INLEIDING

Om een computer iets te laten doen moet hem eerst verteld worden welke handelingen eksakt moeten worden verricht. Dit is de taak van de computerprogrammeur. Aangezien de computer niet in staat is een natuurlijke taal te verstaan, zullen momenteel de programmeurs een taal moeten leren die door de computer wel wordt verstaan. BASIC is zo´n taal.

BASIC is ontwikkeld aan de universiteit van Dartmouth met als doel een taal te maken die door de computer verstaan wordt en door de mens eenvoudig te leren is. Alhoewel BASIC tot een van de simpelste computertalen wordt gerekend is het toch zeer krachtig. Het gebruikt een klein aantal woorden die een vaste betekenis hebben. Daar komt nog bij dat het een samenspel met de programmeur mogelijk maakt door alle commando´s ook direkt uit te voeren wanneer ze zijn ingetoetst.

Doordat de programmeertaal BASIC op alle computers beschikbaar is, is er een veelheid van (cursus-) boeken en programma´s voor deze taal geschreven. Hierdoor is het mogelijk programma´s die op een andere computer geschreven zijn ook op Uw computer te laten werken.

HET GEBRUIK VAN BASIC

Nadat de BASIC interpreter is gestart, verschijnt de vraag

```
memory limit?
```

Omdat BASIC het hele beschikbare geheugen in gebruik neemt, is het soms wenselijk een stuk geheugen voor eigen (niet BASIC) toepassingen vrij te houden. Dit kan door nu aan BASIC het eerste geheugenadres op te geven welke door BASIC niet meer gebruikt mag worden.
Dit adres kan in decimaal of in hexadecimaal worden opgegeven:

```
memory limit? 20000          decimaal

memory limit? &5000          hexadecimaal
```

Wanneer geen getal wordt ingegeven ´meet´ BASIC hoeveel geheugen beschikbaar is en neemt dit allemaal in gebruik.

Vervolgens vraagt BASIC hoe breed de programmauitvoer mag worden afgedrukt. Wanneer U een 40-koloms beeldscherm heeft en een 80-koloms printer en U wilt de uitvoer op de printer, dan geeft U als ´width´ 80 op.

```
width? 80
```

Wanneer U geen getal opgeeft, wordt de breedte van Uw beeldscherm door BASIC gekozen. Voor de PDS-65 is dit 80, voor de PC-2 40.

Hierna meldt BASIC zich met een rapportering van de totaal beschikbare geheugenruimte:

free space: XXXXX (weergegeven in decimaal)

Nu is BASIC gereed om commando´s te verwerken. Deze commando´s zijn in de volgende hoofdstukken beschreven.
Tijdens het intypen op het toetsenbord zijn een aantal toetsen beschikbaar voor het corrigeren van de zojuist ingetypte tekens:

toets	PC-2	PDS-65
´pijltje links´	cursor links	verwijder laatste teken
´pijltje rechts´	cursor rechts	
´pijltje omhoog´	cursor omhoog (*)	
´pijltje omlaag´	cursor omlaag (*)	
´char insert´	voeg 1 plaats tussen	
´char delete´	verwijder 1 plaats	
´ctrl/Q´	naar ´command-mode´	naar ´command-mode´

(*) Werken NIET tijdens input-statements.

Op de PC-2 is een ´full screen editor´ aangebracht, waarmee het programma op een snelle wijze kan worden ingevoerd of veranderd.
Voor de beschrijving van deze editor zie hoofdstuk 4.
Algemeen geldt dat een ingetypte regel als programmaregel wordt opgeslagen als de regel begint met een regelnummer.
bv: 10 print "Hallo daar" is een programmaregel en wordt in het geheugen opgeslagen en een regel: print "Hallo daar" wordt direkt uitgevoerd.

HOOFDSTUK 1 Commando´s en instructies

Hieronder volgt een kort overzicht van de BASIC woorden die voor het gebruik in programma´s en als direkte commando´s beschikbaar zijn.

COMMANDO´s

naam	funktie
clear	geef alle variabelen de waarde 0
cont	continueer na een ´break´
list	list het programma
load	laadt het programma van cassette (evt floppy)
save	schrijf het programma naar cassette (evt floppy)
new	verwijder het programma uit het geheugen
run	start het programma
@	print geheugenlocaties
upcase	zorg voor ´grote´ letters bij programmalisting
lowcase	grote en kleine letters bij programmalisting
auto	start automatische regelnummering
renum	hernummer de regels (niet in versie 2.0)
mon	verlaat BASIC en start het monitorprogramma

STATEMENTS

def fn	definieer een funktie
dim	definieer een matrixgrootte
end	beeindig het programma
for . . . next	programmalus met een lus-teller
goto	spring naar een andere regel
gosub	voer een onderprogramma uit (subroutine)
if . . goto	spring als de conditie ´waar´ is
if . . then . . (else . .)	voer uit als de conditie ´waar´ is
let	geeft een variabele een waarde
on . . goto	meerweg splitsing
on . . gosub	meerweg splitsing voor onderprogramma´s
repeat . . until	programmalus met eindconditie
rem	zet een ´remark´ regel (commentaar)
\	een ´remark´ regel (commentaar)
restore	zet de data-lees-wijzer aan het begin
return	beeindig een onderprogramma
stop	onderbreek een programma
usr	voer een machinecode funktie uit
sys	voer een machinetaal routine uit
wait	wacht op speciale conditie

IN en UITVOER

data	deze regel bevat invoer gegevens
get	haal 1 karakter van het toetsenbord
input	haal een getal of tekst van het toetsenbord
read	lees de volgende data-regel
print	druk een regel af
spc	spatieer (tijdens print)
tab	tabuleer (tijdens print)

inkey	lees het toetsenbord (wacht niet op een toets)
open	open een input- of outputkanaal
close	sluit een input- of outputkanaal
peek	kijk in een geheugenlocatie
dpeek	kijk in een geheugenlocatie (16-bit)
poke	zet iets in een geheugenlocatie
dpoke	zet iets in een geheugenlocatie (16-bit)

