DAI NAMIC

18

tweemaandelijks tijdschrift september - oktober 1983

T. Berkx

personal computer users club

een uitgave van dainamic v.z.w.
verantw. uitgever w. hermans, heide 4 - 3171 westmeerbeek

International

DAI NAMIC

PERSONAL COMPUTER USERS CLUB

4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	54344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

belangrijke ASCII-waarden in DAIpc

functie/symbool	HEX	DEC
back-space	8	8
TAB	9	9
linefeed	A	10
clear screen	C	12
CURSOR UP	10	16
CURSOR DOWN	11	17
CURSOR LEFT	12	18
CURSOR RIGHT	13	19
space-bar	20	32
Ø	30	48
A	41	65
a	61	97
pijltje rechts	89	137
pijltje links	88	136
pijltje boven	5E	94
pijltje onder	8C	140
volle blok	FF	255
verticale lijn	A	10
horizontale lijn	B	11
6 hor. lijnen	1D	29

ASCII - HEX - ASCII CONVERSION TABLE

LSD	MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

COLOFON

DAInamic verschijnt tweemaandelijks.
Abonnementsprijs is inbegrepen in de jaarlijkse contributie .
Bij toetreding worden de verschenen nummers van de jaargang toegezonden.
DAInamic redactie :

Dirk Bonné
Freddy De Raedt
Wilfried Hermans
René Rens
Jos Schepens
Roger Theeuws
Bruno Van Rompaey
Jef Verwimp

Vormgeving : Ludo Van Mechelen.

U wordt lid door storting van de contributie op het rekeningnr. **230-0045353-74** van de **Generale Bankmaatschappij, Leuven,** via bankinstelling of postgiro
Het abonnement loopt van januari tot december.

DAInamic verschijnt de pare maanden.
Bijdragen zijn steeds welkom.

CORRESPONDENTIE ADRESSEN.

Redactie en software bibliotheek

Wilfried Hermans
Heide 4
B 3171 Westmeerbeek
België
tel. : 016/69.86.23

Vanaf 1 nov. '83 :
Mottaart 20
3170 Herselt
Tel. 014/54 59 74

Kredietbank Westmeerbeek
nr. 406-3016141-33
BTW : 420.840.834

Lidgelden

Bruno Van Rompaey
Bovenbosstraat 4
B 3044 Haasrode
België
tel. : 016/46.10.85

Voor Nederland :

GIRO : 4083817
t.n.v. J.F. van Dunne'
Hoflaan 70
3062 JJ ROTTERDAM
Tel. : (010) 144802

Generale Bankmaatschappij Leuven
nr. 230-0045353-74

Inzendingen : Games & Strategy

Frank Druijff
's Gravendijkwal 5A
NL 3021 EA Rotterdam
Nederland
tel. : 010/25.42.75

Westmeerbeek, oktober 83

Beste leden,

Als alles verloopt zoals gepland,is dit het laatste nummer dat in Westmeerbeek samengesteld wordt. De redactie, in casu ondergetekende, wordt momenteel namelijk belast met verbouwing en verhuizing. Waarschijnlijk hebben sommigen dit reeds tot ergernis bemerkt aan de onbeantwoorde telefoonoproepen , onze excuses , wij hopen dat binnen een aantal weken de toestand alweer gestabiliseerd is. Vanaf +/- 1 november bevindt DAInamic zich op volgend adres : Mottaart 20 , 3170 Herselt , en heeft de PTT ons bedacht met dit telefoonnummer : 014/545974 . Een aantal feiten hebben onze clubactiviteiten de laatste weken nog bemoeilijkt : - de poststaking in Belgie, - het enorme succes van onze verjaardagsaanbieding (er moesten dagelijks 10 to 30 pakketten de deur uit !), - het chronische gebrek aan DCR-cassettes . Bovendien schijnt PHILIPS een partij cassettes op de markt gebracht te hebben die het label "betrouwbaar" beslist niet verdienen. Deze partij was voorzien van oranje wieltjes, wat beslist niet wil zeggen dat we alle cassettes met oranje wieltjes moeten wantrouwen ... Heeft U van ons DCR-cassettes ontvangen die moeilijkheden geven, dan kunnen deze steeds geruild worden. Geef ons echter enige tijd, want onze voorraad is weer totaal uitgeput ! Van het floppy-front is er voorlopig nog geen nieuws te melden: noch INDATA, noch KEN-DOS hebben ons een compleet werkend systeem ter test kunnen aanbieden. Met een 200-tal DCR-cassettes in gebruik wordt het ondertussen op de redactie erg moeilijk om in het software-bos de programmaboompjes terug te vinden: de nood aan een degelijk floppy-systeem is erg groot ! Wij wensen de heren ontwerpers moed en succes (op korte termijn graag !). Als dit nummer verschijnt bevinden we ons weer in de gezellige drukte van de HCC-dagen in Utrecht, daar wordt weer bewezen dat onze hobby maatschappelijk in het licht van de schijnwerpers staat,wie had een paar jaar terug kunnen denken dat het zo'n vaart zou lopen ?
Met de aankondiging van GAMES COLLECTION 12 sluiten we dit voorwoord, tot een volgende keer,

GAMES COLLECTION 12

(Chase, shoot and hunt ...)

Duck shooting by Hans Peters
Slangenjacht by Hendrik-Jan van Randen
Deer Hunt by Luc Maes
Cobra by Fred van Amerongen
Star-hunt by Koert van Espen

price :audio :750 Bfr , DCR :900 Bfr
available : 1/11/83

Dear members,

Dainamic will be on a new address from 1st of november : Mottaart 20, 3170 Herselt, the new phone number will be : 014/545974. It seems that PHILIPS has produced a lot of DCR-cassetts that are not 100 % reliable , the cassettes have orange wheels. If you have troubles whit tapes you got from us, please return for exchange ! INDATA-floppy system is not ready , KEN-DOS is not ready : will there never come an end to our growing collection of DCR-cassettes ?
Above you can find the contents of GAMES COLLECTION 12 .

W.Hermans

NEWSLETTER 18

INHOUD — CONTENTS

DAInamic subscription rates :

Benelux	:	900 Bfr
Europe	:	1000 Bfr
Outside Europe (Air Mail)	:	1400 Bfr

pay to : Dainamic SUBSCRIPTIONS
B.Van rompaey
Bovenbosstraat 4
3044 HAASRODE-BELGIUM

* by check or
* on Bancaccount nr 230-0045353-74 of Generale Bank Leuven c/o DAInamic

PROGRAMMEERTECHNIEKEN

Half augustus kreeg ik een band binnen van Koert van Espen die mij zeer interresseerde. De band was keurig voorzien van de testarray's om inlezen te vergemakkelijken en het programma zelf stond er ook meermalen op, keurig !
In de begeleidende brief vertelde Koert de inhoud van de band en verklaarde dat hij STARBUILDER volledig zelf had verzonnen. Dit wekte vanzelfsprekend mijn belangstelling en ik laadde het programma dan ook snel in. Het bleek een eenvoudig maar aardig actiespel te zijn. Een programma dat, hoewel het vrij klein is, toch heel wat te bieden heeft. De bestudering van de listing overtuigde mij ervan dat dit inderdaad origineel werk van Koert is. Er zaten weliswaar wat 'foutjes' in, maar het geheel en bepaalde details toonden inzicht. Mogelijke problemen werden onderkend en de oplossing daarvan meestal gevonden. Aan de andere kant waren de foutjes en aanpak soms zo illustratief voor de gedachtengang van vele programmeurs (en echt niet alleen beginners!) dat ik besloot Koert op te bellen om hem te vragen of ik zijn programma mocht gaan gebruiken als basis voor een van mijn artikelen. Koert stemde hiermee in wat ik niet alleen zeer waardeer maar ook bewonder. Hij weet dat ik in zijn programma van alles ga bekritiseren en dat dan nog eens ga laten publiceren ook. Kritiek zeker indien die opbouwend bedoeld is kan prettig zijn maar om ten overstaan van alle DAInamic-lezers (ik hoop tenminste dat ze het lezen) je te laten bekritiseren getuigt van moed.
Ik hoop dan ook dat anderen zich de moeite willen getroosten om Koerts versie te bestuderen en dan zelf eens een lijstje te maken met foutjes die zij in dit programma zien of menen te zien en dan pas verder te lezen.
Daarnaast zijn er natuurlijk ook nog smaakgevoelige punten en mogelijk zelfs punten die ik over het hoofd heb gezien. Zou ik inderdaad iets belangrijks over het hoofd hebben gezien; laat mij het dan weten, ik kan er in de toekomst rekening mee houden. En als ik in een bui even moedig als Koert ben geef ik die onvolkomenheid nog toe ook.

Het oorspronkelijke programma van Koert :

```
10    REM Program by Koert Van Espen   Date:1983 08
20    GOSUB 10000
30    PRINT "Do you want to play - without red bricks (type '1')?"
40    PRINT "                    - with red bricks    (type '2')?"
50    G!=GETC:IF G!<>49.0 AND G!<>50.0 GOTO 50
100   FOR B=1 TO 3
110   PRINT CHR$(12):MODE 2A
120   CURSOR 0,2:PRINT "SCORE:    ";S:CURSOR 30,2:FOR W=1 TO 3-B:IF B<>
3 THEN PRINT CHR$(255);" ";
130   NEXT
140   DRAW 0,0 XMAX,0 3:DRAW 0,YMAX-14 XMAX,YMAX 3:DRAW 0,0 0,YMAX-14
3:DRAW XMAX,0 XMAX,YMAX-14 3
150   X=XMAX/2:Y=YMAX/2
160   DOT X,Y 5
170   E=30:R=15:FOR K=1 TO G!-48
180   FOR O=1 TO E:XX=INT(RND(2.0*X-2.0)+1.0):YY=INT(RND(2.0*Y-16.0)+1
.0):IF SCRN(XX,YY)=15 OR SCRN(XX,YY)=3 THEN O=O-1:NEXT
190   DOT XX,YY R:NEXT:E=15:R=3:NEXT
200   ENVELOPE 0 15,200;0,200;:IF T=0 GOTO 1000
300   H=SCRN(X,Y):IF H=3 GOTO 500
310   IF H=15 THEN S=S+100:T=T+1:SOUND 0 0 15 0 4545
320   IF T=30 GOTO 900:DOT X,Y 3:CURSOR 8,2:PRINT S;"  ":SOUND 0 0 15 1 36363
330   GG=GETC:IF GG=0 THEN 1100:G=GG:ON GG-15 GOTO 2000,3000,4000,5000
500   SOUND 0 1 15 0 64516:ENVELOPE 1 15,20;10,20;5,10;:WAIT TIME 100:SOUND OFF
510   G=0:GG=0:T=0:NEXT
520   PRINT "Again?(Y/N)"
```

```
530   A=GETC:IF A<>89 AND A<>78 THEN 530:IF A=78 THEN END
800   G=0:GG=0:T=0:S=0:GOTO 100
900   G=0:GG=0:T=0:S=S+5000:GOTO 110
1000  G=GETC:IF G=0 GOTO 1000
1100  S=S-1:ON G-15 GOTO 2000,3000,4000,5000
2000  Y=Y+1:GOTO 300
3000  Y=Y-1:GOTO 300
4000  X=X-1:GOTO 300
5000  X=X+1:GOTO 300
10000 PRINT CHR$(12):COLORT 7 0 0 0:COLORG 0 3 5 15:MODE 0
10010 CURSOR 19,18:PRINT "S T A R B U I L D E R "
10020 CURSOR 19,17:PRINT "- - - - - - - - - - - "
10030 CURSOR 19,15:PRINT "  KOERT VAN ESPEN"
10040 PRINT :PRINT :PRINT :PRINT "DO YOU WANT INSTRUCTIONS? (Y/N)"
10050 G!=GETC:IF G!<>89.0 AND G!<>78.0 THEN 10050:IF G!=78.0 THEN RETURN
10060 PRINT CHR$(12):PRINT "Try to catch the white stars as fast as po
ssible,":PRINT "but avoid anything that is red coloured."
 10070 PRINT "Use the cursorkeys to move your ship.":RETURN
```

Zoals U kunt zien is het een vrij klein programma (40 regels) en als U de moeite neemt om het in te tikken kunt U zien dat het bijna altijd goed werkt.
De enige fout in de werking (HEEFT U DIE GEVONDEN ????) komt slechts zelden aan het licht. Bijna altijd zult U normaal kunnen spelen.
Bij het bestuderen van de listing zal U zijn opgevallen dat als regelnummers alleen tientallen worden gebruikt; een uitstekende gewoonte, die mij vertelt dat niet alleen bij het programmeren zelf is nagedacht, maar ook nadien het programma nog eens bekeken werd voor het werd ingestuurd. Dit laatste ontbreekt bij vele inzendingen. Velen zijn tevreden zodra het programma werkt. Maar ik vind dat na het programmeren zelf altijd nog een of twee stappen horen. De beroepsprogrammeur zal na de laatste regel nog een handleiding / bedieningsaanwijzing of iets dergelijks moeten maken OOK als het programma zelf uitleg geeft. Ik zie bijvoorbeeld mensen aan de slag willen gaan met DAInatext. Op de eerste uitleg-pagina staat dat je met [SHIFT]+[RETURN] terug kunt komen in het menu, maar dat is niet leesbaar als je met de tekst bezig bent. Ook de aanwijzing dat je niet eerst [SHIFT] en dan [RETURN] moet drukken maar tegelijkertijd met eerst de [SHIFT] en dan pas de [RETURN] erbij ontbreekt. De amateurprogrammeur zal zijn programma in een vorm moeten vermaken dat hij er mee voor de dag kan komen. Het is toch uw liefhebberij en dan is het toch fijn iets moois af te leveren.
Het eerste is ook voor amateurs aan te bevelen, zeker als hun programma's voor anderen bedoeld zijn. Het tweede zullen beroepsmensen zich ook moeten aantrekken al kunnen die dit om economische redenen (een volgende opdracht wacht) achterwege laten. Overigens zou moeten gelden dat juist bij beroepsmensen de structurele opzet al zo groot is dat er weinig meer verbeterd kan worden.

Zo, en nu komen we dan bij de kritiek aan.

Zoals reeds eerder vermeld ben ik een tegenstander van het vragen of de gebruiker instructies nodig heeft als de uitleg zo kort is als hier. De intro / uitleg staat correct aan het eind van de listing met logische regelnummers. Maar deze subroutine heeft twee RETURNs en dat vind ik voor de structuur minder geslaagd. Het geheel kan trouwens ook goed met GOTOs gedaan worden. We gaan alleen vanaf regel 20 naar regel 10000 en keren dan altijd terug naar regel 100. Willen we dan later nog een CLEAR opnemen kan dat dan probleemloos. Met CLEAR is het programma trouwens beter: nu loopt het stuk na een CLEAR4 voor de RUN. CLEAR 256 is bv goed.
De regels 30,40 & 50 vind ik bij de uitleg horen, daar de keuze maar eenmaal in het programma wordt gemaakt.

