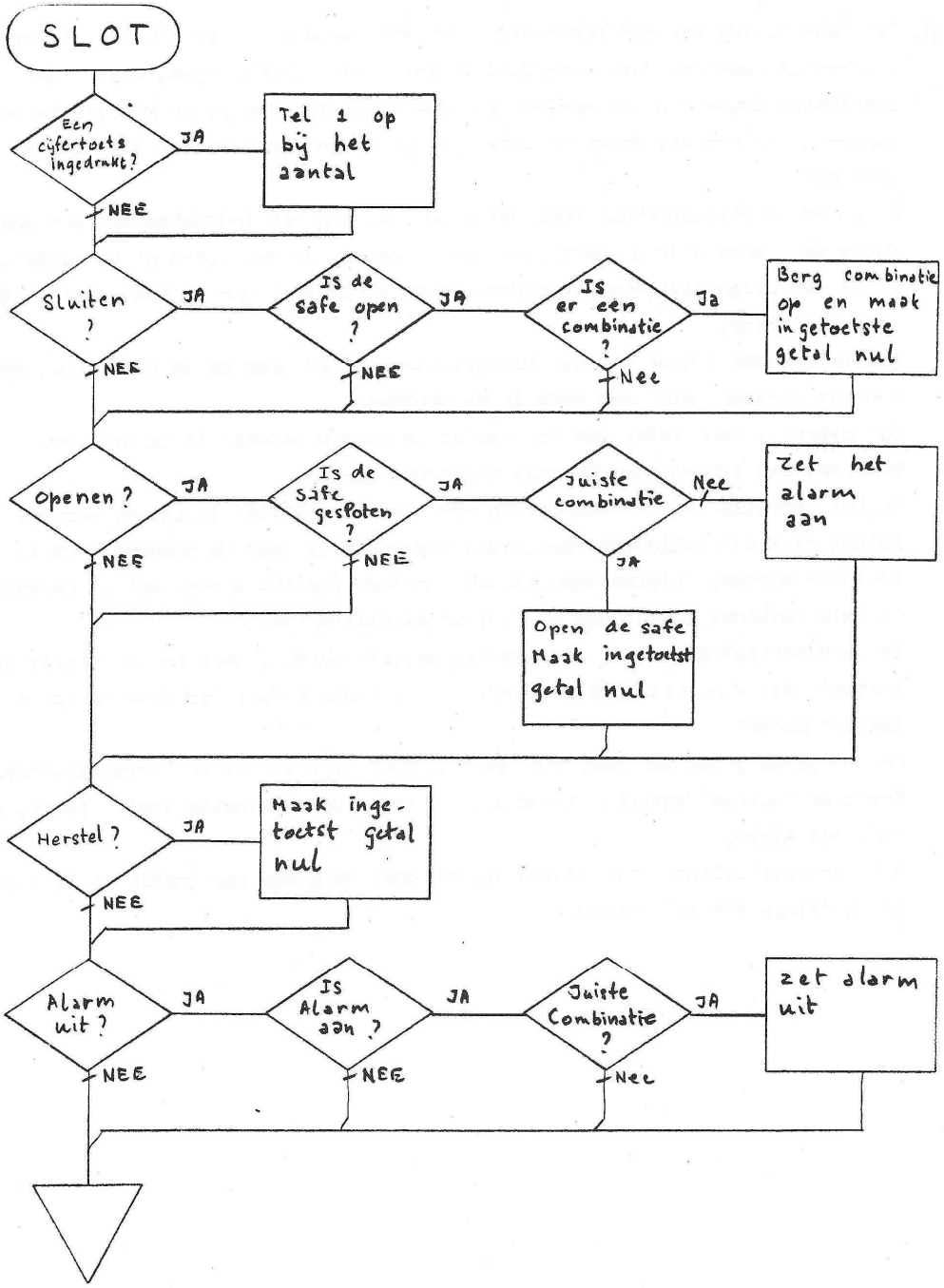
Zolang niet een volledige combinatie ingesteld is, kan de brandkast niet gesloten worden. Vergissingen kunnen ongedaan gemaakt worden met de herstelknop. De hele combinatie moet dan opnieuw ingetoetst worden.

Is de brandkast gesloten, dan kan hij geopend worden, door de combinatie in te toetsen, die voor het sluiten gebruikt is, gevolgd door het drukken op de toets "openen".

Een vergissing kan ook hier ongedaan gemaakt worden met de "Herstel"-knop.

Wordt op "openen" gedrukt, terwijl een verkeerde combinatie ingesteld is, dan gaat het alarm.

Dit kan uitsluitend stil gemaakt worden door de goede combinatie in te toetsen en op "Alarm af" te drukken.


SLOT
Een cijfertoets ingedrukt?
JA
Tel 1 op bij het aantal
NEE
Sluiten ?
JA
Is de safe open ?
JA
Is er een combinatie ?
Ja
Berg combinatie op en maak ingetoetste getal nul
NEE
NEE
Nee
Openen ?
JA
Is de safe gesloten ?
JA
Juiste combinatie
Nee
Zet het alarm aan
NEE
NEE
JA
Open de safe Maak ingetoetst getal nul
Herstel ?
JA
Maak ingetoetst getal nul
NEE
Alarm uit ?
JA
Is Alarm aan ?
JA
Juiste Combinatie ?
JA
Zet alarm uit
NEE
NEE
Nee

```
; . . . . SUBROUTINE SLOT
;
;     DEZE SUBROUTINE VORMT SAMEN MET HET PROGRAMMA "LOGICA" EEN  CIJFERSLOT
;
; . . . . EXTERNE VARIABELEN
KNOP1=INP10                          KNOP 1 VOOR COMBINATIE
KNOP2=INP10+1                         "   2   "     "
KNOP3=INP10+2                         "   3   "     "
KNOP4=INP10+3                         "   4   "     "
OPENEN=INP10+4                       OPEN DE SAFE
SLUITIN=INP10+5                      SLUIT DE SAFE
HERSTL=INP10+6                       HERSTEL VERGISSING
ALMUIT=INP10+7                       ZET ALARM UIT
UIT = OUTSTA                         OPEN/SLUIT SAFE
ALRM=OUTSTA+1                        ALARM UITGANG
;
; . . . . INTERNE VARIABELEN
;
COMBI = . . . .                      4 LOKATIES VOOR DE COMBINATIE
INPGET= . . . .                      4     "     "  INGETOETST
;
; . . . . BEGIN PROGRAMMA
;
SLOT    LDA    KNOP1                 IS KNOP 1 INGEDRUKT?
        BEQ    NOT1                  BRANCH=NEE
        INC    INPGET                JA, TEL 1 OP
NOT 1   LDA    KNOP2                 IS KNOP 2 INGEDRUKT?
        BEQ    NOT2                  BRANCH=NEE
        INC    INPGET+1              JA, TEL 1 OP
NOT 2   LDA    KNOP 3                IS KNOP 3 INGEDRUKT?
        BEQ    NOT 3                 BRANCH=NEE
        INC    INPGET+2              JA, TEL 1 OP
NOT 3   LDA    KNOP 4                IS KNOP 4 INGEDRUKT?
        BEQ    NOT4                  BRANCH=NEE
        INC    INPGET+3              JA, TEL 1 OP
NOT 4   LDA    SLUITN                IS "SLUITEN" INGEDRUKT?
        BEQ    NOTSLU                BRANCH=NEE
        LDA    UIT                   JA, IS SAFE OPEN?
        BNE    NOTSLU                BRANCH=NEE
        LDX    #3                    JA, IS ER EEN COMBINATIE?
```

```
TSCOM   LDA   INPGET,X
        BEQ   NOTSLU          BRANCH = NEE
        DEX
        BPL   TSCOM
        LDX   #3              JA, BERG HEM OP
BERGOP  LDA   INPGET,X
        STA   COMBI,X
        LDA   #0              MAAK INGETOETST GETAL
        STA   INPGET,X        NUL
        DEX
        BPL   BERGOP
NOTSLU  LDA   OPENEN          IS OPENEN INGEDRUKT?
        BEQ   NTOPEN          BRANCH=NEE
        LDA   UIT             IS DE SAFE GESLOTEN?
        BEQ   NTOPEN          BRANCH=NEE
        LDX   #3              JA, JUISTE COMBINATIE?
TSGOED  LDA   INPGET,X
        CMP   COMBI,X
        BEQ   MAYBE
        LDA   #$FF            NEE, ZET ALARM AAN
        STA   ALRM
        IMP   NTOPEN
MAYBE   DEX
        BPL   TSGOED
        LDX   #3              JA, MAAK INGETOETST GETAL NUL
        LDA   #0
CLRA    STA   INPGET,X
        DEX
        BPL   CLRA
        LDA   #0              OPEN DE SAFE
        STA   UIT
NTOPEN  LDA   HERSTL          HERSTEL INGEDRUKT?
        BEQ   NTHERS          BRANCH=NEE
        LDX   #3              JA, MAAK INGETOETST GETAL NUL
        LDA   #0
CLRB    STA   INPGET,X
        DEX
        BPL   CLRB
```

```
NTHERS  LDA   ALMUIT                ALARM UIT?
        BEQ   KLAAR                BRANCH=NEE
        LDA   ALRM                 JA, ALARM AAN?
        BEQ   KLAAR                BRANCH=NEE
        LDX   #3                   JA, COMBINATIE OK?
TSKLAR  LDA   INPGET,X
        CMP   COMBI,X
        BNE   KLAAR                BRANCH=NEE
        DEX
        BPL   TSKLAR
        LDA   #0                   Ja, ZET ALARM UIT
        STA   ALRM
KLAAR   RTS                        KEER TERUG
        .END
```