TEKST BEHANDELING (STRINGS)

asc	geeft een getalrepresentatie van de letter
chr$	geeft de letterrepresentatie van een getal
left$	geeft het linker deel van een string
right$	geeft het rechter deel van een string
mid$	geeft een middendeel van een string
len	geeft de lengte van de string
val	geeft het getal dat in de string staat
str$	maakt een string van een getal
hex$	maakt een hex-string (2 cijfers) van een getal
whex$	maakt een hex-string (4 cijfers) van een getal

REKENFUNKTIES

abs	geeft de absolute waarde van een getal
exp	geeft de e-macht van een getal
sgn	geeft het teken van een getal
log	geeft de natuurlijk logaritme van een getal
int	geeft de gehele waarde van een getal
sqr	geeft de vierkants wortel uit een getal
sin	geeft de sinus van een getal
cos	geeft de cosinus van een getal
tan	geeft de tangens van een getal
atn	geeft de arc-tangens van een getal
rnd	geeft een willekeurig getal

REKEN OPERATOREN

*	vermenigvuldigen
/	delen
+	optellen
-	aftrekken
^	tot-de-macht (machtsverheffen)
( )	voorrangsbewerking tussen haakjes
>	groter dan
<	kleiner dan
=	is gelijk
<>	is ongelijk
<= of =<	kleiner-of-gelijk
>= of =>	groter-of-gelijk
not	logisch 'niet'
or	logisch 'of'
and	logisch 'en'

HOOFDSTUK 2 HET PROGRAMMA

2.1 Variabelen: Getallen en teksten

Getallen en variabelen werken binnen BASIC met twee methoden:
integer (gehele getallen 0,1,2,3 . . .)
real (getallen met drijvende komma en exponent)
Het getalbereik van de integer getallen is van -32767 t/m +32767 en de reele getallen van 2.93873588 E-39 tot 1.70141183 E38 (+ of -). Het onderscheid tussen deze getallen wordt bij BASIC gemaakt door een %-teken achter de variabelenaam te zetten wanneer een integer getal bedoeld wordt.
Variabelen krijgen van de programmeur een naam. Deze naam mag tot 16 letters lang zijn. Een naam moet beginnen met een letter welke door letters en cijfers gevolgd mag worden.
Wel dient er op gelet te worden dat geen lettercombinaties van de gereserveerde woorden in een naam kunnen voorkomen bv: variabele ORDER wordt door basic opgevat als ´or´ DER, waarbij ´or´ de of-funktie tussen twee getallen wil doen.
Hieronder volgen voorbeelden van geldige namen:
VERKOOP WINST INKOOP AANTAL
Een programma voor het berekenen van het verkoopbedrag:

```
10 \  bereken het verkoopbedrag VERKOOP
20 input "geef de inkoopprijs" ; INKOOP
30 input "geef het aantal     " ; AANTAL
40 \ 15% winst
50 WINST = 1.15
60 VERKOOP = INKOOP * AANTAL * WINST
70 print "de verkoopprijs     " ; VERKOOP
80 end
```

In het voorbeeld zien we dat de variabele WINST een waarde krijgt. Deze programma instructie heet ´assignment statement´ (toewijzings-instructie). Eigenlijk hoort aan het begin van de regel het kenwoord ´let´ te staan:

```
50 let WINST = 1.15
```

Dit ´let´ is niet nodig binnen dit BASIC.
Achter het ´=´ teken mag een formule staan die door de computer moet worden berekend. Een voorbeeld hiervan is regel 60. Ook op deze regel staat een assignment statement.
Uit het voorbeeld zien we tevens dat slechts 1 statement (instructie) per regel voorkomt. Wanneer meer dan 1 statement op een regel wordt geschreven, moeten deze gescheiden worden met een ´:´.
bv 100 COUNT=12 : FACT= 1.25

Om in een programma logische beslissingen te kunnen nemen is voorzien in het ´if´-statement:
bv 110 if COUNT > 100 then print "klaar"

Tussen de woorden ´if´ en ´then´ staat het vergelijkingstest. Dit heet een ´relational´ of ´boolean´ expression. In zo´n expressie kunnen alle rekenoperaties en alle vergelijkingoperaties worden gebruikt.
bv 120 if COUNT^2 * (FACT + 3) < 0 and (TEST = 1) then print "end"

Een voorbeeld waarin de if-then instructie getest kan worden:

```
10 input TEST
20 if TEST = 0 then goto # ZERO
30 print "was niet 0."
40 end
50 # ZERO : print "was wel 0."
60 end
```

Tevens zien we hier het gebruik van de onvoorwaardelijke sprong met de ´goto´ instructie. Met de goto-instructie wordt de programma afloop onderbroken en op een andere regel hervat. In het voorbeeld is dit regel 50 want hier staat het ´label´ #ZERO.

Naast beslissingen nemen komt in een programma veelal een herhaling voor. Deze kan met het if-then statement worden gemaakt, maar ook met de for-next loop (lus).
bv
```
10 for INDEX = 0 to 100
20 print INDEX ; INDEX / 3
30 next INDEX
40 end
```

Dit programma heeft een herhalingslus met een ingebouwde lusteller. De lusteller is hier INDEX genoemd en in regel 20 wordt INDEX en INDEX/3 afgedrukt. Regel 40 geeft het einde van de lus aan en zorgt ervoor dat de processor opnieuw de lus inloopt.