Over regel 50 nog het volgende : beseft U dat
50 G=GETC:IF G=49 OR G=50 GOTO 100:GOTO 50 hetzelfde doet ?
Maar pas op, Koerts oplossing laat U vrij een regel tussen regel 50 en regel 100 in te voegen en de tweede methode niet direct, al is dat aan te passen. In beide mogelijkheden wordt echter door de AND respectievlijk OR extra tijd gebruikt. Hier misschien niet zo belangrijk maar het toetsenbord zal niet zo fijn reageren.
Vandaar mijn voorstel :
50 G=GETC:IF G<49 GOTO 50:IF G>50 GOTO 50
Hetzelfde recept kan ook worden toegepast op regel 530.
Koert wist vermoedelijk niet en daarin is hij zeker niet de enige dat bij een IF AND of een IF OR combinatie de DAI altijd beide mogelijkheden bekijkt en dan combineert alvorens actie te ondernemen. Beter zou zijn te stoppen als het toch al fout is. Dit is zo bij een OR als al aan de eerste voorwaarde wordt voldaan en bij een AND als de eerste al niet voldoet.
Structureel is de AND en OR te prefereren omdat anderen dan sneller kunnen begrijpen hoe en waarop getest wordt maar uit oogpunt van snelheid (bij DAI) niet.
Toch maak ik de ontwerpers van de DAI hier geen verwijt. De tijdwinst die soms geboekt kan worden gaat weer verloren door structuuronderzoek, terwijl nu de programmeur het kan beinvloeden.

Is de snelheid belangrijk kunnen we meerdere IFs gebruiken en de volgorde juist zo maken dat er zo min mogelijk getest hoeft te worden.

De score wordt bijgehouden op regel 320; achter S wordt nog een tweetal spaties afgedrukt dit is juist, daar anders soms de eindcijfers van een vorig maal kunnen blijven staan. Bedenkt U zelf eens waarom twee spaties nodig zijn en in welk geval een spatie niet vlodoende is. Achter deze " " zou ik echter nog graag een ';' willen zien om het zenuwachtige cursorgedrag te beperken. Of beter nog de cursor onzichtbaar maken door middel van POKE #75,32.

Over regel 180 valt het volgende op te merken X+X is sneller dan 2.0*X. Er is juist gebruik gemaakt van floating point getallen 1.0 en 2.0 omdat RND daarmee werkt en nu dus sneller is dan met integers. De INT is volledig overbodig en vertraagt alleen maar omdat er toch een toewijzing aan een integervariabele is.

En nu de fout. Koert voorzag terecht dat het mogelijk zou zijn dat een 'ster' op een reeds geplaatste ster terecht zou komen en dan zouden er maar 29 (of nog minder) in plaats van 30 sterren zijn en kunt U nooit 30 sterren 'opeten' om een nieuw veld te krijgen.
Koert loste dit schijnbaar goed op door te testen of de gekozen plaats leeg is. Is dit niet zo dan verlaagt hij de loopcounter O met 1. Dit is logisch gezien een uitstekende oplossing alleen werkt het op de DAI niet !! Aan het begin van elke FOR-NEXT loop wordt het aantal malen berekend dat de loop doorlopen moet worden. Het tussentijds veranderen van de loopvariabele O (een dummy) heeft dan ook geen invloed meer op het aantal malen dat de loop doorlopen wordt.
Ook heb ik structurele bezwaren tegen een FOR waar twee NEXT'n bijhoren.
Ook had de test zelf beter gekund, niet alleen de constructie met OR die hiervoor al besproken is maar ook het tweemaal laten berekenen van de SCRN(XX,YY) kost tijd.
En het is niet nodig: beter is HULP=SCRN(XX,YY):IF HULP=15 OR HULP=3 THEN.....
Nog beter IF SCRN(XX,YY)<>0 THEN ....maar dit is water naar de zee dragen, de fout wordt nu mooier en sneller gemaakt. Om de loopvariabele te verminderen is hier zinloos. Hoe lossen we het op ?
Er zijn meerdere methoden voorhanden; ik noem er twee :

I) we houden bij hoeveel sterren we plaatsen. Zijn het er bijvoorbeeld maar 28 dan behoeven we ook maar 28 sterren te pakken voor een nieuw veld.
In plaats van letterlijk te testen of SCRN(XX,YY)=0 is kunnen we de teller bijhouden door TELLER=TELLER+1-SGN(SCRN(XX,YY))

II) we kiezen een ander punt als het gekozen punt al bezet was en om tijd te winnen kiezen we alleen een nieuwe YY-waarde.

```
200   FOR O=1 TO 30:XX=RND(X)
210   YY=RND(Y):IF SCRN(XX,YY)<>0 GOTO 210
220   NEXT
```

Ook de methode van Koert om zowel de witte als de rode punten met dezelfde loop neer te zetten overtuigd mij van zijn programmeercapaciteiten. Zonder meer origineel maar naar mijn smaak hier de leesbaarheid van het programma zo beinvloedend, dat ik liever twee losse FOR-NEXT'n zie om de punten te plaatsen. Het is nauwelijks meer werk maar wel duidelijker voor anderen. Maar weet dat Koerts oplossing juist is en alleen mijn smaak zegt gekunsteld en niet kunstig.
Een ander argument tegen Koerts oplossing is dat nu bij de eerste doorgang van de loop altijd nodeloos op kleur 3 wordt getest die dan nog niet aanwezig is.
Maar genoeg over regel 180, het is weliswaar de interressantste regel van het programma maar zo vullen we tien DAInamics met alleen dit programma.

Bij regel 800 worden G,GG en T op 0 gezet, waarna naar het begin van het programma wordt gesprongen. Het was beter geweest deze initialisering dan daar aan het begin te doen. De nulmaking (wat een woord) kan dan ook op regel 510 en regel 900 achterwege blijven.
Na deze verandering verhoogt regel 900 alleen nog maar de score S met 5000 en gaat dan naar 110. We kunnen echter alleen vanuit regel 320 in regel 900 komen en dus is het beter daar (regel 302) de verhoging te doen en regel 900 te verwijderen.
Regel 800 kan na een kleine verandering ook weggelaten worden.

De ENVELOPE's veranderen nooit en kunnen dus beter bij de initialisatie staan.

De opbouw van het veld voor het overzicht over het programma in een apart blok aan het eind gezet. Dit blok laten beginnen met een aantal GETC's om keybounce tegen te gaan. In de oorspronkelijk versie was je wel eens al dood nog voor je iets gedaan had. Beter gezegd je had wel iets gedaan maar dat was bij de vorige beurt.
De GOTO in regel 200 beviel mij ook niet. De richtingsbepaling is essentieel en dus wil ik die voor in het programma. Het moet in feite het eerste zijn dat we zien afgezien van initialiseringszaken.

De methode die Koert gebruikt om de blokjes in beeld te krijgen die aangeven hoeveel ronden er nog zijn is goed en origineel. (zie de regels 120 en 130)
Toch vond ik het gekunsteld. 'Vreemde' methodes hebben dat nu eenmaal. Als het als volgt gedaan zou zijn opgelost :

```
40    IF B=1 THEN PRINT CHR$(255)+" "+CHR$(255)
50    IF B=2 THEN PRINT CHR$(255)+"   "
60    IF B=3 THEN PRINT "     "
```

zou, iedereen het zeer snel duidelijk zijn wat er gebeurt. Regel 60 kan zelfs na een CHR$(12) weggelaten worden. Hij heeft echter een eigen methode gezocht en dat is zeker te waarderen. Zelf ben ik toen ook een aparte oplossing gaan zoeken.

40 PRINT MID$(CHR$(255)+" "+CHR$(255)+" ",B+B,3) en de B dan van 0 tot 2 en niet van 1 tot 3. Ik wilde echter graag de CHR$(255) vermijden. Ik heb toen het volgende gedaan. De desbetreffende regel regelnummer 1 geven en de string die de MID$ gebruikt vervangen door "AAAAAAA". Toen naar utility gegaan en op #29F, #2A0 gekeken waar de textbuffer begint. Zonder CLEAR zal dit #3EC zijn. Het stuk na #3EC bekeken en de AAAAAAA's opgezocht. (code 41) Vervolgens deze 41's vervangen door FF 20 FF 20 20 20 20. Voor degenen die niet zo thuis zijn in ASCII-codes FF is volle blok en 20 is spatie. Terug naar BASIC en toen in EDIT het regelnummer weer aangepast. Moet U ook eens doen ! 'Vreemde' dingen als U LIST en / of EDIT doet met deze regel.

Regel 330 stond mij ook niet aan. Het is logisch om de GETC in een andere variabele te zetten om zo de oorspronkelijke richting te bewaren, maar waarom moest bij richtingverandering geen strafpunt gegeven worden ? De ON GG-15 GOTO .... dus vervangen door een GOTO 1100.

En ondanks dat het spel snel is toch ook principieel kijken naar de gevolgde logica. In regel 320 staat een test die onzinnig tenzij het laatste punt een wit punt was. De gehele regel hoort dus thuis achteraan in regel 310.
In regel 330 nu nog IF GG=0 veranderd in IF GG<16 omdat dan 'TAB' en 'RETURN' geen problemen meer kunnen geven.
Door verder het programma iets andere volgorde te geven was het mogelijk regel 330 en regel 1100 te combineren. Ik vermoed dat Koert er aan heeft zitten denken maar er niet uigekomen is.
Verder heb ik nog een paar schoonheidsfoutjes verbeterd. Het omzetten van floating point variabelen in integers (regel 50 en regel 10050) is daar een voorbeeld van.
Ik heb programma tot slot gerenumberd en een naamswijziging doen ondergaan.
Dit laatste vind ik zelf een grote ingreep daarom nu de argumenten die mij er toe brachten het toch te doen. Games collection nummer 12 is een jachtcollectie geworden en nu leek mij een naam als STAR-HUNTER beter. Ten eerste omdat dat mijns inziens ook beter de bedoeling van het spel aangeeft en ten tweede omdat die naam duidelijk thuishoort in een collectie als nummer 12.
De gewijzigde versie laat ik nu expres niet zien. U heeft de oorspronkelijke versie en met het doornemen van dit artikel kunt U de 'nieuwe versie verkrijgen.
Ook het aanschaffen van Games Collection 12 verschaft U de nieuwe versie.

Nogmaals Koert hartelijk dank zonder jou was dit artikel niet ontstaan.

Frank Druijff

ERRATUM DAInamic Debugging Tool

```
10    REM ERRATUM DAInamic Debugging Tool  (TK 5)
20    REM Please adapt lines 1350/1360/1370 as follows :
30    REM ------------------------------------------------
1350  GOSUB 51000:REM --- REGISTERS TO VARIABLES---
1360  IF PC=POK(8) OR PC=POK(8)+1 THEN PC=PCNEXT
1370  IF RSTCALLFLAG=1 AND SPOLD-SP=2 THEN POKE MLPSTACKPTR+12,PCNEXT
MOD 256:POKE MLPSTACKPTR+13,PCNEXT SHR 8:GOSUB 51000
```

LIGNES D'EXTENSION

Le moniteur du DAI permet l'utilisation de lignes d'extensions. Ceci est possible jusqu'a 3 lignes supplementaires.
Chaque ligne supplementaire commence par un C et contient 6 espaces à son début.

Mr Markus SIGG à concu un proramme en langage machine (mlp) pour manipuler ces lignes
Il permet l'utilisation de plus de 3 lignes supplementaires, efface le C et permet de mettre plus ou moins de blancs en debut de ligne.

```
INIT    PUSH H
        LXI H    START          ;Ad. du programme
        SHLD     6CH            ;charge le vecteur de RST 5
        POP H
        RET
;
START   PUSH H
        PUSH PSW                ;A=dernier car. entre
        CPI      0DH            ;est-ce un CR ?
        JZ       READY          ;si oui -Fin
        LDA      72H            ;Pos.Curseur (Poids faible)
        LHLD     7AH            ;Pos. dans ligne
        CMP L                   ;Fin de ligne atteind ?
        JNZ      READY          ;Si non -Fin
        CALL     0DD5EH         ;Imprime un CR
SPACE   MVI A    20H            ;espace en ASCII
        MVI H    6H             ;nombres d'espaces
OUTC    RST 5                   ;sortie sur ecran
        DB       3H
        DCR H
        JNZ      OUTC           ;espace suivant
READY   POP PSW
        POP H
        JMP      0C6FDH         ;RST 5 normal

        END
```

La routine INIT doit etre utilisèe une fois au dèbut du programme. Elle remplace l'adresse du vecteur d'interruption RST 5 en #006C/D par l'adresse de ce programme. (Ceci peut etre fait sous utilitaire en faisant UT , V5=START)

Le reste du programme sera utilisè par le moniteur à chaque interruption RST 5 (Gestion de l'ecran).
Si vous ne voulez pas d'espace en debut de ligne, sautez la partie SPACE.
On peut modifier le nombre d'espaces en changeant le contenu de H dans la routine SPACE.
A la fin du programme on retourne à l'adresse normale de RST5:#C6FD
Ce programme n'a pas d'influence sur la sortie RS 232.

DAI RESTART ROUTINES

Le basic du DAI et le système d'exploitation interne sont conçus de telle façon qu'il est difficile de les modifier. L'utilisateur ne peut définir que les Input/Output (Cassette, Clavier, RS 232C, bus DCE, Gestion de la mémoire).

Vous pouvez donc écrire vos propres routines (comme le TOS (Tape Operating System) du MEMOCOM MDCR qui vérifie les entrées au clavier. Dès que (RETURN) est pressée, il regarde s'il y a des commandes DCR.)

Cette méthode fonctionne tant que le basic ne reprend pas la main et dans un programme, vous devez utiliser l'instruction CALLM pour travailler avec le MDCR. Mais il y a d'autres moyens d'étendre le basic et le système d'exploitation interne. Un de ceux-ci par exemple, est la méthode que le DAI utilise pour "switcher" les banques de ROM. Dans les ROM du DAI, les adresses E000-EFFF existent 4 fois (Dans les banques 0, 1, 2, 3). Le numéro de la banque sélectionnée est déterminé par les deux bits les plus significatifs des adresses #FD06 et #40. En basic, la routine ressemblerait à ceci:
POKE #FD06,(PEEK(#FD06) IAND #3F) IOR (NUMEROBANQUE SHL 6)
La meme chose pour l'adresse #40. N'essayez pas ceci car le basic utilise la banque 0!

De façon interne, le DAI utilise la meme méthode pour "switcher" les banques chaque fois qu'un appel est effectué aux banques 1, 2 ou 3. Ceci ne peut se faire d'une façon directe par l'instruction CALL car un CALL ne switche pas les banques. Il existe une instruction spéciale dans le 8080:
RST X (X=0-7) qui est suivie d'un byte-donnée. RST X est une instruction occupant un byte et similaire à CALL.

On utilise les RST X suivants Pour connecter les banques:

RST 4	switche la banque 1	Math & routines Son
RST 5	switche la banque 2	Gestion d'écran/EDIT
RST 1	switche la banque 3	Encodage/UTILITAIRE

TOUS LES NOMBRES SONT EN NOTATION HEXADECIMALE!
Par exemple, la routine pour imprimer un caractère sur l'écran est RST 5, DATA 3 (EF 03). C'est un appel à l'adresse #E003 de la banque 2.
Le registre A doit contenir le code ASCII du caractère.
RST 5 est un appel à l'adresse #28 (= 8*5 = 40D).
A #28, les instructions suivantes préparent l'accès à la banque:

28	NOP	00
29	PUSH H	E1
2A	LHLD 006C	2A 6C 00
2D	PCHL	59
2E	NOP	00
2F	NOP	00

A l'adresse #6C/#6D se trouve FD C6, un saut sera donc effectué à #C6FD avec le contenu initial de HL placé au dessus de la pile. Comme tout ceci se trouve en RAM, vous pouvez modifier le contenu des adresses #6C/#6D ou #2B. Ces adresses peuvent etre modifiées par la commande V (après avoir tapé UT). Ex.:
V5 C6FD-300 (flèche gauche). Ceci force le RST 5 à continuer à l'adresse #300 ou à l'endroit où se trouve votre programme. Un bon exemple est le programme de conversion APPLE/ATARI qui fut publié dans une précédente newsletter.