Opmerking: De 4 lokaties "COMBI" en de 4 lokaties "INPGET" moeten bij voorkeur in het begin nul gemaakt worden. Wordt dit niet gedaan, dan nadat het programma gestart is, de HERSTEL-knop eerst indrukken.

C. Een uitgangssignaal moet 1 zijn, zolang 3 ingangssignalen een bepaalde combinatie vertegenwoordigen. Het moet 0 worden bij een andere combinatie. Bij alle andere dan de twee hiervoor genoemde combinaties moet niets aan het uitgangssignaal veranderen.

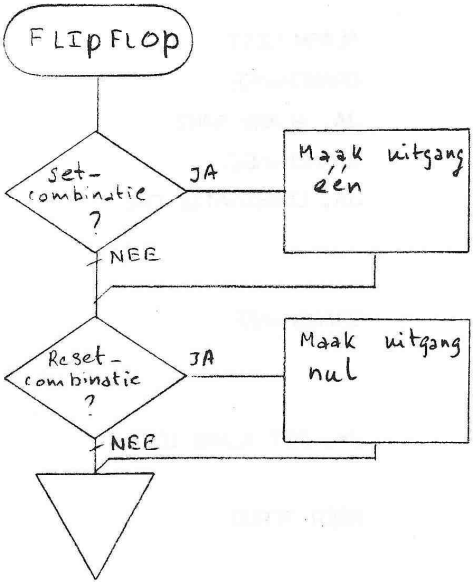
Waarheidstabel:

A	B	C	UIT
0	0	0	X
0	0	1	X
0	1	0	wordt 1
0	1	1	X
1	0	0	X
1	0	1	wordt 0
1	1	0	X
1	1	1	X

Het uitgangssignaal kan beschouwd worden als een flipflop, die geset en gereset wordt.

Formule voor setten: $UIT = \overline{A}.B.\overline{C}$

Formule voor resetten: $UIT = A.\overline{B}.C$




```
; . . . . SUBROUTINE FLIPFLOP
;
; . . . . EXTERNE VARIABELEN
;
INGA=INPSTA                     INGANGSSIGNAAL A
INGB=INPSTA+1                       "    "     B
INGC=INPSTA+2                       "    "     C
UIT=OUTSTA                      UITGANGSSIGNAAL
;
; . . . . BEGIN VAN HET PROGRAMMA
;
FLIP    LDA   INGA              BEPAAL HET SETTEN
        ORA   INGC
        EOR   #$FF
        AND   INGB
        ORA   UIT
        STA   UIT
        LDA   INGB              BEPAAL HET RESETTEN
        EOR   #$FF
        AND   INGA
        AND   INGC
```

```
EOR   #$FF
AND   UIT
STA   UIT
RTS
.END
```

Uitwerking: De setcombinatie wordt bepaald door:

$UIT=((A+C).B)+UIT$

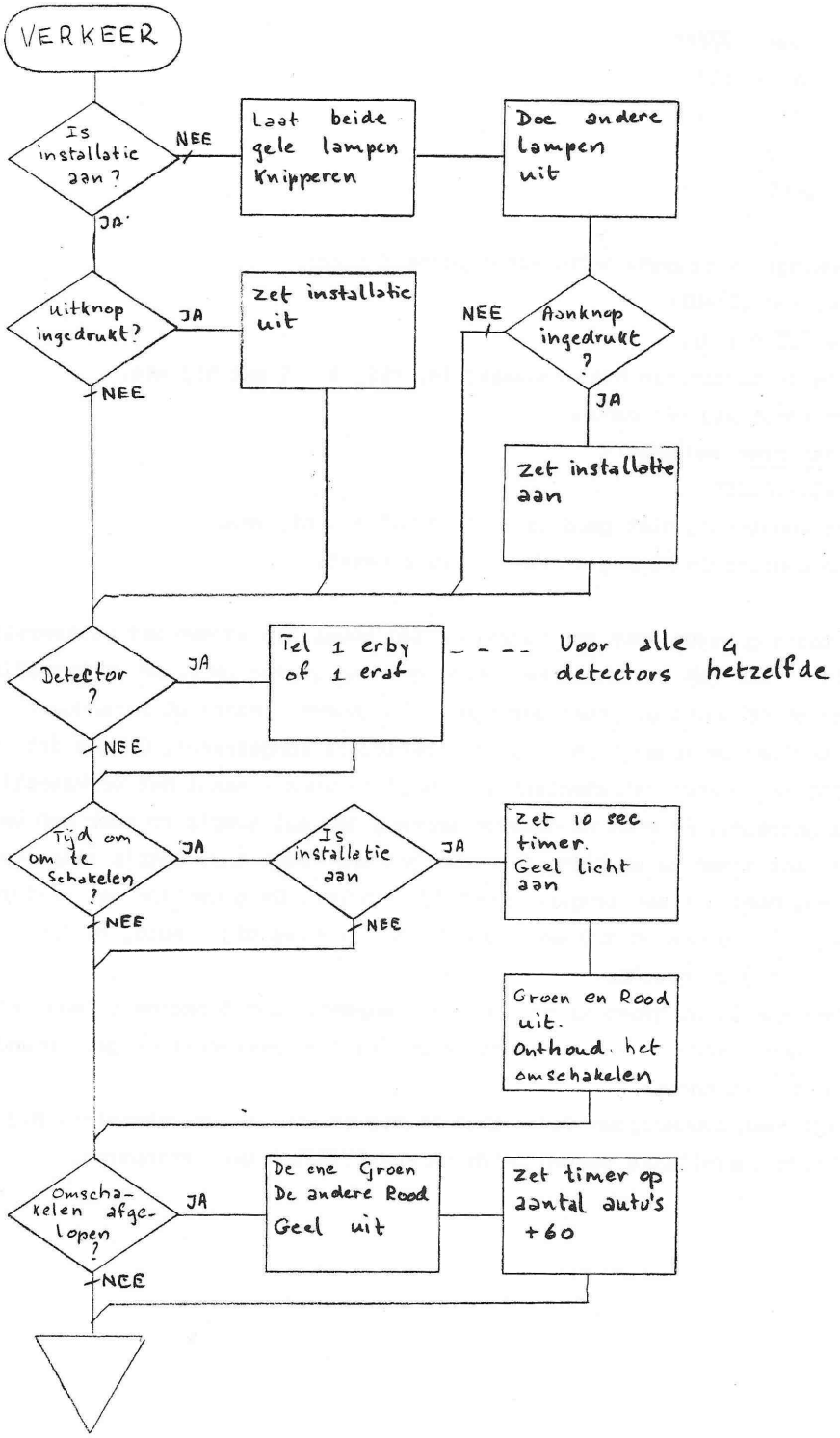
$UIT= \overline{A}.\overline{C}.B + UIT$