Een andere lusstructuur is de repeat-until structuur. Hier is geen ingebouwde lusteller. De lus wordt net zo lang doorlopen totdat aan de conditie achter ´until´ wordt voldaan.
bv
```
10 repeat input "geef getal "; NUMBER
20 print sqr( NUMBER)
30 until NUMBER > 2000
40 end
```

2.2 Reeksen getallen en teksten

BASIC kent variabelen die een reeks getallen vertegenwoordigen. Dit wil zeggen dat het programma bv het 5-de getal van de variabele REEKS kan vragen nl: print REEKS(5)
Wanneer we de wortel van de getallen 1 t/m 12 willen opslaan:

```
10 dim A(12)
20 for INDEX=1 to 12
30 A( INDEX) = SQR( INDEX)
40 next INDEX
50 \
60 \  druk nu de wortels af
70 for INDEX = 1 to 12
80 print "De wortel uit "; INDEX;" is "; A( INDEX)
```

```
 90 next INDEX
100 end
```

Wanneer programmadelen meerdere malen in een programma worden gebruikt is het handiger om zo´n programmadeel als apart stuk te schrijven. Zo´n programmadeel kan dan afzonderlijk getest worden en vanuit een groter programma gebruikt worden. Zo´n programmadeel heet onderprogramma of ´subroutine´. BASIC kent subroutines met de aanroep ´gosub´. De subroutine zelf moet als laatste instructie de ´return´ instructie hebben.

bv
```
10 for INDEX = 0 to 100
20 gosub # PRNT
30 next INDEX
40 end
60 \ nu volgt de subroutine PRNT
50 # PRNT : print " De wortel uit "; INDEX;
70 print "is "; sqr( INDEX)
80 return
```

Wanneer binnen een programma een hoeveelheid getallen nodig is, bijvoorbeeld voor een tabel dan kan dit met behulp van het ´data´-statement worden gedaan. Willen we het volgende getal inlezen, dan schrijven we:

```
10 repeat read NUMBER
20 print NUMBER
30 until NUMBER = -1
40 \
50 data 24,66,12,8690,1,-1
60 end
```

Nu worden de getallen van regel 50 stuk voor stuk binnen gelezen en afgedrukt totdat het getal -1 is gelezen.
Wanneer we herhaalde malen de getallen willen lezen programmeren we voor het gebruik een ´restore´ instructie. Hierdoor weet de processor dat bij de volgende read-instructie het eerste getal moet worden gelezen.

bv
```
 10 for INDEX = 0 to 5
 20 restore : gosub # PRNTDAT
 30 next INDEX
 40 end
 50 # PRNTDAT : repeat read NUMBER
 60 print NUMBER
 70 until NUMBER = -1
 80 return
 90 \
100 data 24,66,12,8690,1,-1
```

Het werken met teksten is een van de sterkste kanten van de taal BASIC. We kunnen met teksten net zo werken als met getallen bv:

```
TEKST$="dit is een tekstregel"
```

Het ´$´-teken achter de variabelenaam geeft aan dat het om een ´string´-variabele gaat. Een string mag tot een maximum van 255 tekens bevatten. Willen we weten wat in een string staat, dan drukken we dit af, net als bij getallen:

```
  print TEKST$
dit is een tekstregel
```

Operaties die op strings mogelijk zijn:

+	voeg twee strings achter elkaar
left$	neem het linker deel van een string
mid$	neem het middendeel
right$	neem het rechterdeel

bv:

```
10 read TEKST$
20 for I=1 to len( TEKST$)
30 print left$( TEKST$,I)
40 next I
50 \
60 for len( TEKST$) to 1 step -1
70 print spc( len( TEKST$-I)); right$( TEKST$,I)
80 next I
90 \
100 for I=1 to len( TEKST$)
110 print spc( I); mid$( TEKST$,I,1)
120 next I
130 data "Dit is een voorbeeld van een tekst"
140 end
```

HOOFDSTUK 3 COMMANDOS en INSTRUCTIES

De grammatika van de taal **BASIC** wordt hieronder per instructie behandeld.

clear zet alle variabelen op nul en resets het interne besturingssysteem.
Clear mag niet in een programma staan.

cont vervolgt het programma nadat de computer gestopt is ten gevolge van een ESC en CNTL/C of na een stop te hebben uitgevoerd. Cont kan niet worden gebruikt als niet door een van de genoemde voorwaarden is gestopt. Dan wordt de error: "can't continue" gegeven.

list list <start regelnummer - eind regelnummer>
maakt een listing van het programma. Optioneel kan 1 regel worden gelist . . . . . list 100
of vanaf een regel . . . . . list 100-
of tot een regel . . . . . list -100
of van/tot . . . . . list 25-100
of het gehele programma . . . . . list
Voor het maken van een listing op de printer dient eerst de printer 'ge-opend' te worden:
open 1,0 : list

load laadt een programma van cassette naar het geheugen.
Programma's worden op cassette opgeslagen met een ID-nummer (identificatie nummer) 1 t/m 254. Dit nummer moet achter load worden opgegeven.
bv load 1 . . laadt programma 1
load 204 . . laadt programma 204

save schrijft het programma wat in het geheugen staat naar de cassette onder het ID-nummer:
save 1
save 204
het programma blijft na het wegschrijven in geheugen staan.

new Hiermee wordt het geheugen geheel gewist. Het programma verdwijnt uit het geheugen.

run Hiermee wordt het programma dat in het geheugen staat gestart. Het is niet mogelijk een regelnummer of een label(naam) op te geven.
bv run . . start vanaf de eerste regel

auto auto <vanaf,stapgrootte>
hiermee worden automatisch regelnummers gegenereerd
Optioneel kan worden meegegeven vanaf welk nummer begonnen wordt en hoe groot de regelnummerstappen zijn.
bv auto 10,10 . . start: 10 , 20 , 30 . . .
auto . . idem als auto 10,10
auto 100 . . start: 100, 110, 120 . .

renum renum <van>,<naar>,<stapgrootte> (niet in versie 2.0)
Hernummert alle regels. Sprongadressen worden NIET bijgewerkt.
Het is hierdoor raadzaam renum alleen te gebruiken bij programma´s die met labels werken.
bv renum 100,500,5 . . hernummert vanaf regelnummer 100 naar 500 in stappen van 5