En utilisant cette méthode, vous pouvez manipuler toutes les fonctions des banques 1 à 3 et meme de la banque 0.
RST 5, DATA 3 par exemple est très utile. Cette routine est utilisée pour tout ce qui doit etre affiché sur l'écran.

Cette routine est utilisée par PRINT, LIST et tous les messages du DAI. Ainsi,

nous pouvons changer toutes ces routines en modifiant:

1) les contenus des registres A-L du 8080.
2) le pointeur de pile.
3) l'adresse de retour.
4) le byte-donnée après RST X.

Pour ceci nous devons connaitre les contenus des registres, l'objet du RST X (la valeur du byte-donnée) ou la valeur du pointeur de pile au moment du RST X. (Ceux qui ne connaissent pas le firmware du DAI peuvent se référer aux routines publiées dans des précédentes newsletters ou au "Firmware Manual" de J. Boerrigter.) Le principal est de vérifier le byte-donnée parce qu'il détermine quelle routine est appellée, et le contenu des registres.

Comme exemple, nous prendrons RST 5, DATA 3 et supposons que notre routine commence à #300. Comment trouver le byte-donnée? Lorsque le RST X est exécuté, l'adresse de retour est placée dans la pile et cette adresse est celle du byte-donnée car elle pointe directement après l'instruction RST X. Avant que le saut ne soit fait à votre propre routine, le registre double HL sera placé dans la pile (V. l'exemple sur la première page).
Dans notre exemple, le registre A est vérifié après que nous ayons trouvé que le byte-donnée est 3. Si le caractère est une majuscule, il sera converti en minuscule.
L'interpréteur basic ne reconnait pas les commandes écrites en minuscules. Pour cela, nous devons changer de nouveau les commandes écrites en minuscules, en majuscules. Ceci peut etre fait. L'interpréteur utilise une routine pour lire des caractères sur l'écran (RAM). Cette routine est RST 5, DATA 15.
Nous devons changer l'adresse de retour après cette routine et placer l'adresse de notre propre routine et ensuite, vérifier si le caractère reçu est une minuscule (et dans ce cas, le convertir en majuscule). De cette façon, l'interpréteur accepte les commandes en minuscules. La routine complète sera la suivante:

```
            COMMANDES EN MINUSCULES
300         POP H       ;retire HL de la pile
301         XTHL        ;échange HL et le sommet de la pile
                        ;l'adresse de retour est dans HL
302         PUSH PSW    ;sauve PSW
303         MOV A,M     ;A reçoit le contenu de l'adresse de retour
304         CPI 3H      ;Est-ce le byte-donnée pour imprimer
                        ;un caractère à l'écran?
306         JZ MAJmin
309         CPI 15H     ;Est-ce le byte-donnée pour lire
                        ;un caractère à l'écran?
30B         JZ minMAJ
30E out0    POP PSW     ;retire HL de la pile
30F out     XTHL        ;RST 5 si ce n'est pas la bonne routine
310         PUSH H
311         JMP C6FDH   ;adresse normale du RST 5

320 MAJmin  LDA 118H
323         CPI FFH
325         JNZ out0
328         POP PSW     ;A reçoit de nouveau la valeur ASCII
329         CPI 'A'     ;si valeur inf. à 'A', pas d'action
32B         JC out
32E         CPI 91      ;si valeur sup. à 'Z', pas d'action
330         JNC out
```

```
333          ORI 20H      ;converti en minuscule
335          JMP out

338 minMAJ   POP PSW      ;retire le PSW
339          INX H        ;l'adresse de retour est celle du byte-donnée
33A          XTHL         ;HL=HL initial
                          ;sommet de la pile=ancienne adresse de retour+1
33B          PUSH H       ;sauve HL
33C          LXI H,ret    ;adresse de retour à notre routine
33F          XTHL         ;HL=HL initial
                          ;sommet de pile=nouvelle adresse de retour
340          PUSH H       ;sommet de pile=HL initial
341          JMP C6FDH    ;continue RST 5 et retour à notre routine
344 ret      DB 15H       ;byte-donnée après l'adresse de retour
345          PUSH PSW     ;sauve les status
346          CPI 'a'      ;si inf. à 'a', pas d'action
348          JC out2
34B          CPI 123      ;si sup. à 'z', pas d'action
34D          JNC out2
350          SUI 20H      ;converti en majuscule
352 out2     XTHL         ;status dans L
353          MOV H,A      ;caractère dans H
354          XTHL
355          POP PSW
356          RET          ;retour à l'ancienne adresse de retour
             END          ;après le byte-donnée
```

Dans cette routine la conversion n'est pas opérationelle pendant l'entrée de commandes basic. Vous pouvez donc mixer majuscules et minuscules. Tous les listings meme en UTILITY ou avec l'assembleur DNA seront en minuscules.

Parfois il n'est pas suffisant de connaitre le byte-donnée ou l'adresse de retour si nous voulons changer une routine parce que la routine qui utilise d'autres routines différentes avant une instruction RST X est utilisée. Comme exemple, prenons les routines d'erreurs du DAI. Si nous voulons créer un ON ERROR GOTO routine, il y a un moment où nous pouvons appeller cette routine avant que l'exécution ne soit stoppée. Ce moment est juste avant que le message ne s'imprime. Ce message est affiché par (encore!) RST 5, DATA 3. Pour savoir si un message d'erreur est imprimé, nous devons regarder la pile et le pointeur de pile. Ceci parce que chaque fois qu'une routine différente est appellée, elle laisse une adresse de retour au sommet de la pile puis le pointeur de pile est décrémenté de deux.
Cela veut dire que nous pouvons retrouver l'adresse de retour de l'erreur quelque part dans la pile. Comme nous pouvons examiner le sommet de la pile au moyen de l'instruction DAD SP et que nous connaissons combien de niveau de sous-routines se trouvant dans la pile, nous savons toujours si l'appel initial est une routine d'erreur. Avec ces connaissances, nous allons traiter la pile comme une mémoire normale. Nous ajoutons au pointeur de pile deux fois le nombre de niveaux de sous-routines et nous obtenons l'adresse de retour au moyen de l'instruction MOV A,M. Si nous trouvons une erreur, le programme basic continue, avant qu'un message d'erreur ne s'imprime, à une ligne de programme spécifiée par l'utilisateur. Si cette ligne n'existe pas, comme les routines d'erreurs sont à ce moment rendues opérationelles, un message "UNDEFINED LINENUMBER" sera imprimé.

En utilisant la méthode décrite ci-dessus, il est possible d'étendre le basic du DAI et le système d'exploitation. Quelques fois très facilement, quelques fois en recopiant un peu et en modifiant les routines déjà existantes en ROM. J'ai écris un programme d'environ 500 bytes utilisant cette méthode et qui rend possible l'extension du basic à 65 nouvelles commandes entièrement compilées au format standard du basic du DAI. Ceci pour illustrer la puissance de cette méthode. J'espère que cet article vous donnera une idée de la façon d'interagir avec le basic du DAI et de faire plus avec votre DAI.

N.P. Looije

STAR DP-510 STAR DP-515

NEW PRINTERS

CHARACTERISTICS

- Printing speed: 100 cps (Normal character)
- Super Subscript printing
- Bit Image
- Mixture printing of varions size characters in one line

SPECIFICATIONS

Printing system	Serial impact dot matrix system
Interface	Parallel interface(Centronics compatible)-standard- Serial interface(RS-232C, Current loop) -option-
Printing composition	Standard characters: 9 x 9 dot matrix Block graphic: 6 x 6 dot matrix Bit image: (7 or 8) x 480 dot matrix (single density) (7 or 8) x 960 dot matrix (double density)
Direction of printing	Standard characters: Bi-directional printing (with logical seeking) Bit image: Uni-directional printing
Nuber of print characters	80, 96, 132 characters (in case of expanded characters: 40, 48, 66 characters)
Printing speed	100 characters/sec
Character types	96: Standard ASCII character type 96: Italic ASCII character type 64: Special character type 32: Block graphic character type 96: Proportional ASCII character type(*) 32: Proportional special character type (*) (*) = option
Character size	2.4 (H) x 2.0 (W) mm (in the case of 80 characters/line printing)
Character pitch	10, 12, 17 characters/inch (in the case of expanded characters; 5, 6, 8.5 characters/inch)
Line pitch	1/6, 1/8, n/72, n/144 inch line feed

Paper feed system	Sprocket feed and friction feed
Paper feed speed	10 lines/sec (in case of 1/6 inch line feed)
Dimensions	392 (W) x 315 (D) x 136 (H) mm (15.2 x 12.4 x 5.3 inches)
Weight	Approx. 7kg (15.4 pounds)
Power requirement	100V, 120V, 220V, 240V· AC ±10%, 50/60Hz, approx. 120W (power consumption)

■ FORM SPECIFICATIONS

Single sheet	Width: Thickness:	8·10 inches 0.07 ·· 0.1mm
Roll paper	Width: Thickness: Diameter of roll:	8.5 inches 0.07 ·· 0.1mm max. 5 inches
Sprocket. paper	Copy: Width: Thickness:	3 sheets (original +2 sheets) no carbon paper. 3 ·· 10 inches 0.07 ·· 0.1mm (single paper) 0.28mm (3 part form)

■ INK RIBBON SPECIFICATIONS

Color	Black sigle color
Material of ribbon	Nylon # 40
Dimensions of ribbon	13 (W) x 10.5 (L) mm (1/2 inch x 11.5 yards) Eyelets are provided near both ends.
Spool standard	Underwood's plastic type. 13mm x 50mm dia.
Recommended product	Type SF-02B made by Fuji Kagakushi Kogyo, Co.

CONTROL CODE

CR	CR code causes print out of buffer.
LF	One line paper feed is done by an input of LF code after buffer data is printed.
VT	Feed paper to the next VT position where programmed by "ESC Pnnnn NUL" after buffer data is printed.
FF	Paper is fed to the print start line in next page by preset program (Header line) after buffer data is printed.
HT	Moves the print head to the next TAB set position.
SO	Turns on Double width character print mode.
SI	Turns on Compressed mode.
DC2	Cancell SI-mode.
DC4	Cancell SO-mode.
BEL	Buzzer sounds for 1/4 sec. when this code is input.
DC1	This code causes printer to be selected.
DC3	This code causes printer to be deselected.
ESC 0	This code changes line feed length to 1/8 inches.
ESC 1	This code changes line feed length to 7/72 inches.
ESC 2	This code changes line feed length to 1/6 inches.
ESC 3n	This code changes line feed length to n/144 inches.
ESC An	This code cahnges line feed length to n/72 inches.
ESC Jn	One time line feed of n/144 inches.
ESC B	ESC B1 Sets 10 CPI. 80 columns. ESC B2 Sets 12 CPI. 96 columns. ESC B3 Sets 17 CPI. 132 columns.

ESC W	Double width character print mode is selected.
ESC Cn	Sets form length to n lines.
ESC C 0 n	Sets form length to n inches.
ESC Mn	Sets LH-Margin at n columns from left hand side in printing area.
ESC Qn	Sets RH-Margin at n columns from left hand side in printing area.
ESC D nn-n NUL	Sets new HT positions (Horizontal Tab).
ESC P nn-n NUL	Sets new VT positions (Vartical Tab).
ESC -	All data after this code is printed with under line.
ESC E	Selects EMPHASIZE print mode.
ESC F	Cancells EMPHASIZE print mode.
ESC G	Selects DOUBLE STRIKE print mode.
ESC H	Cancells DOUBLE STRIKE print mode.
ESC S	Super/Subscript mode is selected.
ESC T	Resets "ESC S" and "ESC U"
ESC U	Selects uni-direction print mode.
ESC Rn	Sets Header line position at n line.
ESC Nn	Sets skip over perforation to n lines.
ESC O	Resets skip over perforation to zero lines.
ESC 4	This code selects character ROM set of ITALICS.
ESC 5	This code resets standard character ROM set of ITALICS.

Personal Printer RX-80

SPECIFICATIONS

Print Method	Serial impact dot matrix
Print Rate	100 characters per second
Print Direction	Bidirectional with logic-seeking in the text mode Unidirectional in the bit image and graphic mode
Number of Pins in Head	9
Line Spacing	1/6, 1/8, 7/72 inch or programmable

PRINTING CHARACTERISTICS

Character Set	Full 96-characters ASCII with descenders plus 11 international character sets
Character Size	2.1(W)×3.1(H)mm (Normal)
Character Structure	Text mode: 9×9 Bit image: 480×8, 640×8, 720×8, 960×8, 1920×8 dots/line

PRINTING SIZE

	Max. Characters/Line
Normal	80
Enlarged	40
Condensed	137
Condensed-Enlarged	68
Elite	96
Elite-Enlarged	48

FORMS HANDLING

Form Feed	Programmable length to 127 lines

MEDIA HANDLING

Paper Feed	Sprocket pin feed
Paper Width Range	Fanfold; 4″ to 10″
Copies	One original plus two carbon copies (total thickness not to exceed 0.3mm)

INTERFACE

Standard	Centronics-style 8 bit parallel
Optional	IEEE 488, RS-232C, etc.

INKED RIBBON

Color	Black
Type	Exclusive cartridge
Life Expectancy	3 million characters

ENVIRONMENTAL CONDITIONS

Operating Temperature	5 to 35° centigrade
Operating Humidity	10 to 80% non-condensing

POWER REQUIREMENT

Voltage	AC 120V, 220/240V ±10%, 49.5 to 60.5Hz
Power Consumption	70 VA max.

PHYSICAL CHARACTERISTICS

Height	107mm
Width	389mm
Depth	303mm
Weight	5.1kg

MAJOR SOFTWARE CONTROL CODES

Control Code	Function
CR	starts printing
LF	advances paper one line
FF	advances paper to top of next form
VT	vertical tabulation
HT	horizontal tabulation
SO	sets enlarged printing mode
SI	sets condensed printing mode
BEL	sounds the buzzer
DC2	cancels condensed printing mode
DC4	cancels enlarged printing mode set by SO code
DEL	cancels last printable code
BS	prints and backs by one character
ESC *	selects 8 dot bit image mode
ESC 0	sets 1/8″ line spacing
ESC 2	sets 1/6″ line spacing
ESC 3+n	sets n/216″ line spacing
ESC 4	selects alternate character set
ESC 5	cancels alternate character set
ESC 8	ignores paper end signal
ESC 9	enables paper end signal
ESC -	sets/cancels underline printing mode
ESC <	prints from leftmost to right for one line
ESC @	initializes the printer
ESC A	sets amount of one line paper feed
ESC C	sets form length
ESC E	sets emphasized printing mode
ESC F	cancels emphasized printing mode
ESC G	sets double-strike printing mode
ESC H	cancels double-strike printing mode
ESC I	selects control code or printable character
ESC K	sets normal-density bit image mode
ESC L	sets dual-density bit image mode
ESC M	sets elite-sized character
ESC N	sets skip-over perforation
ESC O	cancels skip-over perforation
ESC P	sets pica-sized character
ESC Q	sets column end
ESC R	selects international character set
ESC S	sets super/subscript printing mode
ESC T	cancels super/subscript printing mode
ESC U	sets/cancels unidirectional printing
ESC W	sets/cancels enlarged printing mode
ESC Y	sets double speed dual-density bit image mode
ESC Z	sets quadruple-density bit image mode
ESC e	relative horizontal/vertical tabulation
ESC l	sets column head
ESC s	sets/cancels half speed printing

Specifications are subject to change without notice.