Dus als de combinatie niet aanwezig is, blijft uit wat hij was.
Anders wordt hij één gemaakt.
Voor het resetten geldt:

$UIT=\overline{B}.A.C.UIT$

Als de combinatie niet goed is, blijft UIT wat hij was.
Als de combinatie klopt, wordt UIT nul gemaakt.

D. Twee toegangswegen naar een tunnel moeten beveiligd worden met verkeerslichten. Er zijn 2 lichten ieder met een rode, gele en groene lamp. De verkeerslichten moeten om de beurt op groen springen. Dit gebeurt iedere 60 secondes.
In ieder van de wegen zijn 2 verkeersdetectors aangebracht. De ene detector zit 200 meter voor het stoplicht, terwijl de andere naast het verkeerslicht is aangebracht. Er moet nu gemeten worden, hoeveel auto's er voor een verkeerslicht staan te wachten. Als voor het ene licht meer auto's staan te wachten, moet dit een langere groentijd krijgen. De groentijd mag variëren tussen 60 secondes en 120 secondes. De 60 secondes bij 1 auto, de 120 secondes bij 60 auto's.
Als het ene licht groen is en rood moet worden, moet 5 secondes lang het gele licht gaan branden. Pas daarna mag rood bij dit verkeerslicht gaan branden en groen bij het andere.
Er zijn twee drukknoppen om de installatie in- en uit te schakelen. Bij uitgeschakelde installatie moeten beide verkeerslichten geel knipperen.


VERKEER
Is installatie aan ?
NEE
Laat beide gele lampen knipperen
Doe andere lampen uit
JA
Uitknop ingedrukt?
JA
Zet installatie uit
NEE
Aanknop ingedrukt ?
NEE
JA
Zet installatie aan
NEE
Detector ?
JA
Tel 1 erby of 1 eraf
Voor alle 4 detectors hetzelfde
NEE
Tijd om om te schakelen ?
JA
Is installatie aan
JA
Zet 10 sec timer. Geel licht aan
NEE
NEE
Groen en Rood uit. Onthoud het omschakelen
Omschakelen afgelopen ?
JA
De ene Groen De andere Rood Geel uit
Zet timer op aantal auto's +60
NEE

```
; . . . . SUBROUTINE VERKEER
;
; . . . . EXTERNE VARIABELEN
DET1A = INP10                    DETECTOR VOOR VERKEERSLICHT 1
DET2A = INP10+1                     "      "       "          2
DET1B = INP10+2                     "     BIJ      "          1
DET2B = INP10+3                     "      "       "          2
ROOD1 =OUTSTA                    RODE LAMP VERKEERSLICHT 1
GEEL1 =OUTSTA+1                  GELE  "      "   "    1
GROEN1=OUTSTA+2                  GROENE"      "   "    "
ROOD2 =OUTSTA+3                  Rode  "      "   "    2
GEEL2 =OUTSTA+4                  GELE  "      "   "    "
GROEN2=OUTSTA+5                  GROENE"      "   "    "
AAN =INP10+4                     INSTALLATIE AANZETTEN
UIT=INP10+5                         "        UITZETTEN
GROENT=TIMTAB                    TIMER VOOR GROENTIJD
KNIPR =TIMTAB+1                  TIMER VOOR KNIPPEREN GEEL
;
; . . . . INTERNE VARIABELEN
AANUIT=. . . .                   INSTALLATIE AAN IS 1
SCHAKL=. . . .                   OM HET OMSCHAKELEN TE ONTH.
AANTL1=. . . .                   AANTAL AUTO'S VOOR LICHT 1
AANTL2=. . . .                   AANTAL AUTO'S VOOR LICHT 2
;
;
; . . . . OM DE LICHTEN IN EEN GEDEFINIEERDE TOESTAND TE KRIJGEN MOET IN
;         HET BEGIN $FF IN AANUIT, 0 IN SCHAKL EN IN AANTL1 EN AANTL2
;         GEZET WORDEN
;
;
```

```
; . . . . BEGIN VAN HET PROGRAMMA
;
VERKER  LDA   AANUIT          IS INSTALLATIE AAN?
        BNE   TISAAN          BRANCH=JA
        LDA   KNIPR           NEE, LAAT BEIDE GELE
        BNE   NOKNIP          KNIPPEREN
        LDA   #1              1X PER 2 SECONDES
        STA   KNIPR           AAN EN UIT
        LDA   GEEL1
        EOR   #$FF
        STA   GEEL1
        STA   GEEL2
NOKNIP  LDA   #0              ANDERE LAMPEN UIT
        STA   ROOD1
        STA   ROOD2
        STA   GROEN1
        STA   GROEN2
        LDA   AAN             IS AAN INGEDRUKT?
        BEQ   TESDET          BRANCH=NEE
        LDA   #$FF            ZET INSTALLATIE AAN
        STA   AANUIT
        STA   ROOD1           GELE LAMPEN UIT, RODE
        STA   ROOD2           AAN
        LDA   #0
        STA   GEEL1
        STA   GEEL2
        LDA   #60             WACHT 60 SECONDES ALVORENS
        STA   GROENT          TE SCHAKELEN
        IMP   TESDET
TISAAN  LDA   UIT             UITKNOP INGEDRUKT?
        BEQ   TESDET          BRANCH=NEE
        LDA   #0              JA, SCHAKEL UIT
        STA   AANUIT
TESDET  LDA   DET1A           DETEKTOR 1A AAN?
        BEQ   NO1             BRANCH=NEE
        INC   AANTL1          JA, 1 ERBIJ
NO1     LDA   DET1B           DETEKTOR 1B AAN?
        BEQ   NO2             BRANCH=NEE
        DEC   AANTL1          JA, 1 ERAF
```

```
NO2     LDA   DET2A      DETEKTOR 2A AAN?
        BEQ   NO3        BRANCH=NEE
        INC   AANTL2     JA, 1 ERBIJ
NO3     LDA   DET2B      DETEKTOR 2B AAN?
        BEQ   NO4        BRANCH=NEE
        DEC   AANTL2     JA, 1 ERAF
NO4     LDA   GROENT     TIJD OM OM TE SCHAKELEN?
        BNE   NOTIME     BRANCH=NEE
        LDA   AANUIT     JA, IS DE INSTALLATIE AAN?
        BEQ   NOTIME     BRANCH=NEE
        LDA   #10        JA, ZET 10 SEC TIMER
        STA   KNIPR
        LDA   GROEN1     WAS LICHT 1 GROEN!
        BEQ   NOTEEN     BRANCH=NEE
        LDA   #$FF       JA, GEEL AAN, GROEN UIT
        STA   GEEL1
        LDA   #0
        STA   GROEN1
        STA   ROOD1      ROOD OOK MAAR UIT
        IMP   SCHKOM
NOTEEN  LDA   #$FF       HETZELFDE, MAAR VOOR
        STA   GEEL2      LICHT 2
        LDA   #0
        STA   GROEN2
        STA   ROOD2
SCHKOM  LDA   #$FF       ONTHOUDT HET OMSCHAKELEN
        STA   SCHAKL
NOTIME  LDA   SCHAKL     IS OMSCHAKELEN AFGELOPEN?
        BEQ   KLAAR      BRANCH=NEE
        LDA   KNIPR
        BNE   KLAAR      BRANCH=NEE
        LDA   GEEL1      JA, ZET DE ANDERE GROEN EN
        BEQ   ANDER      DEZE OP ROOD
        LDA   #$FF
        STA   ROOD1      ROOD AAN, GEEL UIT BIJ
        STA   GROEN2     DEZE, GROEN AAN, ROOD UIT
        LDA   #0         BIJ DE ANDERE
```

```
        STA   GEEL1
        STA   ROOD2
        LDA   AANTL2              AANTAL AUTO'S VOOR LICHT 2
        IMP   VEDER
ANDER   LDA   #$FF                IDEM VOOR ANDERE SITKATIE
        STA   ROOD2
        STA   GROEN1
        LDA   #0
        STA   GEEL 2
        STA   ROOD1
        LDA   AANTL1
VEDER   LSR                       DOOR2 DELEN
        CMP   #60                 MEER DAN 60?
        BMI   NOMOR               BRANCH=NEE
        LDA   #60                 JA, DAN 60 NEMEN
NOMOR   CLC                       60 SECONDES EXTRA
        ADC   #60
        STA   GROENT              NIEUWE GROENTIJD
        LDA   #0                  OMSCHAKELEN IS GESCHIEDT
        STA   SCHAKL
KLAAR   RTS
       .END
```