@ @ <startadres,aantal>
Hiermee wordt een stuk van het geheugen op het scherm afgedrukt. Het startadres en het aantal moeten worden opgegeven.
bv @ 0,20
4C 00 00 12 89 77 58 4A
12 0A 33 C7 4D AC 93 A2
45 6A C2 A1

upcase Na het ingeven van het commando ´upcase´ converteert BASIC alle letters in de listing naar hoofdletters. Dit omdat andere computers alles in hoofdletters doen, zodat onderlinge communicatie eenvoudig mogelijk is.

lowcase Na het intoetsen van het commando ´lowcase´ wordt de listing weer in grote letters (variabelen) en kleine letters (gereserveerde woorden) afgedrukt.

def Met deze funktie kunnen speciale funkties gedefinieerd worden. Deze funkties zijn vergelijkbaar met de sin, cos en sqr funkties en worden als volgt gedefinieerd:
def fnTOTAAL(A)=SOM+A
Achter ´fn´ staat de nieuwe funktienaam. Daarachter volgt de naam van het argument en dan de expressie van de funktie. De parameter (A) wordt een ´dummy´ argument genoemd omdat deze niet gebruikt wordt tijdens verwerking. Voor dit argument kan tijdens verwerking alles worden ingevuld. ´A´ zelf blijft onveranderd.
bv
```
10 def fn TOTAAL(A) = SOM+A
20 SOM = 100
30 FOR I=1 TO 20
40 print TOTAAL(I)
50 next I
60 print SOM,A
70 end
```
Na uitvoer van het programma zien we welke waarde SOM en A hebben.

dim Hiermee wordt aan BASIC verteld hoe groot een matrix moet zijn.
dim <variabele naam>(grootte-1, grootte-2 . . .)
Het aantal dimensies mag maximaal 120 zijn.
Na het dim-statement staan alle elementen van de matrix op nul.
```
10 dim REEKS(25,5)
20 for I=0 to 5
30 for J=0 to 25
40 REEKS(J,I)=25*I+J
50 next J
```

```
      60 next I
```

Nu staat de matrix geheel gevuld met steeds verschillende getallen.
voorbeelden van matrices:

```
dim TEKST$(100)          : \ ruimte voor 100 strings
dim TEL%(75,10)          : \ matrix van 75 x 10 integers
```

Belangrijk om te vermelden is dat een matrix automatisch wordt gedefinieerd als wordt ingetoetst:

```
MATRIX(3)=55
```

Nu is de matrix MATRIX met 11 elementen (0 t/m 10) gedefinieerd. Wanneer hierna een dim MATRIX(20) wordt gegeven volgt een foutmelding omdat de matrix al bestaat.
De ondergrens van het argument is nul (0), de bovengrens is de waarde die in het dim-statement wordt opgegeven. Hierdoor heeft een matrix altijd 1 element (per dimensie) meer dan bij het dim-statement is opgegeven.

end

Met ´end´ wordt de programmaloop gestopt en print BASIC ´ok´ op het scherm. Wanneer hierna ´cont´ wordt gegeven gaat de verwerking verder op de regel onder de ´end´-regel.
Het ´end´-statement is optioneel wanneer het de laatste regel van een programma is.

for . . next

Dit is de ´for-next´ lusstructuur.
for <variabele=expressie> to <expressie> (step <expressie>)
bv

```
for INDEX=36 to 72 step 3.5
    for I=100 to A-3 step -25
```

De variabelen INDEX en I worden elke keer dat de lus doorlopen wordt van waarde veranderd. De ´step´-grootte wordt er bij opgeteld. Indien geen ´step´-grootte is opgegeven is deze waarde 1. De programmalus wordt met ´next´ afgesloten.
bv

```
10 for I=100 to 55.3 step -0.5
   20 INT = SIN(I)*0.5 + INT
   30 next I
```

Achter ´next´ wordt de ´loopindex´ variabele gespecificeerd. Dit is optioneel, maar levert een controle op programmafouten die U anders moeilijk zou vinden.
Het is toegestaan met 1 ´next´-statement meerdere for-next lussen af te sluiten:

```
  100 next I,J,K
```

Dit is het zelfde als:

```
  100 next I
  110 next J
  120 next K
```

Belangrijk om te weten is dat de waarden van de loopindex en de ´step´-grootte slechts 1 keer worden berekend.

repeat . . until

Deze lusstructuur kent geen lus index maar de lus eindigt zodra de conditie achter ´until´ ´waar´ is. De lus wordt dus altijd minstens 1 keer doorlopen.
bv

```
10 repeat input "geef getal "; NUMBER
20 print sqr(NUMBER)
30 until NUMBER>2000
```

goto Een onvoorwaardelijke sprong wordt met het ´goto´-statement gedaan. Achter goto wordt opgegeven waarheen gesprongen moet worden. Dit kan met een ´label´ of met een regelnummer. In beide gevallen wordt na intoetsen van het ´run´-commando het adres van de regel (waar naartoe moet worden gesprongen) opgezocht en aan de goto-instructie toegevoegd. Hierdoor wordt een zeer snelle werking verkregen. Dit komt vooral tot uitdrukking bij grote programma´s.

```
bv  100 goto 25        : \ sprong naar een regelnummer
    110 goto #START    : \ sprong naar een label
```

gosub Hiermee wordt een ´onderprogramma´ aangeroepen. De gosub-instructie werkt net zoals de goto-instructie met dat verschil dat na het ´return´-statement in het onderprogramma de executie wordt hervat achter de gosub-instructie.

if . . goto

Hiermee wordt een conditionele sprong naar een regelnummer of label gedaan. Tussen ´if´ en ´goto´ staat een vergelijking.

```
if <expressie> goto <label of regelnummer>
bv  100 if REEKS(12)>0 goto 25       : \ naar regelnummer
    120 if TEST$="stop" goto #START : \ naar label
```

if . . then . . (else . .)