Versatile Printer FX-80

SPECIFICATIONS

Print Method ..... Serial impact dot matrix
Print Rate ..... 160 characters per second
Print Direction ..... Bidirectional with logic-seeking in the text mode
Unidirectional in the graphic mode
Number of Pins in Head ... 9
Line Spacing ..... 1/6, 1/8 inch selectable by DIP switch or programmable from 1/216 to 255/216 inch

PRINTING CHARACTERISTICS

Character Set ..... Full 96-character ASCII with descenders plus 9 international character sets
Character Size ..... 3.1(H) × 2.1(W)mm (Normal)
Character Structure ..... Text mode; 9 × 9
Bit image; 480 × 8, 960 × 8, 576 × 8, 640 × 8, 720 × 8, 1920 × 8, 480 × 9, 960 × 8 dots/line

PRINTING SIZE

	Max. Characters/Line
Normal	80
Enlarged	40
Condensed	137
Condensed-Enlarged	68
Elite	96
Elite-Enlarged	48

FORMS HANDLING

Form Feed ..... Programmable length to 255 lines
Horizontal Tab ..... To 32 positions
Vertical Tab ..... To 16 positions at 8 channels each with Electronical Vertical Feed Unit

MEDIA HANDLING

Paper Feed ..... Adjustable sprocket pin feed plus friction feed
Paper Width Range ..... Fanfold; 9.5″ to 10″ (with an optional tractor unit; 4″ to 9″)
Roll Paper; 8.5″ ± 0.12″ (with an optional roll paper holder)
Cut Sheet; 8.3″ to 8.5″
Copies ..... One original plus two carbon copies (total thickness not to exceed 0.3mm)

INTERFACE

Standard ..... Centronics-style 8 bit Parallel
Optional ..... IEEE 488, RS-232C, etc.

INKED RIBBON

Color ..... Black
Type ..... Exclusive cartridge
Life Expectancy ..... 3 million characters

Environmental Conditions

Operating Temperature ... 5 to 35° centigrade
Operating Humidity ..... 10 to 80% non-condensing

POWER REQUIREMENT

Voltage 120, 220, 240V, 50/60Hz
Current ..... 1 Amp max.
Power Consumption ..... 70 VA max.

PHYSICAL CHARACTERISTICS

Height 100mm
Width 420mm
Depth 347mm
Weight 7.5kg

MAJOR SOFTWARE CONTROL CODES

Control Code	Function
CR	starts printing
LF	advances paper one line
FF	advances paper to top of next form
VT	vertical tabulation
HT	horizontal tabulation
SO	sets enlarged printing mode
SI	sets condensed printing mode
BEL	sounds the buzzer
DC2	cancels condensed printing mode
DC4	cancels enlarged printing mode set by SO code
DEL	cancels last printable code
BS	prints and backs by one character
ESC !	selects print mode
ESC %1	selects download character set
ESC :	copies fonts from ROM CG to download CG
ESC &	defines downloadable character font
ESC *	selects 8 dot bit image mode
ESC /	selects vertical format unit channel
ESC 0	sets 1/8″ line spacing
ESC 2	sets 1/6″ line spacing
ESC 3	sets n/216″ line spacing
ESC 4	selects alternate character set
ESC 5	cancels alternate character set
ESC 8	ignores paper end signal
ESC 9	enables paper end signal
ESC –	sets/cancels underline printing mode
ESC <	prints from leftmost to right for one line
ESC >	sets MSB to logical 1
ESC =	sets MSB to logical 0
ESC @	initializes the printer
ESC A	sets amount of one line paper feed
ESC B	sets vertical TAB stop position
ESC C	sets form length
ESC D	sets horizontal TAB stop position
ESC E	sets emphasized printing mode
ESC F	cancels emphasized printing mode
ESC G	sets double-strike printing mode
ESC H	cancels double-strike printing mode
ESC I	selects control code or printable character
ESC K	sets normal-density bit image mode
ESC L	sets dual-density bit image mode
ESC M	sets elite-sized character
ESC N	sets skip-over perforation
ESC O	cancels skip-over perforation
ESC P	sets pica-sized character
ESC Q	sets column end
ESC R	selects international character set
ESC S	sets super/subscript printing mode
ESC T	cancels super/subscript printing mode
ESC U	sets/cancels unidirectional printing
ESC W	sets/cancels enlarged printing mode
ESC Y	sets double speed dual-density bit image mode
ESC Z	sets quadruple-density bit image mode
ESC b	sets VFU stop position
ESC j	reverses paper feed
ESC l	sets column head
ESC m	selects special CG
ESC p	sets/cancels proportional spacing mode
ESC s	sets/cancels half speed printing

Specifications are subject to change without notice.

IMPRESSIONS of the CENTRONICS 739

Looking through the back issues of DAInamic and from contacts with other DAI users it would seem that most have chosen the EPSON MX.. range printers, and having made a different choice myself I thought a review of my printer might be of interest. A year or so ago when I started investigating printers the advertised price of the Centronics 739 was more or less the same as the MX 80, and as my main requirement (after program listing) was for a document printer, I decided to go for the Centronics.

The printer can produce two entirely different character sets, one used for the letter quality proportional space characters and the other for standard print. The standard set (10 CPI Normal) produces 80 characters/line, and the same set can be compacted for 132 characters/line (16.7 CPI Condensed). The printer has a nine wire/pin head which produces a 7x8 matrix for Normal and Condensed print, whilst proportional characters are formed by up to 9 vertical and 6 - 18 horizontal dots. In graphics mode only the top 6 pins are used, with 594 dot positions per line of 8 inches. Each character type can be elongated, and there are optional foreign characters selectable by an internal DIL switch. In addition, two other DIL switches select auto line feed on/off and a self test function. There are three external controls at the front of the printer, on/off, online/local, and rev/forward paper feed. Strangely, the on/off indicator LED was mounted inside the case on the main circuit board making it invisible except when looking straight down on the top. I have since moved it to the obvious position next to the switch. The printer can handle three paper types, 11 x 9.5 inch perforated fan-fold, 8.5 inch continuous roll, and single sheets, and runs at 100 cps in the standard and 80 cps in proportional mode. The printer can produce super and subscript by shifting the character base line half a line up or down, and an underline can be switched on or off anywhere in a line. Within any line the print head can be backspaced or moved on any number of dot positions under program control, and this can be used to advantage for right hand margin justification, as the print on this page shows.

Since having the printer I have suffered two breakdowns, the first time a miniature fuse being the only casualty, however in the second case I had to replace four diodes, which was less easy to trace. Nevertheless, I am well satisfied with the printers overall robustness and the high quality print output. Here are some examples . . .

PROPORTIONAL !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz{|}~

10 CPI NORMAL !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRS
TUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

16.7 CPI CONDENSED !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

PROPORTIONAL ELONGATED !"#$%&'()*+,-./012
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWX
YZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

10 CPI ELONGATED !"#$%&'()*+,-./012
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTU
VWXYZ[\]^_`abcdefghijklmnopqrstuvwx
yz{|}~

16.7 CPI ELONGATED !"#$%&'()*+,-./0123456789:;<=>?@ABCD
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

THE BASIC MONITOR (#C80C-#C956)

PART 1: GENERAL

This article describes the principles of the BASIC monitor routine of the DAI in general. In further articles, the details of certain parts of the monitor will be explained. As reference for this article, the 'DAI pC FIRMWARE MANUAL' is used. The annexed (simplified) flow chart will be the guide through this article.

The BASIC monitor might be seen as the 'heart' of the operating system of the DAI-BASIC. Under control of this monitor, program lines can be entered into the textbuffer, direct commands will be executed, the run of a program is controlled, a 'break' is managed, etc.

The monitor is entered by the DAI after a reset or power-on at address #C818. Other entries (on #C80C, #C814 and #C823) are also used after finishing certain routines. This will be explained in part 2.

INITIALISATION (#C818-#C843):

The monitor must be initialised. That means, a defined start situation has to be created. The stackpointer is set, all flags indicating the running of inputs, programs, subroutine calls and of encoding stored lines are cleared. Result: the DAI is 'at rest'. Keyboard interrupts and clock interrupts are enabled, otherwise no inputs can be made and no output to the screen would be possible.

INPUT FROM THE EDIT BUFFER (#C846-#C84E):

The encoded input switch #0135 is checked. This switch defines the source for inputs which have to be encoded. Encoding means: Translation of BASIC lines (direct commands or program lines) from the readable BASIC into the code which is understandable by the DAI semi-compiler in order to excute these lines. In this code a program is also stored in the textbuffer. Details about this pseudo-code can be found in Newsletter 11, page 196 and further.

If the input source is the editbuffer, then handling of the inputs from this buffer is prepared via #D879, and encoding takes place via #C867. When one textline is encoded, the monitor starts again at the beginning, until the editbuffer is empty.
See also a previous Newsletter for the 'editor story'.

Remark: Encoding of 'undefined rubbish' (namely the momentary contents of the encoded input buffer) occurs also if the switch #0135 is >2 !! This firmware failure is based on the assumption that #0135 is always <=2.

INPUT FROM KEYBOARD (#C851-#C864):

If the input source is not the editbuffer, but the keyboard, the prompt ('*') is displayed on the screen and the input of a BASIC textline is awaited via #DD1A. In plain text: the DAI waits for a direct command line or a program line to be stored in the textbuffer, which must be entered via the keyboard. This input sequence is only finished by typing a CR ('RETURN') or by a break. All characters typed in are displayed on the screen. The screen memory is at the same time the onliest place where the inputs are stored.
When CR is typed, the monitor starts working again. It takes the first character from the input line in the screen memory for investigation.

The difference between direct command lines and textlines to be stored in the textbuffer is the existance of a linenumber at the beginning of program lines to be stored.
If the first character of an input line is a number, then the line must be stored in the textbuffer after encoding. Otherwise it is a direct command line, and execution of this line must take place directly after encoding.

If the first character in the input line is a CR, then no input line is available, and the monitor waits for a new input line.

ENCODING PROGRAM LINES (#C867-#C86A):

If a program line has been typed in, this line is encoded and stored in the textbuffer on the correct location; that means in sequence of linenumber. The monitor is restarted to get new input lines.
The details on how encoding is processed will be explained in a next article.

ENCODING OF DIRECT COMMAND LINES (#C86D-#C87D):

The direct command lines are encoded via the routine for encoding a BASIC command via RST1/00. The result of this process - the semi-compiled code - is stored in the encoded input buffer #013E-#01BD. This buffer is 128 bytes large, so the maximum length of an encoded program line is therefore 128 bytes, otherwise an error message 'LINE TOO COMPLEX' will be the result.
The flag #0117 is set (#FF), simulating a 'RUN' of a program line; the registers BC - used to indicated the position in a program line where the monitor is busy executing it - point to the beginning of this encoded direct command line in the encoded input buffer EBUF, and the direct command line will be executed.

RUNNING OF DIRECT COMMANDS OR PROGRAM LINES (#C87F-#C88F):

The next part is valid both for the execution of direct command lines as for the execution of program lines stored in the textbuffer.
The onliest difference is the contents of the register pair BC. For direct command execution, it points to the encoded input buffer EBUF. For the execution of a program stored in the textbuffer, it points to the start of a textline in this buffer.

The first character of a direct command or a stored line is taken. If it is a token - an abbreviation for a BASIC command - the startaddress of the execution routine of the BASIC command belonging to this token is found in the table #CF02, and this routine is executed via #C8A9.

If the first byte is not a token (#C8E5), then it could be a '0' or the length byte at the beginning of a stored program line. A '0' indicates the end of the textbuffer contents (where a '0' is inserted after the last statement), or the end of a direct command line (see #C876).
When encountering a '0', the monitor is restarted.

If the first byte read by the monitor is the length byte of a stored program line, then the trace/step flags are tested (#0115/#0116). If one of these flags is set, the new program line will be listed on the screen. In case of 'STEP', pressing the spacebar is awaited.
If no 'break' is pressed, the monitor continues on #C87F.

SPECIAL END OF ACTIONS (#C8AA-#C8B5):

If a token has been found, and the particular execution routine has been performed, the monitor returns to #C88F. Here is checked if the execution routine for the particular BASIC routine has eventually a special ending.
This special end is indicated by a code in accumulator A and the carry-flag set after running the BASIC execution routine. See for example #DF03: 'STOP': CY=1, A=3.

Four special actions are possible:

0: #C908-#C915: After 'LOAD': If 'LOAD' is not a direct command, the loaded program will be runned immediately.

1: #C818: Can't continu: After commands which do change program pointers or which alter the contents of the textbuffer, the monitor is restarted.

2: #C8C0-#C8C8: After a 'soft break': The 'break' must be handled.

3: #C8B8-#C8BD: After a 'STOP' command in a program line. The place where the program execution is stopped must be remembered and 'STOPPED IN LINE ....' has to be printed. Further handling will be as a 'soft break'.

'SOFT BREAK' HANDLING (#C8CB-#C8E2):

If a 'break' has been given during the run of a program, 'BREAK IN LINE ....' will be printed. All pointers which are relevant to the program status - the so-called FRAME #0100-#0115 - are saved on stack, and flag #0126, indicating the existance of a suspended (= interupted) program is set before the monitor is restarted.
If a 'break' is given during excution of a direct command, then only a CR is printed. No program status have to be saved now.

SUMMARY OF THE MAIN LOOPS IN THE MONITOR:

Once the monitor is initialised, a program is stored in the textbuffer, and the direct command 'RUN' is given, the following happens:

- 'RUN' is a direct command, which is encoded into the encoded input buffer EBUF. Via its token #87, the execution routine on #DF9E is found. There - amongst others - the register pair BC is set to the startaddress of the textbuffer, and the monitor is re-entered on point (E).

- Via main loop 1, the first character of the textbuffer is taken. This is the length byte of the first program line. If no TRACE or STEP flags are set, the monitor returns at point (B) with incremented linepointer BC.

- Via main loop 2 the monitor continues. If a token is found, after the linenumber, the appropriate BASIC excution routine is executed, and the monitor continues at point (B).