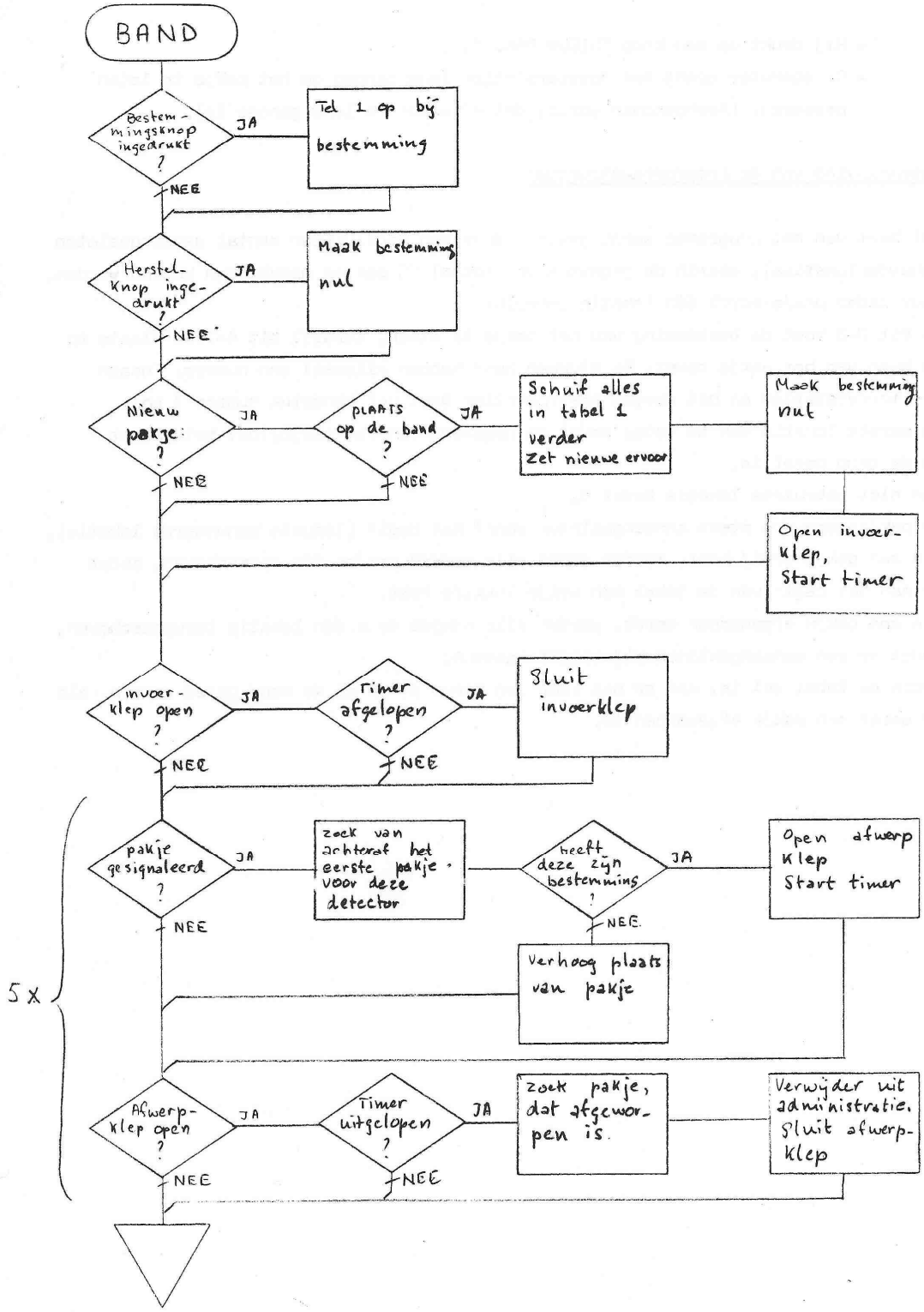
E. Een transportband heeft 5 afwerpstations en een toevoerstation. Op deze transportband komen maximaal 50 pakjes te liggen. Bij het toevoerstation zit iemand, die voor ieder pakje, dat op de band komt een bestemming opgeeft. Hij geeft het nummer van het afwerpstation (1 t/m 5) op door een aantal malen op een drukknop te drukken.
Bij ieder afwerpstation bevindt zich een pakjesdetektor en een afwerpinrichting. De pakjes kunnen elkaar niet passeren op de band en er kunnen ook geen pakjes van de band vallen.
De computer kan aan de detektor's zien of een pakje voor een afwerpstation is en moet aan de volgorde, waarin de pakjes passeren, zien welk pakje zich waar bevindt.
De procedure bij het toevoerstation is als volgt:

- De bedieningsman drukt het nummer van het afwerpstation in. Hij kan bij vergissingen corrigeren door een correctieknop.
- Hij plaatst het nieuwe pakje in het toevoerstation.

