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.

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 'braek'
list	list het programma
load	laad 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

STATEMENTS

def fn	definieer een funktie
dim	definieer een matrixgrootte
end	beeindigd het programma
for . . . next	programmalus met een lus-teller
goto	spring naar een andere regel
gosub	voer een onderprogramma uit
if . . goto	spring als de conditie 'waar' is
if . . then . .	voer uit als de conditie 'waar' is
let	geeft een variabele een waarde
on . . goto	meerweg splitsing
on . . gosub	meerweg splitsing voor onderprogramma's
rem	zet een 'remark' regel (comentaar)
\	een 'remark' regel (comentaar)
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 on 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
poke	zet iets in een geheugenlocatie

TEKT 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
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
( )	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'

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 U een integer
getal bedoelt.
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 slecht 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 vergelijkingoperatie 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
```

REEKS-FUNKTIES

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 getalleen 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 ook subroutines met de aaroep '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
```

eer 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 string mogelijk zijn:

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

De grammatica van de taal BASIC wordt hieronder per instructie behandeld.

clear zet alle variabelen op nul en resets het interne besturingssysteem.
Clear mag niet een 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 een 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. Optioneel kan een regelnummer of een labelnaam worden opgegeven welke aangeeft waar gestart dient te worden.
bv run . . start vanaf de eerste regel
run 100 . . start vanaf regel 100
run #start . . start vanaf label start

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 hernummert alle regels. Sprongadressen worden NIET bijgewerkt. Het is hierdoor raadzaam renum alleen te gebruiken bij programma's die met labels werken.

@ @ <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 communucatie kan worden gedaan.

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 funktie 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 dimenties 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 matrixen:

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

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

```
MAXTRIX(3)=55
```

Nu is de maxtrix MATRIX met 11 elementen (0 t/m 10) gedefinieerd. Wanneer hierna een dim MATRIX(20) wordt gegeven volgt een foutmelding omdat de maxtrix 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.

goto

Een onvoorwaardelijk 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 TESK$="stop" goto #START : \ naar label

if . . then
Hiermee kan als de expressie die tussen 'if' en 'then' staat 'waar' is de statements achter 'then' worden 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 weg-gelaten worden.
bv 10 let A=1
20 B=23
In beide gevallen wordt een waarde-toewjzing 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.
Het waarde van de expressie bepaalt welk label (of regel-nummer) genomen wordt. Het eerst 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 een foutmelding 'illegal funktion' 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 . . gosub
Dit werkt hetzelfde als de 'on . .goto' behalve dat gesprongen wordt naar een onderprogramma (subroutine) die 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 comentaar op te nemen in het programma. Vanaf het woord 'rem' tot het einde van het statement (':' of einde van de regel) overgeslagen.
bv 10 rem dit is een regel die wordtovergeslagen
20 rem terwijl deze wordt overgeslagen tot aan : goto #START

De 'back-slash' is het zelfde als de 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 Stopt de programmaexecutie 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.

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 in de Floating-point accu achter wordt gelaten door de machinetaal routine wordt aan de variabele toegekend.

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.

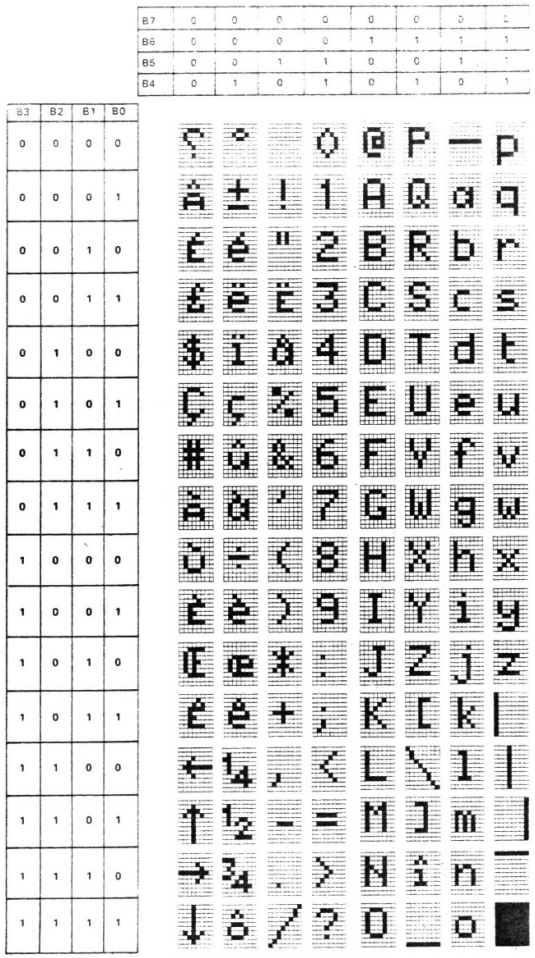
PROTON ELECTRONICS
Energiestraat 36, 1411 AT NAARDEN

Tel. 02159 - 4 82 24*
Telex 73415
Amro Bank, Hilversum
Rek.nr. 41.67.63.162
Giro van de bank 32750

PC-2 Basic video commands

Clear screan	CTRL/Z	CHR$(26)
Home	CTRL/↑	CHR$(30)
Erase till eol	ESC/T	CHR$(27);"T"
Start inverse	ESC/J	
End ,,	ESK/K	
Start blink	ESC/↑	
Double Heigth	ESC V	
Single ,,	ESC W	
Double Width	ESC @	
Single Width	ESC A	
Start grafic	ESC F	
Stop grafic	ESC G	
Red on	ESC CTRL/@	CHR$(½&);CHR$(0)
Green on	ESC CTRL/A	CHR$(27);CHR$(1)
Blue on	ESC CTRL/B	CHR$(27);CHR$(2)
Red off	ESC CTRL/C	CHR$(27);CHR$(3)
Green off	ESC CTRL/D	CHR$(27);CHR$(4)
Blue off	ESC CTRL/E	CHR$(27);CHR$(5)
Start underline	ESC L	
Stop ,,	ESC M	
Start blanking	ESC [	
End ,,	ESC]	

ALPHANUMERIC CHARACTER SET



NOTA : Black dot = output high (GEN)

MOSAIC Semi-graphic SEPARATED Semi-graphic