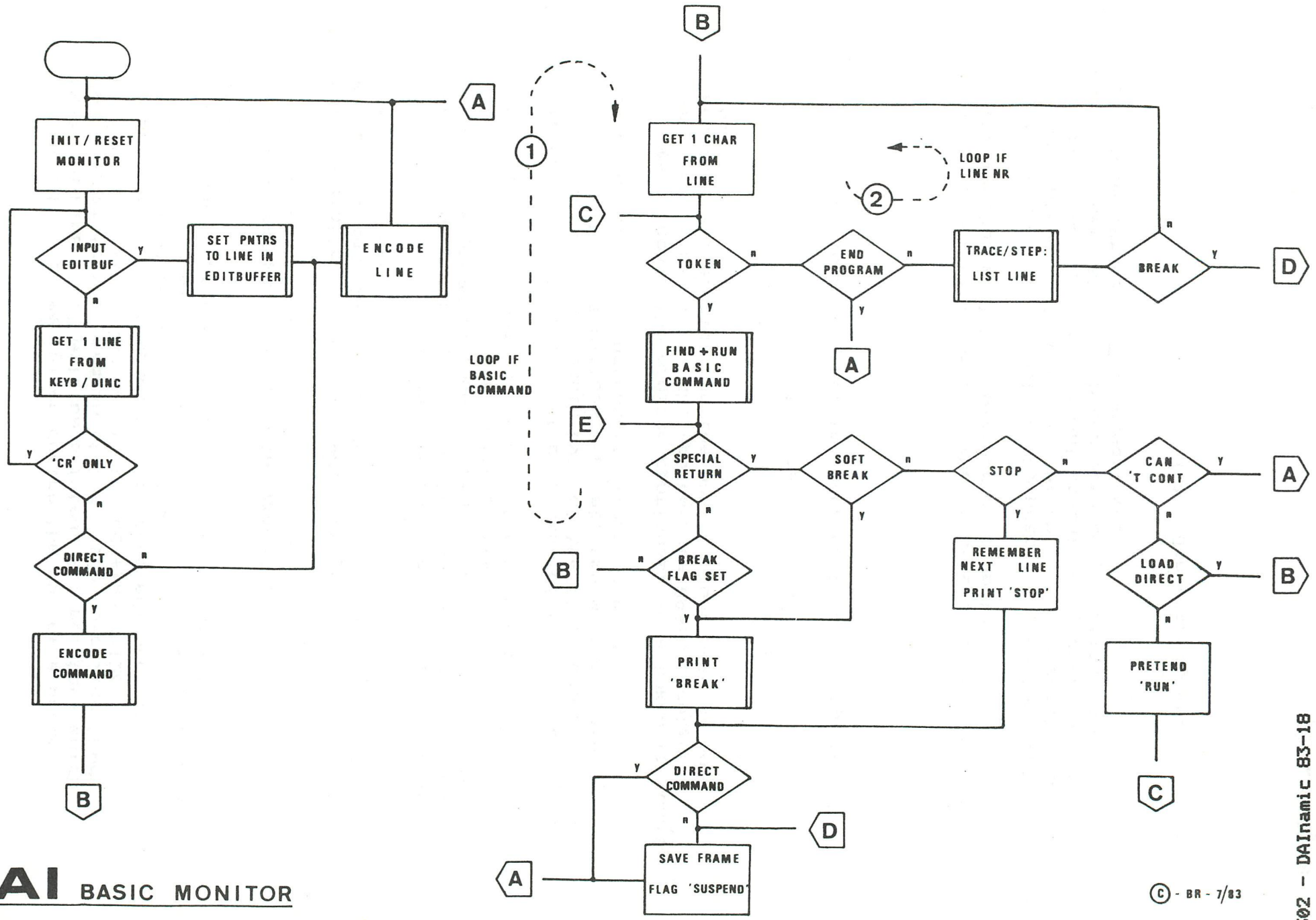
- This goes on until the end of the textbuffer - a '0' - is found. Then the monitor restarts itself.

For a direct command line, the same explication applies.

REMARKS:

In further articles, some of the details of certain parts of the monitor will be explained extendedly.

For those who are not yet in the possession of the 'DAI pC FIRMWARE MANUAL': It can still be obtained by sending an Eurocheque or an international postal money order of the amount of Hfl. 68.- to 'Micro-Service', Fabritiusstr.15, 6174 RG Sweikhuizen, The Netherlands.



DAI BASIC MONITOR

302 - DAInamic 83-18

PROGRAMMING TECHNIQUES

(from DAInamic 13, page 301)

I will start with my apologies for some errors which crept into my last article. I hope that the article was clear enough for you to correct them.

The main subject on this occasion is colours 16 to 19, with colours 20 to 23 as a subsidiary topic. Beginning with the latter, it would be a good thing if all our contributors would employ colours 20 to 23 when drawing in the 4-colour mode. It is so simple! at the start of the program we give a COLORG A B C D, with ABC & D being the 4 wanted colours. Then we give a DRAW instruction in the program using 20 if we want colour A, 21 for B, 22 for C and 23 for D. It is good practice first to give the COLORG instruction and then the MODE instruction. This avoids starting with, say, a bright yellow screen from the previous COLORG and then changing back to the wanted colour. In the 16-colour mode the first colour in COLORG is used for the background and the rim too takes on the same colour. This cannot be cleared by a FILL 0,0 XMAX,YMAX C, but only by a new COLORG followed by a new MODE 1,3 or 5. The advantage of using colours 20 to 23 lies in the fact that beginners can easily change a poor colour combination. I can make excellent use of COLORG 0 1 2 3 with my RGB monitor but I am sure viewers in black and white would be unhappy with my efforts if I changed over to 4 5 8 13.

Before we get on to colours 16 to 19 I must explain the screen arrangement in the RAM. If my explanation is inadequate refer to the articles by N J Looije or J Boerrigter in previous issues.

The screen RAM can be seen as two blocks of memory which together determine the ultimate picture. The memory banks are separated hardwarewise - the B bank is for even numbered bytes and the C bank for the odd numbered ones. This is important for the programmer.

What follows is the picture construction for the 4-colour modes. The 16-colour modes have a different structure. The colour which will be adopted by a dot on the screen is defined as follows :- two bits appropriate to each dot indicate from which of the four colour registers that dot will take its colour. Firstly, the standard case : after a reset we give a COLORG 0 3 9 14 command and then a MODE 4. The whole screen will become black and then change to MODE 4A. By going to Utility with UT we can establish with a Display that all the line control bytes show 00. (DB000 B500 for example). What happens now if we set these bytes to a different value? Let us try. POKE #B000,#AA and see 4 dots appear on the screen coloured red. For those with little experience of bit information #AA=10101010. We now follow with POKE #B002,#66 (#66=01100110). Two small red lines appear next to the dots. POKE #B001,#F0 ! Of the four red dots the leftmost two change to yellow and two blue ones come up. With these examples and a little thought we can reason out how the colouring is effected.

Even byte bit	Odd byte bit	Normal colour	Colour in example
0	0	20	0 = black
1	0	21	3 = red
0	1	22	9 = blue
1	1	23	14 = yellow

It is even possible to have more than four colours in 4-colour modes. The article by N J Looije about the video screen RAM gives much information on that topic so I will not discuss it again here. I will limit myself to establishing that in any line we want we can choose the colour control byte so that the colour registers can be changed one by one. One of the colours can be changed per line. We can take care of this with a bit of clever programming whereby we give each dot the precise colour we want while still arranging all the colours over the whole screen.

But to return to the subject. I wanted to deal with colours 16 to 19. Firstly, the colours above 16 are not real colours but special applications: colours 20 - 23 refer to the colours of the COLORG instruction and the colours 16 - 19 put bytes in the screen RAM on or off. Here we again come back to the introduction: the colour of each dot in a picture is stipulated by two bits. These two bits are in two different bytes, one having an even numbered address and the other an odd number, the odd one being higher than the even one. By now giving a DRAW instruction with colour 17 the bits belonging to the even byte are set to one. If after a reset and MODE 4, the command FILL 0,0 44,44 C is given it makes no difference whether C is 5,17 or 21. If without a reset but after another program, we use 21 it could happen that we do not get green but some other colour. However 21 is better than 5 because using 5 after another program brings with it the chance of an error message, COLOUR NOT AVAILABLE. There is no risk of that when drawing in colour 17. We will demonstrate the difference between drawings in colours 17 and 5. Key in MODE 4

```
FILL 22,22 66,66 22
FILL 0,0 44,44 5
```

and see that the green area has overwritten a part of the orange (22=10 standard) square. Tap in the instructions again but this time use colour 17 instead of colour 5 and the green area is seen to be no longer a square but a kind of L.

The part where the squares overlap is now a new square coloured white. Let us analyse that: Through the MODE 4 command we set (unless we came from MODE 4A) all the bits which designate the colour to zero. This is also valid if we had not made colour 0 the first colour of a COLORG although the screen is then another colour. The first FILL produces an orange square and sets the appropriate memory bits, the even bytes to zero and the odd bytes to one. The second FILL, using colour 5 sets the even byte bits to one and the odd byte bits which became one on the first FILL, now become zero again. The resultant colour is therefore that which was last given, in this case green (5). Because with the second FILL command we gave 17, only the bits of the even bytes will go to one and the bits of the odd bytes will NOT be changed. Thus the area where the two squares overlap, on the first FILL receives ones in the odd bytes, and with the second FILL using 17, ones in the even bytes. So this overlap square takes colour 23 (normally white). We can therefore set at one, only the bits of the even bytes, with a 17. However, with a 16 we can zero them again. Similarly with colour 19 we can set to one the bits of the odd bytes and with colour 18 switch them to zero again.

What is the significance of this to the programmer? According to the Manual we can achieve an effect of animation, but we must not expect too much from this. The effect is all right in theory but the drawing is too slow to give the appearance of film.

But we will show how this can be done. We do a drawing in colour 17 with a COLORG 0 5 x x and then see it on the screen in colour 5 (green). If we choose zero for x in the COLORG we can afterwards draw what we want in colour 19 without it being visible. Colour 19 puts ones in the odd bytes and therefore the points are either in colour 22 if the other bit is not one, or colour 23 if the other bit is one. But then the point becomes black. Also green points become black if they are drawn over. We can overcome this by letting colour 23 be green. That is, by using COLORG 0 5 0 5. When the new drawing in colour 19 is ready we give a COLORG 0 0 5 5.

All points with two bits set (colour 23) are green and all points with only the bit from the odd byte set (colour 22) are green. The points where both bits are zero (colour 20) are black and all points where only the bits from the even bytes are at one (colour 21) are black. We thus see nothing of the old drawing but all of the new. Then we erase drawing 1. Give all instructions

which had colour 17 again, but this time with colour 16 (the bits of the even bytes at zero), or if it is quicker, with a FILL 0,0 XMAX,YMAX 16. Then do a new drawing (3) in colour 17. COLORG 0 5 0 5 and drawing 3 is visible and drawing 2 is not. Erase drawing 2 with colour 18, do drawing 4 in colour 19 and then again COLORG 0 0 5 5 to make drawing 4 visible and drawing 3 disappear. Erase drawing 3 with colour 16, do drawing 5 in colour 17 and then again a COLORG 0 5 0 5, etc, etc.

In this way we can, for example, draw a car and move it across the screen. If the car takes up a lot of lines its speed is too low for the motion to be classed as animation. Even a drawing of 5 or 6 lines each with about 20 points can not reasonably be called dashing. Some people may be less critical and could find a larger number of lines acceptable. I am thinking of a large scale static drawing (in colour 23) with only a few moving parts, such as a clock. Remember though the snag which limits these animations to the 2 colours (foreground and background).

However there are applications where the facilities are suitable in DAI-basic. For example J Visser uses them very effectively in his 3-dimensional maze. The situation seen from the player's standpoint is shown on the screen. The player chooses a move, the program draws the fresh situation invisibly and then a COLORG rapidly changes the picture.

I myself have also written a maze program which by the use of colours 16-19 makes a specified part of the screen visible. One thus sees only a small part of the maze. It is done like this: The maze is drawn in colour 17, then we put a block in colour 19 (FILL A,B C,D 19). After a COLORG 0 0 14 9 the maze is invisible because colour 17 signifies even bytes and with odd bytes being zero, colour 21, thus 0=black. However where the block is, the background is yellow and where both maze and block are the colour is 23, thus 9=blue. We can let the block move by drawing a line on one side of it in colour 19 and erasing a line on the other side of it with colour 18. Another happy incidental is the possibility of giving an inverted colour to a particular part of the picture. Think about it with me! Make a drawing in colour 17 and colour in the required part (FILL) with colour 19. Then give a COLORG 0 12 12 0 and the drawing goes blue with a black background. But without the FILL this is reversed. So we can point out or make conspicuous a particular part of a picture without disturbing the remainder.

With this article are a number of programs which show how we can craftily make use of colours 16-19. [see pages 304 & 305 of DAInamic 13].

Frank H Druijff

A CHEAP & VERSATILE EPROM-PROGRAMMER

(DAInamic 13, p344/9)

The DAI personal computer has many extended input/output possibilities. With these many applications can be realised. This article deals with one such application.

WHAT IS AN EPROM ?

An Erasable Programmable Read Only Memory is a memory element which, as its name suggests, can be only read by the computer. The characteristic of Read Only Memories is that they do not lose their contents when their power supply is switched off. They are thus especially suitable for a microprocessor system which must work as soon as it is switched on. ROMs have the disadvantage that their contents must be set in the factory, by a process which is only

economically attractive for large quantities. An EPROM on the other hand has the ability of being repeatedly programmed by the user. After programming the contents can, if needs be, be erased with Ultra Violet light. Therefore it is an ideal building block for prototyping, for short runs and for the hobbyist.

PROGRAMMING AN EPROM

Nowadays most EPROMs are constructed as a matrix of 1024x8, 2048x8 or 4096x8 bits, ie, the 1K, 2K and 4K EPROM. They can be easily incorporated in a system with words of 8 bits. Fig 1 shows the EPROMs with type numbers and pin identities. In normal circumstances the EPROM is part of a microprocessor system. The microprocessor gives an address, sends a chip select and a read signal; then a memory element will be selected and its contents written, via its output lines, to the microprocessor. This data can be, for example, the next instruction for the system. The EPROM must of course have been programmed. The starting point for this is an erased IC, in which all the bits are "1". Programming is the changing of selected bits to "0" by connecting a +25V programming voltage to the Vpp pin, then giving an address and the required data. A 50 msec TTL pulse on the PROG input then programmes the EPROM. In effect the data is "burned" into the addressed memory location. By repeating this process the whole EPROM can be programmed. This is usually done by incrementing the address by 1 each time and writing a new data word. Table 1 gives the operating specification for the EPROMs. As programming takes 50 msec for each location, fully programming a 2785 takes 50 sec, a 2716 or 2516 takes 100 sec and a 2732 or 2752 takes 200 sec. A programmed EPROM can be erased but professional equipment is expensive. Erasure can however be done using a special UV-lamp, the Philips TUV 6W-E which costs about 50 Dutch guilders. If the EPROM is fastened to the lamp with an elastic band it will be wiped clean in 15 to 20 minutes. (Warning! Ultra Violet light damages the eyes, so wear red goggles or operate the lamp in a closed box).

BUILDING A PROGRAMMER

We have seen that to program an EPROM we must provide an address, a data word and the required control signals to the IC. These can all be generated by the DAI and sent via the DCE bus to the programming equipment. It is also necessary to provide the various voltages which the IC needs. The supply voltage, +5V is available from the DAI; the +25V programming voltage must be separately generated.

THE HARDWARE

The EPROM programmer needs connection to the DAI and the 25V supply (figs 2 and 3). The programming voltage is provided from a 24V AC supply, rectified, smoothed and then stabilised by a 7812 (+12V) voltage regulator which is adjusted to the correct operating level by the potentiometer connected to its base lead. The potentiometer should be set so that the output level reaches precisely +25V. Fig 2 shows which DCE bus leads are to be connected to the EPROM pins. The numbers printed nearest the IC are its pin numbers; the others are for the DCE bus. Table 2 shows how the DCE bus is used.

The schematic (fig 2) shows two relays which are switched by the DAI, via the DCE bus, each being buffered by a transistor. 12V relays were chosen as they can use the DAI's power supply. There is a 12V, 1A feed in the DAI and here only about 0.5A of it is used. It is also possible to use 24V relays, feeding them from the programming voltage, but then a larger 24V transformer would be needed to cope with the increased load. The diodes connected across the relay coils are to dissipate the voltage surge induced when the relays are switched, a surge which would destroy the transistors. One relay selects the 2732 or the other EPROMs as the 2732's connections differ from the rest. Three connections have to be switched and most programmers accomplish this with a hand operated switch which can cause mistakes. The second relay connects the programming voltage to the IC. All the components can be mounted in a small case. The base for the EPROM can be a normal IC socket but as so much care is needed to insert and

remove the ICs it is much better to use a zero-insertion-force socket.