Hiermee worden, als de expressie die tussen ´if´ en ´then´ staat ´waar´ is, de statements achter ´then´ uitgevoerd.

```
bv  if A>0 then print "A is groter dan nul"
```

Het is tevens mogelijk op de zelfde regel ook nog een ´else´-deel op te nemen. Dit wordt uitgevoerd als de conditie niet ´waar´ is:

```
    if A>0 then print "A is groter" else print "A is kleiner"
of  if A>0 then goto #START else print "A is kleiner"
```

algemeen geldt:

```
if <condition>  then <statement> ( else <statement> )
```

let Het woord ´let´ geeft aan dat een toewijzing van een waarde aan een variabele gaat volgen. ´let´ mag echter ook weggelaten worden.

```
bv   10 let A=1
     20 B=23
```

In beide gevallen wordt een waarde-toewijzing gedaan. Het ´let´-statement heet ook wel het ´assignment statement´.

on . . goto

Een meerweg keuze door:

```
on <expressie> goto <label>,<label> . . . .
```

Voor de labels mogen ook regelnummers gebruikt worden. De waarde van de expressie bepaalt welk label (of regelnummer) genomen wordt. Het eerste label wordt genomen als de expressie 1 oplevert. Als de expressie een getal oplevert waarvoor geen label aanwezig is wordt op de volgende regel doorgegaan. Bij een getal >255 of <0 wordt de foutmelding ´illegal function´ gegeven.

```
bv.  10 for I=1 to 4
     20 on I goto #L1,#L2,#L3,#L4
```

```
30 stop
40 #L1 : print " L1" : goto #CNT
50 #L2 : print " L2" : goto #CNT
60 #L3 : print " L3" : goto #CNT
70 #L4 : print " L4"
80 #CNT : next I
```

on . . **go**sub
Dit werkt hetzelfde als de ´on . .goto´ behalve dat gesprongen wordt naar een onderprogramma (subroutine) dat afgesloten dient te zijn met ´return´. Nadat de ´return´ is uitgevoerd wordt het volgende statement na de ´gosub´ uitgevoerd.

rem Het ´rem´-statement (remark = opmerking) biedt de mogelijkheid commentaar op te nemen in het programma. Vanaf het woord ´rem´ tot het einde van het statement (´:´ of einde van de regel) wordt overgeslagen.
bv 10 rem dit is een regel die wordt overgeslagen
20 rem terwijl deze wordt overgeslagen tot aan : goto #START

De ´back-slash´ is het zelfde als het woord ´rem´.

restore Wordt gebruikt om de ´data-statements´ opnieuw te lezen. Na ´restore´ wordt met ´read´ het eerste data-statement van het programma gelezen.

return Zorgt voor de terugkeer naar het statement achter de laatste ´gosub´-instructie.

stop Onderbreekt de programmaverwerking en meldt dit met de tekst Break in xxxx , waarbij xxxx het regelnummer van de ´stop´-instructie is.
bv
```
100 print "Hallo"
110 stop

run                          <-- ingetoetst
Hallo

Break in 110
ok
```

Na deze melding kan met ´cont´ worden doorgegaan. In dit voorbeeld zijn er echter geen instructies meer na regel 110.

usr Hiermee kunnen assemblerroutines worden aangeroepen.
Met de usr(X) instructie wordt een argument meegegeven aan de assemblerroutine.
<variabele> = usr(expressie)
De assemblerroutine kan op zijn beurt weer een getal achterlaten voor BASIC.
bv A = usr(12+X*3)
De waarde die in de Floating-point accu achter wordt gelaten door de machinetaal routine wordt aan de variabele toegekend.
Het adres van de floating-point accu is &A9-&AE (169-174).
Het adres van de machinetaal routine moet voor het gebruik van het ´usr´ commando op adres 4 (low-byte) en 5 (high-byte) worden geplaatst (bv met een dpoke).

sys Met de ´sys´-instructie kunnen machinetaal routines direkt worden aangeroepen:
<variabele> = sys(<adres>,<AY-registers>,<X-register>)
Het meegeven van het A,Y en X register is optioneel.
Het A-register en het Y-register worden in 1 16-bit woord samengevoegd. De samenvoeging gaat als volgt:

```
10 A = &12
20 Y = &C0
30 X = &A9
40 P = sys(&F862,A*256+Y,X)
```

De waarde die door de routine in de registercombinatie AY wordt (A=high-byte, Y=low-byte), wordt aan de variabele toegekend.
De AY-waarde kan op 2 manieren worden overgedragen:
- unsigned 16-bit number Carry moet 0 zijn.
- signed 16-bit number Carry moet 1 zijn.

wait wait <adres>,<masker> (,<select>)
Hiermee wordt op een bepaalt bitpatroon gewacht dat op het opgegeven adres moet verschijnen. De instructie leest de inhoud van het adres, doet een ´exclusive or´ met de ´select´-waarde en vervolgens een ´and´ met het masker. Deze bewerking wordt net zo lang herhaald totdat het resultaat ongelijk aan nul is. Als geen ´select´-waarde is meegegeven wordt hiervoor de waarde nul genomen.
bv wait &E010,&80,&80
Deze instructie wacht totdat PB7 van de VIA laag is.