- Hij drukt op een knop "NIEUW PAKJE".
- De computer opent het toevoerstation lang genoeg om het pakje te laten passeren. (Aaangenomen wordt, dat 40 secondes lang genoeg is).

Beschrijving van de programma-structuur

Het hart van het programma wordt gevormd door een tabel (=een aantal aaneengesloten geheugenlokaties), waarin de gegevens van totaal 50 pakjes opgeborgen kunnen worden. Voor ieder pakje wordt één lokatie gebruikt.
In bit 0-3 komt de bestemming van het pakje te staan, terwijl bit 4-7 de plaats op de band van het pakje bevat. De stukken band hebben allemaal een nummer. Tussen het invoerstation en het eerste afwerpstation heet het bandstuk nummer 1 enz.
De eerste lokatie van de tabel bevat de gegevens van het pakje, dat het laatst op de band gezet is.
Een niet gebruikte lokatie bevat 0.
De pakjesgegevens staan aaneengesloten vanaf het begin (laagste genummerde lokatie).
Als een pakje erbij komt, worden eerst alle andere pakjes één opgeschoven, zodat er aan het begin van de tabel een vrije lokatie komt.
Als een pakje afgeworpen wordt, worden alle pakjes erna één lokatie teruggeschoven, zodat er een aaneengesloten rij blijft bestaan.
Zodra de tabel vol is, kan er pas weer een nieuw pakje op de band gezet worden, als er eerst een pakje afgeworpen is.


BAND
Bestem-mingsknop ingedrukt ?
JA
Tel 1 op bij bestemming
NEE
Herstel-knop inge-drukt ?
JA
Maak bestemming nul
NEE
Nieuw pakje ?
JA
PLAATS op de band ?
JA
Schuif alles in tabel 1 verder zet nieuwe ervoor
Maak bestemming nul
NEE
NEE
Open invoer-klep, Start timer
invoer-klep open ?
JA
Timer afgelopen ?
JA
Sluit invoerklep
NEE
NEE
5 X
pakje gesignaleerd ?
JA
zoek van achteraf het eerste pakje voor deze detector
heeft deze zijn bestemming ?
JA
Open afwerp klep Start timer
NEE
NEE
Verhoog plaats van pakje
Afwerp-klep open ?
JA
Timer uitgelopen ?
JA
zoek pakje, dat afgewor-pen is.
Verwijder uit administratie. Sluit afwerp-klep
NEE
NEE

```
; . . . . SUBROUTINE BAND
;
; . . . . EXTERNE VARIABELEN
;
DET1=INP10                      DETEKTOR AFWERPSTATION 1
DET2=INP10+1                      "       "       "    2
DET3=INP10+2                      "       "       "    3
DET4=INP10+3                      "       "       "    4
DET5=INP10+4                      "       "       "    5
PAKIN=INP10+5                   NIEUW PAKJE OP BAND
NUMMER=INP10+6                  KNOP OM BESTEMMING IN TE VULLEN
HERSTL=INP10+7                  CORRECTIE
;
TIMER1=TIMTAB                   TIMER AFWERPSTATION 1
TIMER2=TIMTAB+1                   "     "       "    2
TIMER3=TIMTAB+2                   "     "       "    3
TIMER4=TIMTAB+3                   "     "       "    4
TIMER5=TIMTAB+4                   "     "       "    5
TIMPAK=TIMTAB+5                   "   INVOERSTATION
;
AFWRP1=OUTSTA                   AFWERPKLEP 1
AFWRP2=OUTSTA+1                   "        2
AFWRP3=OUTSTA+2                   "        3
AFWRP4=OUTSTA+3                   "        4
AFWRP5=OUTSTA+4                   "        5
INVOER=OUTSTA+5                 INVOERKLEP
; . . . . INTERNE VARIABELEN
ADMIN= . . . .                  50 LOKATIES VOOR ADMINISTRATIE
BESTEM= . . . .                 OM BESTEMMING TE ONTHOUDEN
;
HULP= . . . .                   HULPLOKATIE
;
```

```
; . . . . BEGIN PROGRAMMA
BAND    LDA     NUMMER      BESTEMMINGSKNOP INGEDRUKT?
        BEQ     NONUM       BRANCH=NEE
        INC     BESTEM      JA, TEL 1 OP BIJ BESTEMMING
NONUM   LDA     HERSTL      HERSTELKNOP INGEDRUKT?
        BEQ     NOHERS      BRANCH=NEE
        LDA     #0          JA, MAAK BESTEMMING NUL
        STA     BESTEM
NOHERS  LDA     PAKIN       NIEUW PAKJE?
        BEQ     NOPAK       BRANCH=NEE
        LDA     BESTEM      JA, BESTEMMING INGEVULD
        BEQ     NOPAK       BRANCH=NEE
        CMP     #6          JA, BESTEMMING OK?
        BMI     TISPAK      BRANCH=JA
        LDA     #0          NEE, MAAK BESTEMMING NUL
        STA     BESTEM
        IMP     NOPAK       EN WEIGER
TISPAK  LDA     ADMIN+49    PLAATS OP DE BAND?
        BNE     NOPAK       BRANCH=NEE
        LDX     #48         JA, SCHUIF ALLES " PLAATS VERDER
SCHUIF  LDA     ADMIN,X
        STA     ADMIN+1,X
        DEX
        BPL     SCHUIF
        LDA     BESTEM      ZET NIEUWE PAKJE VOORAAN
        CLC
        ADC     #$10        BANDSTUK=1
        STA     ADMIN
        LDA     #0          MAAK BESTEMMING NUL
        STA     BESTEM
        LDA     #$FF        OPEN INVOERKLEP
        STA     INVOER
        LDA     #40         ZET ZIJN TIMER
        STA     TIMPAK
NOPAK   LDA     INVOER      INVOERKLEP OPEN?
        BEQ     NOTINV      BRANCH=NEE
        LDA     TIMPAK      JA, TIMER UITGELOPEN?
        BNE     NOTINV      BRANCH=NEE
```

```
        LDA     #0              JA, SLUIT INVOERKLEP
        STA     INVOER
NOTINV  LDX     #4              VOOR 5 AFWERPSTATIONS
WERDET  LDA     DET1,X          PAKJE GESIGNALEERD?
        BEQ     NOTOUT          BRANCH=NEE
        LDY     #49             JA, ZOEK HET PAKJE
WERZOK  LDA     ADMIN,Y         IS DIT HET JUISTE?
        AND     #$FD            BIT 4-7 = PLAATS OP BAND
        LSR
        LSR
        LSR
        LSR
        STA     HULP
        CPX     HULP
        BEQ     PAKFND          BRANCH=JA
        DEY                     NEE, VOLGENDE
        BPL     VERZOK
; . . . . ALS HET PROGRAMMA HIER OOIT KOMT, IS DIT EEN DETEKTORFOUT.
;         WE ZULLEN HEM MAAR DOEN ALSOF ER NIETS GEBEURD IS.
        IMP     NOTOUT
;
PAKFND  LDA     ADMIN,Y         HEEFT DEZE ZIJN BESTEMMING
        AND     #$F             BEREIKT?
        STA     HULP
        CPX     HULP
        BEQ     TISER           BRANCH=JA
        LDA     ADMIN,Y         NEE, VERHOOG ZIJN PLAATS
        CLC                     OP DE BAND MET 1
        ADC     #$10
        STA     ADMIN,Y
        IMP     NOTOUT
TISER   LDA     #$FF            OPEN AFWERPKLEP
        STA     AFWRP1,X
        LDA     #40             START ZIJN TIMER
        STA     TIMER1,X
NOTOUT  LDA     AFWRP1,X        AFWERPKLEP OPEN
        BEQ     NOTOPN          BRANCH=NEE
        LDA     TIMER1,X        JA, TIMER UITGELOPEN?
```

```
        BNE     NOTOPN          BRANCH=NEE
        LDY     #49             JA, ZOEK AFGEWORPEN PAKJE
FINDEM  LDA     ADMIN,Y
        AND     #$F
        STA     HULP
        CPX     HULP            GEVONDEN?
        BEQ     WERPEM          BRANCH=JA
        DEY                     NEE, VOLGENDE
        IMP     FINDEM
WERPEM  LDA     ADMIN+1,Y       VERWIJDER DIT PAKJE
        STA     ADMIN,Y         KLAAR?
        BEQ     DONVER          BRANCH=JA
        INY                     NEE, VOLGENDE
        CPY     #49             LAATSTE PLAATS?
        BNE     WERPEM          BRANCH=NEE
DONVER  LDA     #0              JA, SLUIT AFWERPKLEP
        STA     AFWRP1,X
NOTOPN  DEX                     LAATSTE STATION GEHAD?
        BPL     WERDET          BRANCH=NEE
        RTS                     JA, KEER TERUG
; . . . . SUBROUTINE OM HET PROGRAMMA OP DE
;         JUISTE WIJZE TE INITIALIZEREN
;
INIT    LDX     #49
        LDA     #0
INTWER  STA     ADMIN,X
        DEX
        BPL     INTWER
        STA     BESTEM
        RTS
; . . . . DE VOORGAANDE SUBROUTINE MOET IN HET BEGIN VAN "LOGICA" MET EEN
          -ISR INIT - AANGEROEPEN WORDEN.
       .END
```