THE SOFTWARE

The program ensures that the 8255 (DCE bus) is in the correct mode and that the required signals are sent. The software consists of both a BASIC and a machine language section. The ML program is put in the Symbol Table and then on the first RUN of the BASIC program, is moved to the work area. There are 8 single-key commands which are listed in Table 3.

Check: checks that the EPROM is in the erased state. The computer reads all memory locations and checks that they contain #FF, that is, all bits are "1". The message "OK" or "BAD" appears as appropriate.

Load/Next load: These two instructions are meant especially for use with the DNA (DAInamic Assembler). The load command loads an object file, address #3000, to the work area. The work area is a 4K block of RAM in the DAI where the data destined for the EPROM is stored. See Table 4. The load instruction always places the object file at the start address #A000 in the work area. The Next load command is used to load several files, one after the other, into the work area. A pointer stores the start address for loading subsequent files. The command is useful when several small files are to be put into the EPROM; also, when a very large assembler program needs to be loaded but it exceeds the 16K buffer size, the program can be divided and each part separately assembled. There is a snag: if there is a reference from one part to the other such as with a JMP or CALL it is necessary for at least part of the symbol table or EQU statements to appear in both parts.

Three error messages are available for both commands: GAP ERROR and CHECKSUM ERROR are identical to those in the BASIC LOADER V1.0, and in addition there is Memory range exceeded. It can happen that when loading from an object file, the memory range of the selected EPROM is exceeded; in that case there is an immediate stop and the bytes remaining are not transferred to the work area. The DNA psuedo-op REServe cannot be used - Load negates the command. The use of DNA with the programmer is summarised as follows: If an assembler program is input it can start at an arbitrary location, given by the ORG statement. It will then be assembled and written to tape as an object file (#0). The EPROM-Programmer can then be loaded after which, in Utility, the object file will be read in (>R). The Load or Next load command will transfer the file to the work area ready for the last step, programming the EPROM.

Program: Programs the EPROM with the data that is in the work area.

Read: Reads an EPROM and copies its contents to the work area.

Type: Used to select one of the 5 types of EPROM.

Utility: To jump out of the program into Utility. Useful for examining the work area of the memory or reading in an object file.

Verify: Reads the contents of an EPROM and compares it with the data in the work area. Verify permits programming to be supervised. Reports "OK" or "BAD".

The complete software consists of a source program, an associated machine language program, Basic program and the EPROM-Programmer. This is the machine language and Basic sections combined in the manner described in DAInamic 6, p135.

APPLICATIONS

The programmer can be used in the development of a microprocessor system. The combination of

DNA and DAI software presents some nice possibilities, especially for developing an 8080/8085 system. Use of the programmer need not be restricted to the 5 types of EPROM mentioned here. Altering it, for example for a 2708 is relatively simple, while with a suitable printed board it is possible to program a one-chip microprocessor like the Intel 8748. There will be more about this later if there is sufficient interest.

COMPONENT LIST

Most of the items in the component list, Table 5, need no translation. Here are the less obvious ones:- B1 Bridge rectifier B40/C1000, L1 Green LED, Z1 100mA Fuse. Extras: 2 x 12V Relays, fuse-holder, 24V 100mA mains transformer, ribbon cable with 34 pin connector, zero insertion force IC socket, case, mains lead, piece of circuit board.

G. Knoops

UNIT COLOUR MODES AND TEXT MODES

(from DAInamic 13, page 326)

Dear DAI Friends

Since the last issue of DAInamic I have had a number of reactions from members and others about the unit colour and text modes. The Handbook would appear to be neither clear nor sufficiently comprehensive. This letter and the accompanying demonstration program are therefore intended to supplement the earlier article.

According to the Handbook the DAI recognises eight resolutions, namely:

> Unit colour mode	4 resolutions	=	4 bytes
> Low resolution	88 blobs per line	=	24 bytes
> Medium resolution	176 blobs per line	=	46 bytes
> High resolution	352 blobs per line	=	90 bytes
> Super resolution	528 blobs per line	=	134 bytes

The first omission is that the book says nothing about the number of bytes per line, so the numbers shown above are the values normally assumed. Each is calculated from the number of bytes for the characters times two, with two added for the LINE MODE BYTE and the COLOUR TYPE BYTE. There will be more later about the exceptions. The arrangement of the screen hardware shows, in simplified form, the following: Firstly the highest RAM address is loaded and two instructions are read (the LINE MODE and COLOUR TYPE BYTEs). In association a counter is loaded with the number of bytes still to be read before the following instructions. At the same time readout speed, the number of repeats, the colour mode and character or graphics mode are established. When the counter has run down (from top of RAM - length of line) the next RAM address holds the following instructions which are then read: LINE MODE BYTE and COLOUR TYPE BYTE.

If for example, in MODE 0 (Super resolution), you put a line in High resolution by replacing the LINE MODE BYTE with #6A the video hardware will read 90 of the original 134 bytes, the 91st byte will be read as LINE MODE BYTE and the 92nd as COLOUR TYPE BYTE. Suppose the 92nd is a colour byte set at #00; the hardware puts the line in unit colour mode and the resolution then depends on whatever character is in the 91st byte. The result would be a repeating pattern of stripes or bits of characters for a minimum of 11 lines (44 bytes divided up into 11 unit colour mode lines) until the hardware encounters the LINE MODE BYTE of the following line. These

patterns can be changed by a COLORT x y x x but they do not go away. This is noticeable as all lines are pushed down. The technique is used in all the DEMOs of large characters, the advantage being that the screen driver moves nicely on to print the next line. The disadvantage is that the number of lines is limited and only two colours are possible. To obtain more lines of large letters the screen must be divided up anew. Herein lies the Handbook's second omission, there being some exceptions to the above mentioned calculations for the number of bytes in a line. As the unit colour mode is not dealt with in detail some observations and a table are given below.

EXCEPTIONS & PECULIARITIES

1 COLOUR BYTE
There is an important difference between character and graphics modes, namely the place of the COLOUR BYTE.
GRAPHICS > COLOUR BYTE ADDRESS = DATA BYTE ADDRESS - 1
CHARACTERS > COLOUR BYTE ADDRESS = DATA BYTE ADDRESS - 3
This means that the last character in a line gets its colour information from the COLOUR TYPE BYTE of the following line.

2 LOW & MEDIUM RESOLUTIONS
CHARACTERS > LOW res 12 data bytes total 26 bytes per line
> MEDIUM res 23 data bytes total 48 bytes per line
GRAPHICS > LOW res 11 data bytes total 24 bytes per line
> MEDIUM res 22 data bytes total 46 bytes per line.
There is thus a variation for characters.

3 INVISIBLE CHARACTERS
In all graphics and unit colour modes the whole line is visible. In character modes, apart from High resolution, the penultimate character is partly cut off and the last character is invisible. In High res the last character is half visible, obtaining its colour information from the COLOUR TYPE BYTE of the next line. This is visible in a complete picture as a small block in colour 2 of the COLORT command. (With the majority of TVs the last characters are not visible)

4 16 COLOUR CHARACTERS
For computers of revision 5 or 6 onwards this modification is inbuilt as standard. It has the same meaning as in 16 colour characters, namely that the screen hardware must first encounter a 1 in a character byte before it will change the background colour. This must be taken into account when using POKEs.

Example :-	FA	40	7F	88	44	xx	41	yx
	^	^	^	^	^	^	^	^
	LCB	CTB	DATA1		DATA2	COL1	DATA3	COL2

This makes the line colour x and the character colour y.

5 UNIT COLOUR MODE
The hardware has two ways of reading out the unit colour mode, one for MED/LOW res and the other for HIGH/SUPER res. In both cases the following 4 bytes are needed per line: LINE MODE BYTE, COLOUR TYPE BYTE + DATA + COLOUR BYTE.

In LOW/MED res the distance between the LINE MODE BYTES is two bytes while for HIGH/SUPER the gap is 4 bytes. Thus in the first case the DATA and COLOUR BYTE of the first line are respectively LINE MODE and COLOUR TYPE BYTE from the second line, so a fixed pattern. In the second case the DATA and COLOUR BYTE chosen are filled in.

The table below summmarizes the effect of CONTROL BYTES.

Control bytes	Display	Colours	Resolution	Bytes to new control byte	Data and col. bytes	Visible fields and blobs
0x 40	GRAPHICS	4	LOW	24	11	11 - 88
1x 40	GRAPHICS	4	MEDIUM	46	22	22 - 176
2x 40	GRAPHICS	4	HIGH	90	44	44 - 352
3x 40	GRAPHICS	4	SUPER	134	66	66 - 528
4x 40	CHAR	4	LOW	26	12	10 - 87
5X 40	CHAR	4	MEDIUM	48	23	21 - 173
6x 40	CHAR	4	HIGH	90	44	43 - 345
7x 40	CHAR	4	SUPER	134	66	64 - 519
8x 40	GRAPHICS	16	LOW	24	11	11 - 88
9x 40	GRAPHICS	16	MEDIUM	46	22	22 - 176
Ax 40	GRAPHICS	16	HIGH	90	44	44 - 352
Bx 40	GRAPHICS	16	SUPER	134	66	66 - 528
Cx 40	CHAR	16	LOW	26	12	10 - 87
Dx 40	CHAR	16	MEDIUM	48	23	21 - 173
Ex 40	CHAR	16	HIGH	90	44	43 - 345
Fx 40	CHAR	16	SUPER	134	66	64 - 519
xxxxxxxxxxxxxxxxxxxxxxxx			UNIT COLOUR MODE	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx		
0x 00	GRAPHICS	4	LOW	2	1	11 - 88
1x 00	GRAPHICS	4	MEDIUM	2	1	22 - 176
2x 00	GRAPHICS	4	HIGH	4	1	44 - 352
3x 00	GRAPHICS	4	SUPER	4	1	66 - 528
4x 00	CHAR	4	LOW	2	1	11 - 88
5x 00	CHAR	4	MEDIUM	2	1	22 - 176
6x 00	CHAR	4	HIGH	4	1	44 - 352
7x 00	CHAR	4	SUPER	4	1	66 - 528
8x 00	GRAPHICS	16	LOW	2	1	11 - 88
9x 00	GRAPHICS	16	MEDIUM	2	1	22 - 176
Ax 00	GRAPHICS	16	HIGH	4	1	44 - 352
Bx 00	GRAPHICS	16	SUPER	4	1	66 - 528
Cx 00	CHAR	16	LOW	2	1	11 - 88
Dx 00	CHAR	16	MEDIUM	2	1	22 - 176
Ex 00	CHAR	16	HIGH	4	1	44 - 352
Fx 00	CHAR	16	SUPER	4	1	66 - 528

These are the right values for correct screen construction. Here too is a demonstration program showing all the possibilities.

With friendly greetings N. P. LOOIJE

BANK 0:#C000 to #EFFF

C000 Entrypoints
C035 Package Initialisation
C04B Load (00D0/1) on stack and continue on this address
C05E Continue on (00D2/3)
C06C Continue on (00D6/7)
C073 Continue on (00D2/3)
C079 Floating point compare
C0AC Integer compare
C0BB Increment integer number in memory
C0D5 Decrement integer number in memory
C0F3 Increment floating point number in memory
C1FB Decrement floating point number in memory
C21E Save FPAC on stack
C234 Retrieve FPAC from stack
C249 Input floating point number to FPAC
C361 Convert a floating point number for output
C486 Pretties up FPT or INT number
C51A Move 11 bytes 1 memory location higher
C531 Move bytes one memory location down
C573 Input integer number to FPAC
C5B2 Convert integer number for output
C614 Input hex number to IAC
C653 Convert IAC to hex for output
C6C0 Math. restart (RST 4)
C6CF ROM bank switching
C6F2 Switch to ROM bank
C6FD Screen restart (RST 5)
C70E Utility/encode (RST 1)
C719 Reset
C7A8 Initialisation screen data
C7E0 Initialisation parameters
C7FB Check for highest RAM addr
C80C Start from scratch
C918 Encoding when line number is given
C957 Error while encoding a stored line
C9D1 Initialise program buffers
CA01 Memory management routine
CA25 Emergency stop routine (Graphics modes)
CA34 Find string BASIC instruction in table
CB23 Empty HEAP
CB9E Load 0 into 4 consecutive memory locations
CBBF Strings - BASIC commands
CD64 basiccmd TALK
CD8B Pointers to strings of BASIC commands
CE70 Print ','
CE75 Print a string between spaces
CEB5 Change screen MODE
CEC6 Set HEAP size to default value
CECF Print DAI PERSONAL COMPUTER in medium resolution
CEE4 Select ROM bank 0 , print message
CEF9 Print 'COMPUTER' under 'DAI PERSONAL'
CF02 Pointers to routines BASIC-COMMANDS
CF86 Strings arithmetic and logical operators

D101 Reset pointer 0000H
D106 Add string data after end other string
D121 Compare two strings
D16D Transfer of string data
D195 Organise program buffers
D1DB Keyboard scanning: Check source and input
D236 Set MSB of byte
D23D basiccmd SAVE
D270 basiccmd LOAD
D2A8 Loading error
D2B8 Open tape file
D2C3 basiccmd CHECK
D2F1 Write block on tape
D30F Write byte,update checksum
D316 Write block length, update checksum
D325 Start file reading
D340 Read block
D384 Read byte, calculate checksum
D38D Read name length
D3A2 Read + check program name and file type
D3F4 Read file header
D40C Write file leader
D422 Write file trailer
D42E Start cassette motors
D445 Stop cassette motors
D453 Read bit
D480 Read leader
D4D4 Read byte
D4ED Write leader
D509 Write byte
D524 Write bit
D53C Write cycle
D550 Write trailer bits
D560 Initiate keyboard pointers
D578 Keyboard interrupt service (RST 6)
D59A Scan keyboard, store result
D606 Break
D620 Complete keyboard scan
D632 Get ASCII value of key pressed
D63F Get key-ASCII and store it
D642 Check if output to RS232
D668 Write 2 blocks + trailer on tape
D68D Cursor handling
D695 Print character
D69C Keyboard scanning: Check if ASCII-buffer full
D6A5 Keyboard scanning: Check if new inputs
D6BB Keyboard scanning
D6DA Wait for spacebar
D71A Stop loading programs
D720 Initiate writing file leader
D72D Init SOUND gen,GIC,start HEAP,transfer CASSETTE data
D750 Check break pointer
D755 SOUND interrupt (RST 3)
D783 Failure during ropen
D78A Check if LOAD during run program
D790 Check free RAM space