IN en UITVOER

data data <reeks gegevens>
Met het data-statement kunnen vaste programmagegevens worden ingevoerd in het programma. Voor de programma-loop is het data-statement het zelfde als een rem-statement. bv:

```
10 data 1,3,5,7,11,13,-1
20 repeat read A
30 print A
40 until A<0
50 end
```

Dit programma drukt de getallenrij van het data-statement op het beeldscherm af.
In het data-statement mogen in plaats van getallen ook teksten worden opgegeven:
data "Dit is een data-statement tekst"

Er mag een onbeperkt aantal data-statements in een programma staan.

get Haalt een karakter van het toetsenbord en zet dit karakter in de variabele:

```
10 #start : get A$
20 print A$
30 goto # start
```

Het zelfde kan ook met getallen, vervang dan de A$ door A.
Wanneer nu echter een letter wordt ingetoetst geeft BASIC een syntax-error.

input Haalt getallen of teksten van het toetsenbord.
bv.
```
10 input "Naam: "; NAME$
20 input "Adres:"; ADRESS$
30 input "Stad: "; CITY$
40 print:print NAME$,ADRESS$,CITY$
50 end
```
Ook getallen worden op deze manier binnengelezen:
```
10 input "getal A : "; A
```
De tekst achter het woord input wordt op het scherm afgedrukt, waarna het getal wordt binnengelezen.
Wanneer de informatie van het active input-apparaat moet worden gehaald, moet gespecificeerd worden welk input-apparaat bedoeld wordt:
```
input ! 3, LINE$
```
Dit haalt een regel tekst van input-apparaat 3.

read Werkt identiek als input, maar alle gegevens worden van de data-statements gelezen. Uiteraard mogen na het woord read geen teksten staan, alleen variabelen.

print print <variabelen-lijst>
Het print-statement dient als algemene gegevensuitvoer voor het programma. Hiermee kunnen numerieke- en tekstvariabelen worden afgedrukt. Voor het ´netjes´ afdrukken van getallen is voorzien in een format-specificatie.
```
Normaal      :   print A, B, 2, 5.00467
geformateerd:   print [ 5,2 ] A, B, 2, 5.00467
Met teksten :   print " hallo ", 3*7, A$
```
Bij het geformateerd afdrukken geeft het eerste format-cijfer het aantal velden voor de decimale punt aan en het tweede cijfer het aantal velden achter de decimale punt. Getallen worden niet afgerond, alleen de cijfers die in het veld achter de punt passen worden afgedrukt.
Wanneer de parameters van de print-instructie door een comma (,) worden gescheiden, wordt de indeling van de uitvoer door BASIC gekozen. Wanneer een punt-komma (;) wordt gebruikt worden door BASIC geen spaties toegevoegd. Tevens wordt na de print-instructie geen nieuwe regel gegeven als de print-instructie be-eindigd is met een punt-komma.
bv.
```
10 print "hallo ";
20 print "daar"
30 end
run
hallo daar
ok
```

spc
```
print spc(12);"Hallo"
```
In het print-statement kan een aantal spaties worden gegeven met het spc(x) statement. Hierachter moet echter een ´;´ staan.

```
print tab(30);"Hallo"
```
Tijdens het printen wordt nu naar de 30-ste positie getabuleerd.

inkey Leest het toetsenbord. Wanneer geen toets is ingedrukt is de variabele nul (indien een string-variabele, een lege string).
bv:
```
10 \ ECHO-PROGRAMMA
20 #MAIN : INKEY A$
30 if A$<> "" then print A$
40 goto #MAIN
```
(werkt niet op de PDS-65).

open Voor het lezen of schrijven van informatie van/naar andere kanalen dan het toetsenbord en het beeldscherm kan een ander kanaal 'geopend' worden.
open <dev#>,<I/O>(,<id#>)

```
          |      |      \___ TAPE ID-number
          |      \__________ I = input,  O = output
          \_________________ 0 = console
                             1 = printer
                             2 = serial port
                             3 = tape (PC-2)
```

close Wanneer de gewenste informatie is geschreven/gelezen, dient het kanaal weer gesloten te worden.
close <I/O>

```
        \_______________ I = input,  O = output
```

bv Het maken van een listing op de printer:
```
open 1,0 : list : close 0
```

Voor het PDS-65 basic geldt dat dev#=3 is input/output met de disk. In plaats van het ID-number wordt dan de filenaam en het drivenummer opgegeven:
```
10 \ LIST A DISK-FILE
20 input "Filename  : "; A$
30 input "Drivenr   : "; D
40 open 3,I,A$,D
50 #LL1 : input ! 3, B$
60 print B$
70 if B$<>"" then goto #LL1
80 close I
```

peek peek (adres)
Hiermee kan de inhoud van een geheugenlocatie worden uitgelezen.
bv A = peek (&1000) haalt de inhoud van $1000 naar A.
```
10 \ MEMORY-DUMP
20 repeat print peek(X);
30 until X>&0100
40 end
```

dpeek Indem als peek, maar nu wordt de 16-bits inhoud van een adres opgehaald (2 posities uit het geheugen).

poke poke <adres>,<data>
Hiermee kunnen getallen (0..255) in het geheugen worden gezet. bv poke X,Y
Het adres (X) dient tussen 0 en 65535 te liggen. Wanneer een van de getallen buiten de aangegeven waarde ligt, wordt een function error gegeven.
LET OP: Met deze instructie kan het geheugen direct worden veranderd. Het is hiermee mogeljk dat Uw programma beschadigt, zodat het niet meer correct werkt.

dpoke Idem als poke, doch nu worden steeds 16-bits (2 geheugen-locaties) beschreven. Het getalbereik van de ´data´ is nu 0..65535.