Hoofdstuk 5.

Wachten en timing I.

Bij het programmeren kan het voorkomen, dat het programma moet wachten op een gebeurtenis van buitenaf, voordat het bepaalde akties onderneemt.
Eén van de meest voorkomende dingen, waarop gewacht moet worden is de tijd.
Hetzij dat een bepaalde hoeveelheid tijd moet verstrijken, hetzij dat gewacht moet worden tot een bepaald absoluut tijdstip gepasseerd is.
We zullen ons eerst gaan bezighouden met het wachten tot een bepaalde hoeveelheid tijd verstreken is.
Principiëel zijn er twee verschillende wijzen, terwijl één van de manieren twee verschillende uitvoeringsvormen kent.

- Manier 1. Als het totale computerprogramma in de letterlijke zin des woords een bepaalde tijd moet wachten, kan dit gedaan worden door instrukties uit te voeren, die "een pas op de plaats" maken.
  Als een tijd X gewacht moet worden, kan dit gedaan worden door een stukje programma, dat X microsecondes lang een NOP-instruktie uitvoert. (de zg. wachtlus)
  Voorbeeld: Er moet 150 microsecondes lang gewacht worden.

```
Programma:      LDA   #X
                SEC
          WACHT SBC   #1
                BNE   WACHT
```

Een LDA #X duurt 2 microsecondes
" CLC " 2 " "
" SBC #1 " 2 " "
" BNE ... " 2 " "

De laatste 2 instrukties worden X maal doorlopen. De totale wachttijd is dus:

$$T = 4+4X = 4(X+1)$$

Als de wachttijd 150 microsecondes moet zijn, krijgen we de vergelijking:

$$150=4(X+1)$$

$$X = \frac{150}{4} - 1 = 36,5$$

Aangezien we geen 0,5 voor X in kunnen vullen, moet hier iets beters op gevonden worden. Stel dat we X 36 nemen.
De totale tijdsduur bedraagt dan:

$$T = 4(36+1) = 148 \text{ microsecondes}$$

Als nu het stukje programma voorafgegaan of gevolgd wordt door een NOP (2 microsecondes), komen we precies op 150 microsecondes.
Het maximale wachtbereik wordt op deze manier bereikt als we X 0 maken.
De laatste 2 instrukties worden dan 256 maal doorlopen.

T = 4(256+1) = 1028 microsecondes.

Aangezien onze tijdbasis hier de kristaloseillator van de KIM is, is de nauwkeurigheid vrij groot, en we kunnen tijden creëeren in veelvouden van 1 microseconde.
Als de te wachten tijd groter moet worden dan 1028 microsecondes, kan dit gedaan worden door het stuk programma meerdere malen aan te roepen.
Voorbeeld: Een subroutine om 1 milliseconde te wachten:

```
; . . . . SUBROUTINE WACHT1
;
;        DE SUBROUTINE LAAT DE REGISTERS A,X EN Y ON AANGETAST.
;        DOOR DE SUBROUTINE N MAAL AAN TE ROEPEN, WORDT N MILLISECONDES
;        GEWACHT
;
; . . . . AANROEP:        LDA   #N
;               WEER       ISR   WACHT+1
; . . . .                  SEC
;                          SBC   #1
;                          BPL   WEER
; . . . . BEGIN SUBROUTINE
;
WACHT1    PHA
          LDA   #243
          SEC
WACHT     SBC   #1
          BPL   WACHT
          ADC   #0        DEZE EN VOLGENDE ZIJN ER OM 5
          NOP             MICROSECONDES TE GEBRUIKEN
          PLA
          RTS
         .END
```

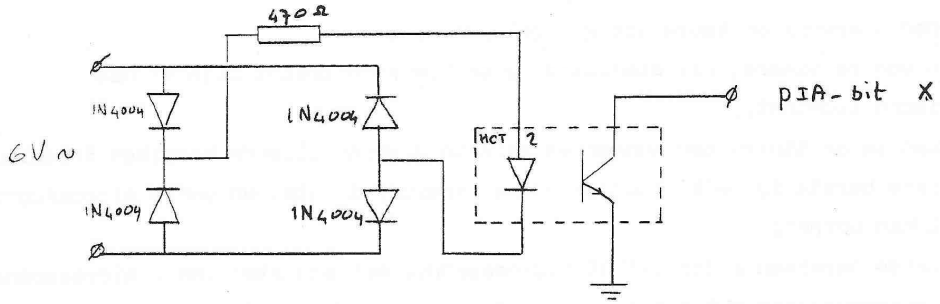
Merk op, dat bij dit programma de werkelijke wachttijd altijd 2 microsecondes te lang is, vanwege de LDA #N instruktie.