D795 Transfer CASSETTE vectors
D7A4 DISC/CASSETTE switching vectors
D7D8 Write block + trailer on tape
D7DE Failure during ropen
D7EB Write byte on CURSOR pos addr and update CURSOR pos
D7F8 SAVE : Write name length
D7FF Initialise loading from tape
D81D basiccmd SAVEA
D85E basiccmd LOADA
D879 Input from edit buffer
D8A6 Initiate SOUND generator
D8C8 Output to DCE-BUS
D8E0 Input from DCE-BUS
D8FB Initialise interrupt system
D949 Set up interupt vector area
D96B Interrupt vector routine
D973 Update TICC interrupt mask
D9A9 Clock interrupt (RST 7)
D9CD General interrupt return
D9DB Enable clock interrupt
D9E2 Stack interrupt (RST 2)
D9F5 Error handling
DA0B Pointers to error messages
DA3D Error while running program/inputs
DA50 Print error message
DA64 Error when not running program
DA75 Print line number in which error occured
DA94 Pointers to error messages
DAD4 Print message
DB32 Print string
DB44 Print string message
DB6F Strings for maching messages
DD1A Scan keyboard and print characters
DD55 Get X-coordinate CURSOR position
DD60 Output routine
DD94 Output to RS232
DDB4 Input from RS232
DDBA RS232 frame error
DDD1 Get character from line, neglect TAB + SPACE
DDE0 Get character from line
DE02 Check if upper case character
DE09 Check if character is number or upper case
DE14 Compare (HL) and (DE)
DE1A Calculate length of block
DE26 Two complement of 16-bits data
DE30 Calculate off-set address
DE39 Calculate address after sting
DE41 Delay routine
DE4F Data block transfer
DE7C Load bank with identical data
DEB5 basiccmd NEW
DED5 basiccmd CONT
DEFE basiccmd STEP
DF03 basiccmd STOP
DF0C basiccmd END
DF15 basiccmd IF

```
DF20  basiccmd IF
DF2A  basiccmd GOSUB
DF4C  basiccmd RETURN
DF63  basiccmd GOTO
DF6A  basiccmd ON
DF71  basiccmd ON
DF9E  basiccmd RUN
DFBA  basiccmd RUN
DFC0  basiccmd POKE
DFC9  basiccmd OUT
DFD5  basiccmd WAIT
DFF7  basiccmd WAIT mem
E016  basiccmd WAIT TIME
E02B  basiccmd FOR
E0C5  basiccmd NEXT
E0E5  basiccmd NEXT
E13C  Save on soft break  ??
E167  Restore pointers to continue
E18F  basiccmd DATA - REM
E197  basiccmd LIST
E1AA  basiccmd LIST
E1B6  basiccmd LIST
E1F5  Run edit commamd
E253  List edit
E25C  List edit
E265  Initialise EDIT buffer
E2B3  basiccmd PRINT
E2FC  basiccmd INPUT
E323  basiccmd READ
E401  basiccmd RESTORE
E45A  basiccmd LET
E4BC  basiccmd SOUND
E50C  basiccmd NOISE
E570  basiccmd ENVELOPE
E5B2  basiccmd CURSOR
E5BB  basiccmd MODE
E5C1  basiccmd DOT
E5CE  basiccmd DRAW
E5D7  basiccmd FILL
E60E  basiccmd COLORT
E615  basiccmd COLORG
E62F  basiccmd DIM
E69E  basiccmd UT
E6A4  basiccmd CALLM
E6B5  basiccmd CLEAR
E6CE  basiccmd TRON
E6D5  basiccmd TROFF
E8EE  Table of jumps to INT/FPT operator routines
E9D9  Calculate start address routines
E9F0  Pointers to start address routines
EB51  Calculate free RAM space
EBC1  Paddle operation (?)
ECCC  Print instruction from token
ECF8  Pointers to start address routines
EE8D  Set input direction
EFED  Print string
```

EFF5 Print character

BANK 1:#E000 to #EFFF

E000 Entry points
E0FE FPT multiplication
E108 FPT division
E126 Store registers ABCD in RAM 00D5-00D8
E133 Restore registers ABCD from RAM 00D5-00D8
E16D Integer addidion
E18D Integer subtraction
E1AC Integer multiplication
E22B Integer division
E3CF Store registers BCDE in RAM 00D5-00D8
E855 FPT power
EDAA FPT addition
EDB4 FPT subtraction

BANK 2:#E000 to #EFFF

E000 Entry points
E030 Screen parameters MODE 0
E045 Screen parameters MODE 1/1A
E05A Screen parameters MODE 2/2A
E06F Screen parameters MODE 3/3A
E084 Screen parameters MODE 4/4A
E099 Screen parameters MODE 5/5A
E0AE Screen parameters MODE 6/6A
E0C3 Initialize screen
E102 Output one character
E13D Output carriage return
E159 Output form feed
E166 Output backspace
E1A9 When end of line reached
E1CB Scrolling
E1FD Initialise screen character area
E21C Change to character MODE
E237 Set text colours
E254 Set colour parameters
E267 Load colours in header/trailer area
E279 Set cursor position
E2CC Ask cursor position and size character screen
E316 Set cursor mode
E330 Update cursor pointers
E344 Flash cursor
E36B Load screen location pointed at by cursor
E38B Get character from line
E3D9 Change MODE
E407 Set up screen RAM area
E545 Load pointers with screen parameters
E59A Start addresses tables screen parameters
E5A6 Initialise memory management routine
E5FC Initialise header
E687 Place cursor on beginning of line
E6A4 Set graphics colours
E6F2 (HL) = (HL) - (DE)

E6FB Compare DE - HL
E701 Add offset to address
E706 Two complement of 16-bits data
E710 Draw a DOT on screen
E71B Draw a line on the screen
E818 Fill a rectanglar area on the screen
E884 Ask colour of point and size of graphics screen
EB46 Calculate number of bytes on extended lines
EBF4 Initialise editor
EC1E Obey edit
EC4B Window up
ECB3 Window down
ECF8 Window right
ED50 Window left
ED88 Cursor up
EDAB Cursor down
EDD2 Cursor left
EDF6 Cursor right
EF4B Insert character in buffer
EFCC Delete character in buffer

BANK 3:#E000 to #EFFF

E000 Entry points
E018 Update input pointer
E024 Encode inputs without line number
E04F Check if character after basiccmd is ':' or car.ret.
E09A Get address encoding instruction and go to it
E2E6 Get char. from line, check if an upper case char.
E2F1 Strings variable types
E81F Set D depending on contents of A
E859 Check statement terminator
E8C5 ASCII table upper case
E8FD ASCII table lower case
E935 Get inputs from keyboard or RS232
E93F Load ASCII value for key pressed in buffer
EA00 Start UTILITY
EA0D Initialise UTILITY
EA62 Error
EA74 Entry from BASIC
EA7D Initialisation parameters
EA8E Table with UTILITY commands
EAB3 D - Display
EADB Handle inputs after a UT command
EB15 Convert ASCII-character to HEX-value
EB26 L - Look
EB41 Set look windows, start look
EB56 Go/Look check for carriage return
EB5D Restart 0 (RST 0)
EC45 Compare DE with HL
EC63 Restore CPU registers
EC7C Calculate DE - HL
EC83 M - Move
ECBA Z - Reset
ED01 Print space
ED06 Scan keyboard, print character

```
++++++++++++++++++++++
+ SFGT  POKE - NOTES +
++++++++++++++++++++++
```

POKE :

768,POS	hor. position for mode 1 - 4
769,0	
768,POS MOD 256	hor. position for mode 5 - 6
769,POS / 256	
770,POS	ver. position
771,STEP	hor. position between two caracters
772,STEP	ver. position between two caracters
773,STEP	hor. step by ver. CR
774,STEP	ver. step by hor. CR
775,MIN MOD 256	hor. min.
776,MIN / 256	
778,MAX MOD 256	hor. max.
779,MAX / 256	
777,MIN	ver. min.
780,MAX	ver. max.
781,BEGIN	matrix begin
782,END	matrix end
783,BEGIN	begin of the string
784,LENGHT	lenght of the string
785,X	x = 0 : display right- up
	x = 1 : display left- up
	x = 2 : display right- down
	x = 3 : display left- down
	+4 : retain hor. position
	+8 : retain ber. position
	+16 : upside- down
	+64 : high speed
	+128 : inkey flag
786,COL	for-& background color in 16 color mode
	number of colorg in 4 color modes
787,DELAY	gives delay / caracter
788,CHAR	number of characters / image
789,HORSTAP	horstap / character
790,TABEL	tabel choise
791,NUMBER	the number of characters to be combined to one character

mikroshop hageland

COMPUTERS en ALLE TOEBEHOREN

Herseltsesteenweg 103 - 3220 AARSCHOT

tel. 016/56 87 70

OPEN : dinsdag - vrijdag 14 u - 20 u.
zaterdag 9 u 30 - 17 u.
GESLOTEN : zondag, maandag, feestdagen.

1.	MEMOCOM MDCR met TOS ,kabel en doos MDCR cassettes,handleiding	17000 BF
2.	DAI pc ,snoeren,manual,demo cassette en abonnement DAInamic	45900 BF
3.	DAI pc ,idem 2 ,MDCR met TOS,kabel,doos MDCR cassettes	61000 BF
4.	DAI pc, idem 2 ,Kleurmonitor met RGB ingang, RGB kabel	64000 BF
5.	DAI pc, idem 2 ,Kleurmonitor,RGB kabel,MDCR met TOS,kabel	74000 BF
6.	DAI pc, idem 2 ,Kleurmonitor,RGB kabel,DAI floppy 2x80K form.	97700 BF
7.	DAI pc, idem 2 ,Kleurmonitor,RGB kabel,DAI floppy 2x160K form.	102900 BF
8.	DAI pc, idem 2 ,Kleurmonitor,RGB kabel,DAI floppy 2x320K form.	112200 BF

DAI floppy 2 x 80K form.	54470 BF
DAI floppy 2 x160K form.	59000 BF
DAI floppy 2 x320K form.	73600 BF

DAI Hardware manuals Deel I+II	1140 BF
DAI Software manuals Deel I+II	2285 BF
DAI pc testmanual	1785 BF
Draagtas voor DAI pc	1130 BF
Firmware manual(ROM listing)	1290 BF
Grafische interface voor DAI pc	5900 BF

Monitors

BARCO : *	42 cm RGB kleurmonitor met RGB en video ingang sound weergave via ingebouwde luidspreker;met RGB kabel	24500 BF
*	42 cm RGB monitor | TV combinatie met afstandsbediening RGB,video en UHF ingang;ingebouwde luidspreker voor sound	32900 BF
INDATA :*	KAGA medium resolution monitor 380 dots,kleur,31 cm geen ingebouwde sound,RGB ingang,met RGB kabel	21900 BF
*	KAGA super high resolution ;630 dots;kleur ; 31 cm RGB ingang ,geen ingebouwde sound ;met RGB kabel	38000 BF
	RGB kabel apart voor monitor met RGB ingang (prise PPT)	990 BF

Printers

: alle printers zijn grafische interface voor DAI pc inbegrepen.

EPSON : *	RX-80 (tractor feed)	80 kar.100 cps	33500 BF
*	RX-80 F|T (friction + tractor feed)	80 kar.100 cps	
*	FX-80 (friction + pinfeed)	80 kar.160 cps	46900 BF
*	MX-100 Type III F|T (friction|tractorfeed)	132 kar.100 cps	54900 BF
STAR : *	STX -80 Thermal printer (frictionfeed)	80 kar. 60 cps	18500 BF
*	GP 510 (friction|tractorfeed) 2K buffer	80 kar.100 cps	30500 BF
*	GP 515 (friction|tractorfeed) 2K buffer	132 kar.100 cps	40900 BF
Brother : *	CE-50 BT Bidirectionele Daisy Wheel schrijfmachine printer en plotter ,2K buffer,proportioneel	14 cps	42900 BF

BTW 19 % inbegrepen Prijswijzingen voorbehouden 15.10.83

REAL CIRCLES

OPMERKINGEN OVER "INCREMENTAL CIRCLE GENERATION"
DOOR F.VAN AMERONGEN.(CF.DAInamic 14)
****

Dit klein programma is inderdaad knap. De hoge snelheid van tekening wordt bekomen door het vermijden van de conventionele sinussen en cosinussen. Maar het gebruikte algorithme is NIET afkomstig van de cirkelvergelijking.Het gaat hier inderdaad om ELLIPSEN met nagenoeg gelijke assen,schuins liggende met een hoek van 45 graad op de x-as. Dit kan iedereen gemakkelijk vaststellen indien K1 in lijn 30 door een hogere waarde (b.v.=30) vervangen wordt.(Indien een "number out of range" bekomen wordt dient de oorspronkelijke waarde van R verkleind te worden). De enige punten van deze ellipsen die ook tot de cirkels behoren zijn 0,R),(R,0), (0,-R) en (-R,0).

Om een echte cirkel te bekomen dient men gebruik te maken van de parametrische vergelijkingen:

$x = R \cos\theta$ ..... $y = R \sin\theta$

,die door differentieren het volgende geven:

$dx = -y\,d\theta$ ..... $dy = x\,d\theta$

Dit geeft aanleiding tot:

$x_2 \simeq x_1 - y_1 \Delta\theta$ ..... $y_2 \simeq y_1 + x_1 \Delta\theta$

waar $\Delta\theta$ ..een kleine hoekverschil is, en $\simeq$."ongeveer gelijk tot" betekent.

Het vertrekpunt (x_1 , y_1) voldoet aan de vergelijking

$x_1^2 + y_1^2 = R^2$

doch men stelt vast dat punt (x_2 , y_2) niet op de cirkel ligt:

$x_2^2 + y_2^2 = R^2(1 + \Delta\theta^2)$

Men ziet dat elk bekomen punt coordinaten heeft die elk te groot zijn met een factor

$\sqrt{(1 + \Delta\theta^2)}$

,zodat het iteratie-process een "accumulatiefaktor" van $\left(\sqrt{(1+\Delta\theta^2)}\right)^{n-1}$ zal produceren, waar n het aantal zijden van de benaderende veelhoek voorstelt:

$n = \frac{2\pi}{\Delta\theta}$

,en een spiraalkromme i.p.v. een cirkel bekomen zou worden.

Ten einde een cirkel te bekomen dient dus elk nieuw coordinaat door factor

$\sqrt{(1 + \Delta\theta^2)}$

gedeeld te worden.

Hierbij een programma dat duidelijk het verschil laat zien tussen de "pseudo-cirkels" van F.van Amerongen en echte wiskundige cirkels.

G.Doumont

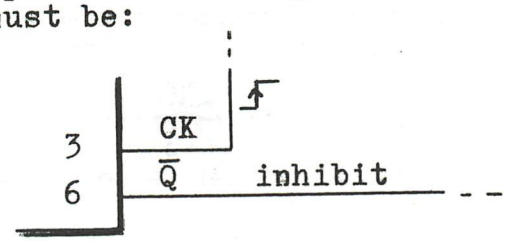
CIRKELS...DIE GEEN (ECHTE) CIRKELS ZIJN

```
5     GOTO 2000
10    PRINT CHR$(12):MODE 0
15    CLEAR 2000:DIM U(190.0),V(190.0)
20    COLORG 0 5 10 15:R=90.0
35    MODE 6:XC=XMAX/2.0:YC=YMAX/2.0
40    DRAW XC,0 XC,YMAX 5:DRAW 0,YC XMAX,YC 5
50    X1=R:Y1=0.0:K1=3.0
55    FOR K1=3.0 TO 31.0 STEP 7.0
60    K=K1/X1:A=SQR(1.0+K*K):B=1.0/A:C=5.0
70    FOR I=0.0 TO (2.0*PI)/K:X2=(X1+K*Y1)*B:Y2=(Y1-K*X1)*B
80    GOSUB 1000:NEXT:WAIT TIME 50
110   FOR I=0.0 TO (2.0*PI)/K:X2=X1+K*Y1:Y2=Y1-K*X2:C=10.0:U(I)=X2:V(I)=Y2
120   GOSUB 1000:NEXT:WAIT TIME 100
122   IF K1=31 GOTO 130
124   C=0.0:DRAW R+XC,YC U(0.0)+XC,V(0.0)+YC C
126   FOR J=1.0 TO (2.0*PI)/K:DRAW U(J)+XC,V(J)+YC U(J-1.0)+XC,V(J-1.0)+YC C
128   NEXT J
130   NEXT K1:WAIT TIME 250:GOTO 10
1000  P=X1:Q=Y1:M=X2:N=Y2:DRAW P+XC,Q+YC M+XC,N+YC C
1100  X1=X2:Y1=Y2:RETURN
2000  PRINT " De cirkel in 't groen is de 'echte wiskundige"
2010  PRINT " cirkel'.In oranje kleur wordt de 'Van Amerongen"
2020  PRINT " cirkel' getekend. Voor elke waarde van K1 tussen"
2030  PRINT " 3 en 31 met step 7 worden beide cirkels op het"
2040  PRINT " scherm voorgesteld en wordt de oranje cirkel daarna"
2050  PRINT " afgeveegd. Op deze manier komt het 'ovaliseren' van"
2060  PRINT " de oranje veelhoeken zeer duidelijk te voorschijn"
2070  WAIT TIME 750:GOTO 10
4000  PRINT CHR$(14);"CIRKELS...DIE GEEN (ECHTE) CIRKELS ZIJN":LIST
4010  PRINT
```

CORRECTIONS DAI SCHEMATICS - 2

Some failures have been found in the DAI schematics. It concerns sheet 4, Timing.