TEKST BEHANDELING (STRINGS)

asc asc(A$) Geeft de getalrepresentatie van de (eerste) letter van de string A$. bv:
```
print asc("ABC")     --> 65
print asc(mid$("ABC",2))  -->  66 (="B")
```

chr$ chr$(A) Geeft de letterrepresentatie van het getal (A).
```
bv  print chr$(65)      -->  A
    A$ = chr$(65)
```

left$ left$(A$,X) Geeft de meest linkse ´X´ letters van A$:
```
    print left$("ABCDEFG",4)  -->  ABCD
```

right$ right$(A$,X) Geeft de meest rechtse ´X´ letters van A$:
```
    print right$("ABCDEFG",4) -->  DEFG
```

mid$ mid$(A$,X,Y) Geeft de letters van A$ vanaf positie ´X´.
Het aantal letters wordt door ´Y´ bepaald. Indien Y wordt weggelaten geeft mid$ alle letters vanaf positie ´X´.
```
bv  print mid$("ABCDEFG",3,2)   --> CD
    print mid$("ABCDEFG",3)     --> CDEFG
```

len len(A$) geeft de lengte van A$ weer:
```
print len("ABCDEFG")            --> 7
X = len(A$)
```

val val(A$) Geeft het getal weer wat in de string staat:
```
print val("3.14")     --> 3.14
```
bv:
```
10 A$ = "-1234.8863"
20 print val(A$)*2
30 end
```
Met ´val´ kunnen ook input-strings van hexadecimale getallen geconverteerd worden bv:
```
10 input A
20 A$="&"+A$
30 print "value ="; val(A$)
```

str$ str$(X) Maakt van het getal X een string welke afgedrukt kan worden:

```
10 A$ = str$(1200.7)
20 print A$
30 end
```

hex$ hex$(X) Idem als str$ maar maakt er een hexadecimale getal-representatie van:

```
print hex$(12)       -->  0C
```

whex$ idem maar dan steeds 4 cijfers:

```
print whex$(12)      -->  000C
print whex$(1200)    -->  04B0
```

REKENFUNKTIES

abs(X) Geeft de absolute waarde van X:

```
Y = abs(X)
```

sgn(X) geeft het teken van het getal X:

```
Als X > 0     --> +1
   X = 0      -->  0
   X < 0      --> -1
print sgn(12)     --> 1
print sgn(-5)     --> -1
```

exp(X) Geeft de waarde van de machtsverheffing van E (=2.71828183) met X:

```
Y = exp(X)
print exp(1)        --> 2.71828183
```

log(X) Geeft de natuurlijke logaritme (basis E) van X:

```
Y = log(X)
print log(exp(1))   --> 1
```

int(X) Geeft het grootste gehele getal (integer) dat kleiner of gelijk is aan X.

```
print int(5.4)          -->  5
print int(5.9)          -->  5
print int(-0.8)         --> -1
print int(-0.2)         --> -1
```

sqr(X) Geeft de vierkantswortel van het getal X.

sin(X) cos(X) tan(X) atn(X)
Geven de waarde van de betreffende goniometrische funkties. Alle hoeken worden in radialen opgegeven.

rnd(X) Geeft een getal tussen 0 en 1 bedoeld als willekeurig getal. De parameter X beinvloed het willekeurige getal:

X<0 start een nieuwe reeks willekeurige getallen met gebruikmaking van X.
X>0 geeft een nieuw willekeurig getal
X=0 geeft opnieuw het laatst gegeven getal.

HOOFDSTUK 4 DE SCREENEDITOR (PC-2/3)

Het woord ´screen-editor´ zegt letterlijk dat (tekst) op het scherm wordt ´ge-edit´ (editen = bewerken).
Dit editen is van belang bij het testen en corrigeren van een programma. Hiervoor zijn vaak veel wijzigingen in de programmatekst nodig. Normaal wordt een regel waarin iets moet veranderen geheel opnieuw ingetoetst. Dit kost veel tijd en brengt het risico van nieuwe (type-) fouten met zich mee.
Daarom is de PC-2 en PC-3 voorzien van een screeneditor.
Wanneer in een regel iets moet worden veranderd drukt U die regel (m.b.v. het list-commando) op het scherm af, gebruikt de toetsen met de pijltjes om de cursor op de regel te plaatsen en verbetert de regel door het foutieve deel over te typen. Met de ´char insert´ en de ´char delete´ toetsen kan de tekst achter de cursor worden verschoven. Wanneer de wijzigingen zijn aangebracht dient een <return> gegeven te worden om de regel van het scherm naar het geheugen van de computer te brengen.
Alle video-scherm commando´s (zoals gegeven in appendix A) kunnen gebruikt worden tijdens screen-editing. Handig is bv het commando ESC T (wis van de cursor tot aan het einde van de regel).
Let op: HOOFDLETTER T

APPENDIX A

De video commando´s PC-2

commando	toets code	BASIC instructie(s)
Clear screen	CTRL/Z	print chr$(26);
Cursor home	CTRL/^	print chr$(30);
Erase line from cursor	ESC T	print chr$(27);"T";
Start inverse	ESC J	print chr$(27);"J";
End inverse	ESC K	print chr$(27);"K";
Start blinking	ESC ^	print chr$(27);"^";
End blinking	ESC Q	print chr$(27);"Q";
Start underline	ESC L	print chr$(27);"L";
End underline	ESC M	print chr$(27);"M";
Start blank	ESC [	print chr$(27);"[";
End blank	ESC]	print chr$(27);"]";
Double height	ESC V	print chr$(27);"V";
Single height	ESC W	print chr$(27);"W";
Double width	ESC @	print chr$(27);"@";
Single width	ESC A	print chr$(27);"A";
Start grafic	ESC F	print chr$(27);"F";
End grafic	ESC G	print chr$(27);"G";
RED on	ESC CTRL/@	print chr$(27);chr$(0);
GREEN on	ESC CTRL/A	print chr$(27);chr$(1);
BLUE on	ESC CTRL/B	print chr$(27);chr$(2);
RED off	ESC CTRL/C	print chr$(27);chr$(3);
GREEN off	ESC CTRL/D	print chr$(27);chr$(4);
BLUE off	ESC CTRL/E	print chr$(27);chr$(5);
Zet cursor	ESC = <Y> <X>	print chr$(27);"=YX"

Bij de cursor adressering worden de verticale en horizontale posities in ASCII opgegeven: een spatie is ´0´, "!" is ´1´ enz (zie ASCII tabel).