- Manier 2. Gebruikten we bij manier 1 de computer zelf om uit te rekenen of er al lang genoeg gewacht was, bij deze manier gebruiken we een apparaat buiten de computer om dit uit te rekenen.
We hebben hiervoor de keuze uit 2 mogelijkheden.
Nl. één van de timers, die standaard op de KIM aangebracht zijn of het elektrische lichtnet.
Gebruiken we de timer, dan kunnen we deze in 4 verschillende bereiken instellen.
Het eerste bereik is 1-127 microsecondes, waarbij in stappen van 1 microsecondes gewacht kan worden.
De volgende bereiken zijn: 1-1016 microsecondes met een stap van 8 microsecondes, 1-8126 microsecondes met een stap van 64 microsecondes en 1-130048 microsecondes met een stap van 1024 microsecondes.
Aangezien we onze wachttijden graag specificeren in decimale veelvouden van 1 microseconde, komt in de praktijk het eerste bereik verreweg het meeste voor.
Voorbeeld: Wacht N millisecondes waarbij N loopt van 1 t/m 256.

```
Programma:    LDX  #N
        WAAR  LDY  #4
        WEER  LDA  #238
              STA  $1704
        WACHT LDA  $1707
              BPL  WACHT
              DEY
              BNE  WEER
              NOP
              DEX
              BNE  WAAR
```

De berekening van de tijden is identiek aan die van manier 1. We moeten er rekening mee houden, dat na de instruktie STA $1704 altijd tenminste 238 microsecondes zullen verstrijken voordat de timer uitloopt. Dit gebeurt onafhankelijk van het programma. De instrukties LDA $1707 en BPL WACHT kosten dus geen tijd, die meetelt.
Dit geeft ons de mogelijkheid om in deze 238 microsecondes toch nog iets te doen met de computer.
Eén van de dingen, die we graag doen in deze zg. "idle-time" is het besturen van het KIM-display. Hiervan zal in de volgende hoofdstukken gebruik gemaakt worden. De enige voorwaarde die voor het gebruik van de idletime geldt, is, dat onder geen enkel beding meer tijd gebruikt mag worden, dan beschikbaar is. Het bedienen van het display en keyboard is een bezigheid, die ruimschoots binnen deze tijd kan plaatsvinden.

Het gebruik van het lichtnet als timer kan door een wisselspanning gelijk te richten en deze pulserende gelijkspanning aan een PIA-bit te schakelen. Het volgende schema voldoet aan deze schakeling.



Pia-bit X zal nu iedere 10 millisecondes 0 en 1 worden. Er is geen variabele timing mogelijk, alhoewel 10 millisecondes voor veel toepassingen een bruikbare maat is.

Voorbeeld: Wacht X secondes met behulp van een lichtnet-timer op PIA bit 0 A-zijde.

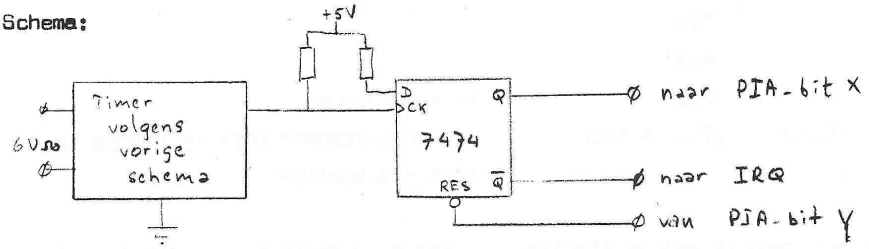
Programma:

```
        LDX   #X
WEER    LDY   #100      100X10=1000 millisecondes
WACHT   LDA   $1700     WACHT TOT HET BIT1
        STA   HULP      WORDT (VAN 0 NAAR 1 GAAT)
        EOR   VORIG
        AND   HULP
        LSR
        BCC   WACHT     BRANCH=NOG NIET
        LDA   HULP      10 millisecondes om
        STA   VORIG
        DEY             1 seconde om?
        BNE   WACHT     BRANCH=NEE
        DEX             JA, AL X SECONDES?
        BNE   WEER      BRANCH=NEE
        . . . . . .     JA
```

Hier hebben we ook weer de mogelijkheid om met het programma iets anders te doen. Dat moet dan gedaan worden tussen WEER en WACHT. Datgene, wat er gedaan wordt, mag uiteraard niet langer duren dan 10 millisecondes.

- Manier 3. Deze manier verschilt van de vorige door het gebruik van de interrupt-faciliteit (IRQ). De tijdbasis is òf de applicatietimer van de KIM òf een lichtnet-timer, die iets uitgebreider is dan de vorige.

Schema:



Voor het gebruik van de KIM-timer per interrupt, is het nodig om PB7 (Appl.connector) aan IRQ (expansion connector) te verbinden.

Door gebruikmaking van de interrupt lopen er als het ware twee programma's. Het lopende programma wordt onderbroken, zodra de timer een interrupt genereert. Hierdoor wordt het interrupt programma gestart. Het interrupt-programma moet er dan tenminste voor zorgen, dat de oorzaak van de interrupt verdwijnt en verder het programma uitvoeren, dat nodig is.

Basisuitvoering van een interruptprogramma:

```
INTRPT   PHA            BEWAAR HET A-REGISTER
         TXA
         PHA            EN HET X-REGISTER
         TYA
         PHA            EN HET Y-REGISTER
         LDA $1707      IS HET DE KIM-TIMER?
         BPL NOKIM      BRANCH=NEE
         ISR KIMTIM     VERWERK DE INTERRUPT
         IMP INTUIT
NOKIM    LDA $1700      TEST PIA-BIT X VOOR DE
         AND #BIT X     LICHTNET-TIMER
         BEQ NOLICH     BRANCH=NEE
         LDA #BIT Y     JA, HET WAS DE LICHTNETTIMER
         EOR #$FF       RESET DE FLIPFLOP
         ORA $1700      MAAK BIT Y  0 EN DAARNA WEER 1
         STA $1700
         ORA #BIT Y
         STA $1700
         ISR LICTIM     DOE WAT NODIG IS
```

```
INTUIT    PLA              KLAAR
          TAY              HERSTEL DE REGISTERS
          PLA
          TAX
          PLA
          RTI              EN KEER TERUG
NOLICH    JMP  $1C00       ALS DE INTERRUPT NIET TE VINDEN IS,
;                          NAAR KIM-MONITOR
```

Voor het gebruik van de Kim-timer is het noodzakelijk om aan het begin van het programma PB7 als input te zetten. Voor de lichtnettimer moet PIA-bit X een input- en PIA-bit Y een outputbit zijn.
Als voorbeeld worden nu twee subroutines gegeven, die allebei de lokatie TIME aftellen tot nul, zodra deze bij een timer-interrupt ongelijk is aan nul. Het tellen gebeurt eens per seconde.

```
; . . . . INTERRUPT PROGRAMMA VOOR DE KIM-TIMER.
;
;        TELT DE LOKATIE TIME LEEG
;
; . . . . DEZE SUBROUTINE WORDT AANGEROEPEN,
;        NADAT GECONSTATEERD IS, DAT DE KIM-TIMER GEÏNTERRUMPEERD HEEFT
;
; . . . . EXTERNE VARIABELEN
TIME= . . . .                  LEEG TE TELLEN LOKATIE
;
; . . . . INTERNE VARIABELEN
;
T1= . . . .                    OM  100 X 100 MICROSECONDES TE TELLEN
T2= . . . .                    OM  100 X  10 MICROSECONDES TE TELLEN
;
; . . . . ATTRIBUTEN
WAARDE=$1706                   DOORTELLING VAN DE TIMER
TIMER=$170C                    HET TIMER 1 MICROSECONDE REGISTER
;
; . . . . BEGIN SUBROUTINE
KIMTIM    LDA    WAARDE        LEES REGISTER UIT
          CLC
          ADC    #96           TEL ER TOTAAL 100 BY
          STA    TIMER         NU LOOPT HIJ WEER
```

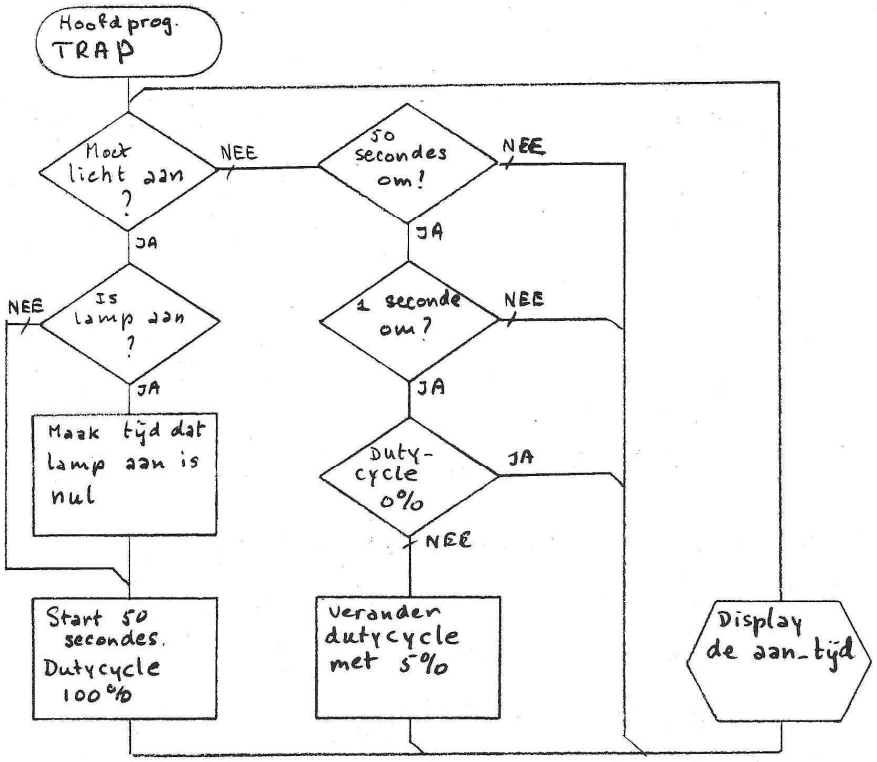
```
          DEC     T1              TEL 100 AF. NUL?
          BNE     UITTIM          BRANCH=NEE
          LDA     #100            JA, ZET WEER OP 100
          STA     T1
          DEC     T2              TEL 100 AF. NUL?
          BNE     UITTIM          BRANCH=NEE
          LDA     #100            JA, ZET WEER OP 100
          STA     T2
; . . . . ER IS NU 1 SECONDE VOORBIJ.
          LDA     TIME            KIJK OF TIMER NIET NUL IS
          BEQ     UITTIM          BRANCH=NUL
          DEC     TIME            NIET NUL. TEL 1 AF.
UITTEM    RTS                     KEER TERUG
         .END

; . . . . INTERRUPT PROGRAMMA LICHTNET TIMER
;
;         TELT DE LOKATIE TIME LEEG
;
; . . . . DEZE SUBROUTINE WORDT AANGEROEPEN
;         NADAT GECONSTATEERD IS, DAT DE LICHTNET-TIMER GEÏNTERRUMPEERD HEEFT.
;
; . . . . EXTERNE VARIABELEN
TIME= . . . .
;
; . . . . INTERNE VARIABELENT1= . . . .     100 x 10 MILLISECONDES
;
; . . . . BEGIN SUBROUTINE
;
LICTIM    DEC    T1.          ER ZIJN 10 MILLISECONDES OM
          BNE    UITTEM       NOG GEEN SECONDE
          LDA    #100         WEL 1 SECONDE
          STA    T1
          LDA    TIME         IS TIME ONGELIJK NUL?
          BEQ    UITTIM       BRANCH=NUL
          DEC    TIME         NIET NUL. TEL AF.
UITTIM    RTS                 KEER TERUG
         .END
```

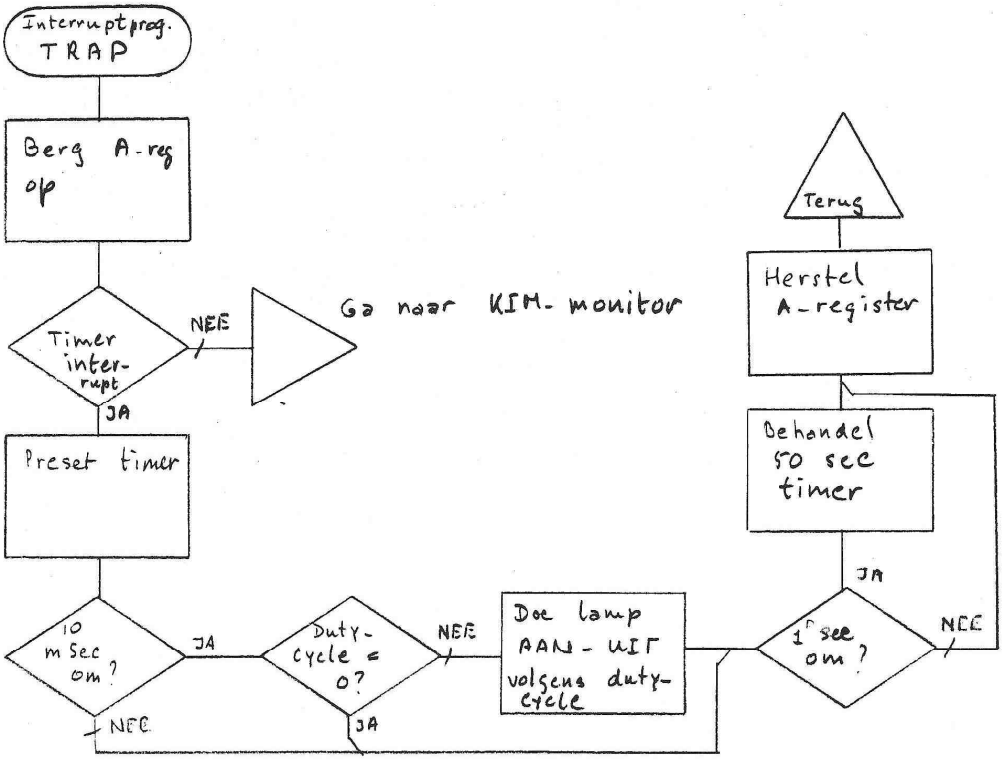
Hoofdstuk 6.

Wachten en timing II.

A. Trapportaal verlichting.

Ieder van ons kent de trappenhuizen, waar men door het indrukken van een knop het licht ongeveer 1 minuut kan laten branden. Gaat na 1 minuut het licht uit, dan moet men in het donker maar weer een knop zien te vinden om het aan te doen. Als de knoppen nu op de KIM aangesloten worden (allemaal parallel op een PIA-bit), evenals de lampen, kan een programma er voor zorgen, dat een minuut het licht langzaam uitgaat, zodat men gewaarschuwd wordt en nog 20 secondes heeft, voordat het licht helemaal uit is.
We zullen dit doen, door de duty-cycle van het licht te veranderen.
Als de lamp op halve sterkte brandt, dan wordt dit gedaan,door hem evenlang aan als uit te doen in een tijdbestek van 10 millisecondes.


Hoofd prog.
TRAP
Moet licht aan ?
NEE
50 secondes om!
NEE
JA
JA
NEE
Is lamp aan ?
1 seconde om?
NEE
JA
JA
Maak tijd dat lamp aan is nul
Duty-cycle 0%
JA
NEE
Start 50 secondes. Dutycycle 100%
Verander dutycycle met 5%
Display de aan_tijd


Interruptprog.
TRAP
Berg A-reg op
Timer inter-rupt
NEE
Ga naar KIM- monitor
JA
Preset timer
10 m Sec om ?
JA
Duty-cycle = 0?
NEE
Doe lamp AAN - UIT volgens duty-cycle
NEE
JA
Terug
Herstel A-register
Behandel 50 sec timer
1 sec om ?
JA
NEE