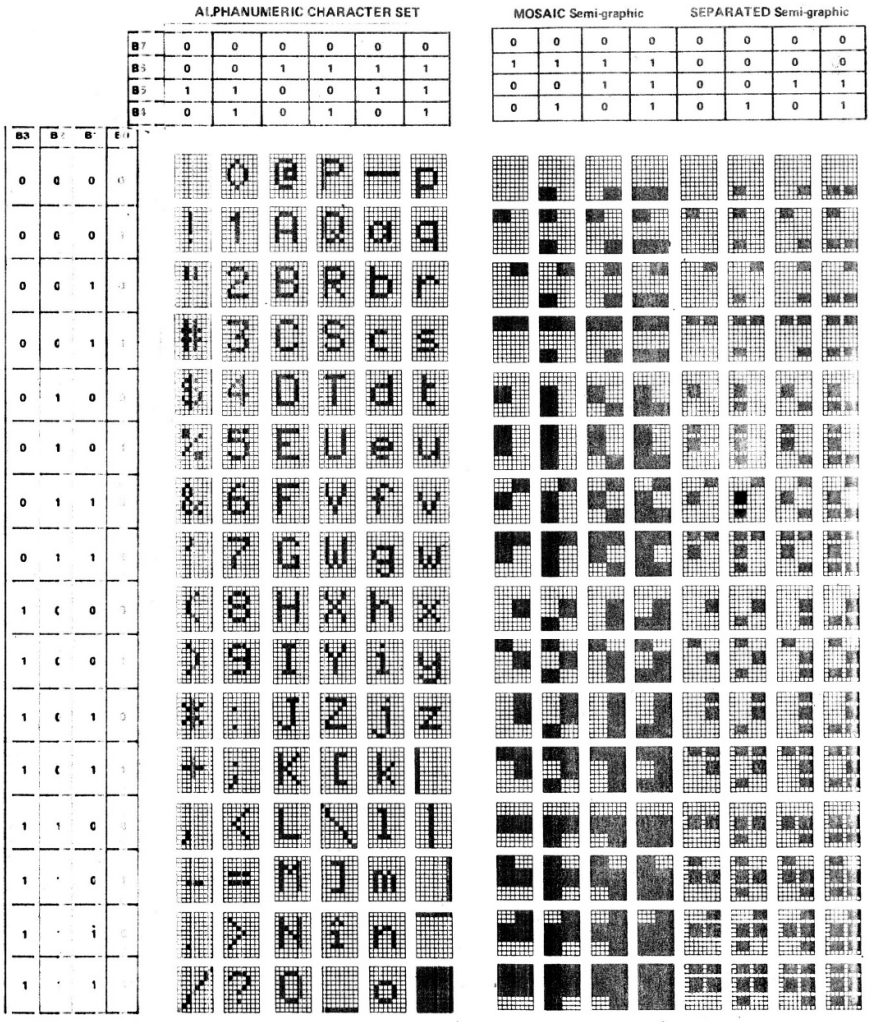
Wanneer videocommando´s vaak gebruikt worden in een programma is het handig ze in een ´string´ te plaatsen bv:

```
10 INV$ = chr$(27)+"T"   \ start inverse
20 NINV$ = chr$(27)+"K"  \ end   inverse

50 print INV$; "Hallo daar"; NINV$
```

APPENDIX B

De karakterset van de PC-2 videokaart (40-koloms)



APPENDIX C

De input/output apparaten

Invoer en Uitvoer apparaten kunnen vanuit BASIC worden bediend door het te ´openen´ en na gebruik te ´sluiten´. Er zijn 4 kanalen voorzien:

kanaalnr	´input´-apparaat	´output´-apparaat
0	toetsenbord	beeldscherm
1	--	printer (Ramboard)
2	serieel	serieel
3	tape (disk)	tape (disk)
4	--	printer (VIA)

Tussen haakjes zijn de PDS-65 apparaten aangegeven.
Zie voor het openen en sluiten van een apparaat hoofdstuk 1.

APPENDIX D

De ASCII karakter tabel

ASCII TABEL

b4	b3	b2	b1	Row	000	001	010	011	100	101	110	111
				Column	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	CHR INS	!	1	A	Q	a	q
0	0	1	0	2	STX	CHR DEL	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS←	CAN	(	8	H	X	h	x
1	0	0	1	9	SKIP HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF↓	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT↑	ESC	+	;	K	[	k	{
1	1	0	0	12	FF→	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	HOME RS	.	>	N	^	n	~
1	1	1	1	15	SI	NEW LINE US	/	?	O	_	o	DEL RUB

APPENDIX E

Adressen van BASIC voor gevorderde programmeurs

adres		funktie
hex	**decimaal**	
00	0	´escape´ jmp na ´eol´ en ´:´
03	3	jmp to user function
12	18	1 byte met de ´width´
A9	169	floating point accu: <exp(1)>,<mantisse(4)>,<sign(1)>

.END

Bij de documentatie van het PC-2 BASIC is per abuis het volgende niet correct gedocumenteerd:

- 'print'-instructie:
  Wanneer programma 'output' naar de printer of een ander kanaal moet worden gestuurd moet eerst dat kanaal 'ge-opend' worden met de 'open' instructie. Vervolgens moet de print-instructie voorzien worden van een code die aangeeft naar welk kanaal de gegevens moeten worden verstuurd.

```
bv.  10 open 1, 0
     20 print @ 1,"Hallo, dit is de printer"
     30 close 0

of:  10 input "Devicenr"; dev
     20 open dev, 0
     30 print @ dev, "Hallo dit is een test"
     40 close 0
```

- 'input'-instructie
  Hiervoor geldt hetzelfde als voor de print-instructie:

```
bv.  10 input "Inputdevice";dev
     20 open dev, I
     30 input A$
     40 print A$
     50 close I
```

- De intructies 'get' en 'inkey' zijn op de PC-2 BASIC niet voorzien in verband met de methode van toets-aftasting van de PC-2.

.END