- IC 37 : It is not a 74LS150, but a 74LS170.
- IC 18 : Some pins are exchanged. The correct lay-out must be:



The wiring to the IC remains as drawn.

Please update your copies accordingly.

Jan Boerrigter - Febr. 1983

E R R A T U M

A bug was reported in the PROGRAMGENERATOR in the last issue of DAINAMIC. Because I sent a wrong programlisting to the editor. Line 140 should be:

```
140 POKE #135,2:CALLM #F800:POKE #135,0:RETURN
```

This will solve the problem of SYNTAX ERROR IN LINE 140 when an INPUT statement is used.

N.P. Looije

CONVERTING A PROGRAM TO 'IMP INT'

As you all will probably know by now (from Frank Druijff) a program runs faster when all variables and constants are integer. Also I have never met (until now) 1.0 person as if there is also 1.247 person. Though some programs do try to make you believe this. Here is a little method to change all variables AND constants (with .0) to integer. Just type this when your program is in memory and your program will be converted to IMP INT. The 2 lines program search for .0 and substitutes it by spaces.

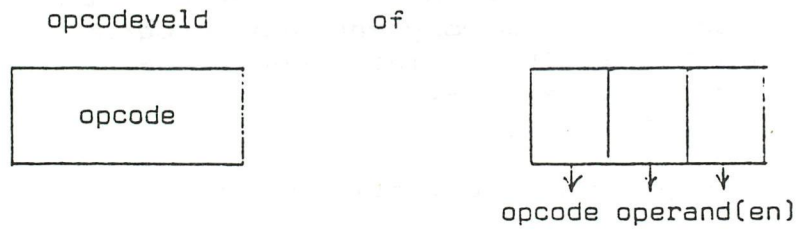
```
*IMP FPT
*CLEAR (size of program + a little more)
*EDIT
<BREAK> <BREAK>
*IMPINT
*NEW
*65534 FOR A=PEEK(#A2)+PEEK(#A3)*256 TO PEEK(#A4)+PEEK
       (#A5)*256:IF PEEK(A)=46 THEN IF PEEK(A+1)=48
       THEN POKE A,32:POKE A+1,32
*65535 NEXT
*RUN 65534
*NEW
*POKE #135,2
wait a little while and your program
is in IMP INT  inclusive the constants
```

N.P.Looije

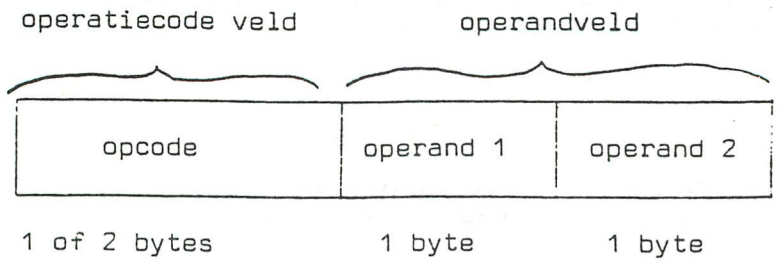
CURSUS MICROPROCESSOREN PART 2

Het getal FFFF in hexadecimaal stelt inderdaad het binair adres 1111 1111 1111 1111 voor en komt overeen met het adres van de geheugenplaats 65535 ($2^{16}-1$).
De benaming van de verschillende adresseringsmethodes is niet dezelfde bij alle constructeurs wat aanleiding kan geven tot verwarring.
De adressering beperkt zich natuurlijk niet alleen tot geheugenplaatsen. Het is eveneens mogelijk bepaalde registers van de microprocessor aan te spreken of te adresseren of eenvoudig de te verwerken data (gegevens) aan te duiden.
Het is meestal hetzelfde register van de microprocessor dat gebruikt wordt als adresseringsregister. Onder adresseringsregister verstaan we het register dat het adres bevat van de te adresseren geheugencel.
Men spreekt over een programmateller (PC : *programcounter*) wanneer het adresseringsregister verwijst naar een instructie die zich in het programmageheugen bevindt.
Betreft het de adressering in het werkgeheugen (data geheugen) dan spreekt men ook van een basisregister (*base pointer*).
In adresseringstechnieken worden eveneens indexregisters gebruikt. De inhoud van het indexregister dient opgeteld te worden bij de inhoud van het basisregister om het eigenlijke (physische) adres te vinden.
In het werkgeheugen is meestal bij de multi-chip processoren, een zone gereserveerd om als stapelgeheugen (*stack*) gebruikt te worden. In het stapelgeheugen kan de microprocessor gegevens onderbrengen die nodig zijn om bijvoorbeeld een programma te kunnen verder zetten na een onderbreking (*interrupt*) of bij het verwerken van subroutines.
Als adresseringsregister van de stack gebruikt men bijvoorbeeld de stapelwijzer (SP : *stack pointer*).
Een instructie bestaat uit een deel dat de aard van de bewerking aangeeft, operatie code veld of kortweg opcode genoemd, meestal gevolgd door een deel dat aangeeft waarop de bewerking betrekking heeft, operandveld genoemd. Dit veld kan meerdere operanden bevatten. De operand kan een geheugen of input/output adres zijn, een register of ook een getal.
In het algemeen bestaat een instructie aldus uit 1, 2, 3 of 4 bytes :

1 byte : voor instructies zonder operand waarbij de operand(en) in het opcodeveld begrepen is (zijn).



2 bytes : voor instructies met 1 byte opcode en 1 byte operand
3 bytes : voor instructies met 1 byte opcode en 2 bytes operandveld
4 bytes : voor instructies met 2 bytes opcodes en 2 bytes operandveld



Als belangrijkste adresseringswijzen onderscheiden we impliciete, directe, indirecte, onmiddellijke, relatieve en geïndexeerde adressering,

waarbij tevens onderlinge combinaties mogelijk zijn zoals bv. geïncexeerd indirect enz.
In hetgeen volgt bespreken we deze methodes met telkens een aanduiding van een instructievoorbeeld voor de vier processor families. In de mate van het mogelijke zullen we de verschillende namen geven die gebruikt worden voor eenzelfde adresseringsmethode. Vermelden we tenslotte dat we ons hier beperken tot de 8 bits processoren. Voor de aanvullende adresseringsmethoden bij de 16 bits processoren verwijzen we naar de studie van de desbetreffende processoren.

1.5.1. Impliciete adressering (*implied*)

Bij de impliciete adressering is de instructielengte 1 byte zodat de volledige informatie voor de instructieverwerking hierin is opgenomen. Hierbij merken we een onderscheid tussen inherente (*implied*) adressering en register adressering. Bij inherente adressering is de operand overbodig, bijgevolg is de bewerking volledig bepaald door de combinatie van acht bits in het opcodeveld.

Voorbeelden

Bij 8085 STC $(37)_{Hex}$ (*set carry*)
De overdrachtsflipflop (*carry flag*) wordt op 1 gezet
(1→CY)

Bij 6800 SEC $(0D)_{Hex}$ (*set carry*)
(1→CY)

Bij Z80 SCF $(37)_{Hex}$ (*set carry flag*)
(1→CY)

Bij 6500 CLC $(18)_{Hex}$ (*clear carry*)
(0→CY)

Bij de registeradressering is de naam van een register (paar) of van meerdere registers opgenomen in de operand(en). Ook hier is de instructielengte slechts 1 byte lang, zodat zowel opcode als operand(en) gevormd worden door een 8 bits woord. Dergelijke adressering is interessant bij instructies met betrekking tot veel gebruikte registers zoals de accumulator(en) of registers van algemeen nut zoals de programmatellers (PC), de stapelwijzer (SP), kladblokregisters, indexregisters enz.
Ook rekenkundige en logische instructies die uiteraard gebeuren met de inhoud van de accumulator, maken meestal gebruik van deze vorm van adressering. Hierbij is de accumulator inherent geadresseerd en wordt beschouwd als een tweede operand.

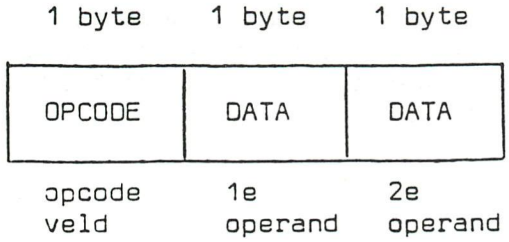
Voorbeelden

Bij 8085	MOV	r_1, r_2	De inhoud van register 2 wordt verplaatst naar register 1. $(r_2) \rightarrow r_1$
	ADD	r	De inhoud van register r wordt opgeteld met de inhoud van de accumulator en het resultaat komt in de accumulator. (A)+(r)→A Het register r vormt de eerste operand en de accumulator de tweede. Deze is niet genoemd in de instructie en is inherent.

PCHL (HL *to stackpointer*)
De inhoud van het registerpaar H(igh) L(ow) wordt in de programmateller gebracht
(HL)→PC

1.5.2. Onmiddellijke adressering (*immediate*)

Bij onmiddellijke adressering zijn de gegevens (*data*) die door de instructie dienen verwerkt te worden, in de opdracht zelf opgenomen. Ze volgen onmiddellijk na de instructiecode als tweede eventueel derde byte. Sommige auteurs spreken alleen van onmiddellijke adressering als de gegevens slechts 8 bits lang zijn (1 operand).
Voor 16 bits gegevens spreken ze van *extended immediate adressing* (uitgebreide onmiddellijke adressering) (2 operanden).



Door deze vorm van adresseren wordt geen geheugenruimte bespaard maar verhoogt de verwerkingssnelheid.

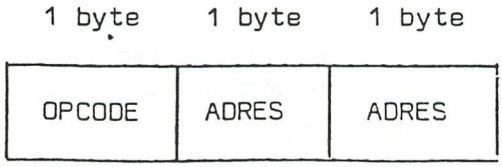
Voorbeelden

Bij 8085 MVI r,d8 (*move immediate*)
Het 8 bits datawoord (d8) wordt in het register gebracht.
d8→r

LXI SP,d16 (*load registerpair immediate*)
Het 16 bits woord (d16) komt in de stack pointer.
d16→SP

1.5.3. Directe adressering (*direct*)

Bij de directe adressering bevat de operand het adres van de te verwerken gegevens. Met 1 byte kunnen 256 plaatsen geadresseerd worden van (00)00 tot (00)FF. De hoogste byte van deze 256 adressen is (00) waardoor men ook spreekt over adressering in pagina nul (*zero page*).
Indien het operand veld bestaat uit 2 bytes kunnen alle 64 K plaatsen geadresseerd worden zodat hiervoor ook de benaming uitgebreide directe adressering (*extended direct*) gebruikt wordt of soms ook absoluut direct.



Merken we ook hier op dat in machinetaal (hex notatie) bij de 8085, de Z80 en de 6500 eerst de laagste byte van het adres geschreven wordt, gevolgd door de hoogste byte, terwijl dit omgekeerd is voor de 6800.

Voorbeelden

Bij 8085 STA 2000H *(store accumulator)*
De inhoud van de accumulator wordt in geheugenadres $(2000)_{Hex}$ geplaatst
(extended)
(A)→M

1.5.4. Indirecte adressering *(indirect)*

Bij de indirecte adressering wordt in de operand niet het adres zelf maar de plaats (register) aangegeven waar het adres zich bevindt.
Het register(paar) dat het adres bevat kan zich zowel binnen de microprocessor als in het geheugen bevinden.
Bij de 8085 processor bevindt dit registerpaar zich in de processor (HL registerpaar). We spreken dan van register indirecte adressering, bijvoorbeeld :

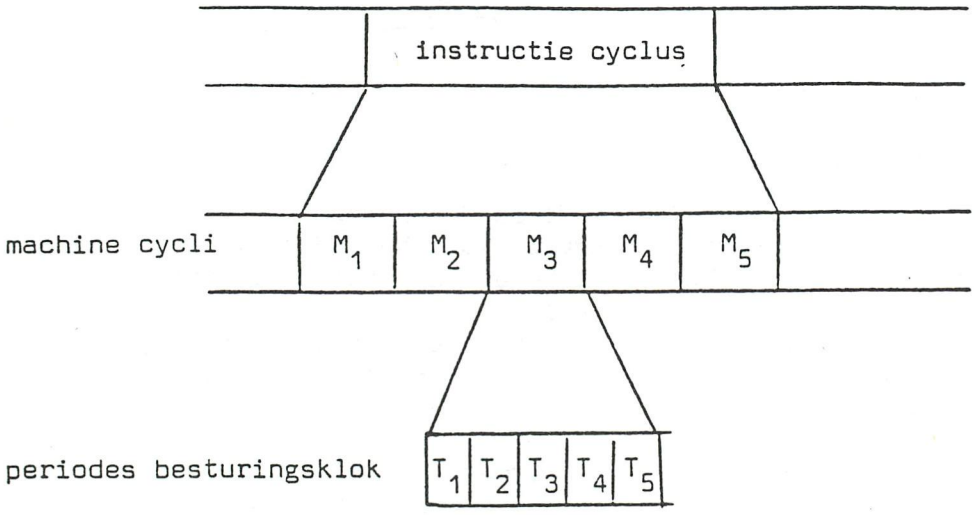
MOV M,A *(move A to M)*
De accumulatorinhoud wordt in het geheugen gebracht met de plaats aangeduid door de inhoud van het registerpaar HL.

Deze instructie is 1 byte lang zodat de eigenschappen dezelfde zijn als deze van de register adressering onder paragraaf 1.5.1.
Het registerpaar HL moet echter ook geladen worden.
Bij de Z80 wordt dit LD r,(HL).
Bij de 6500 bevindt het register zich in het uitwendig geheugen. Voor het bepalen van de geheugenplaats zijn 2 extra bytes nodig.

1.6. Samenstelling van een instructie in fuctie van de tijd

Om de werking van de microprocessor te begrijpen, is het van het grootste belang te weten hoe een programma instructie behandeld wordt.
Een programma instructie of kortweg instructie is ergens opgeborgen in het programmageheugen. De behandeling van de instructie gebeurt in verschillende fasen waarvan de eerste het oproepen van de instructie is.
Onder oproepen (FETCH) verstaan we het feit dat de instructie uit het geheugen in de centrale verwerkingseenheid gebracht wordt voor behandeling.
De eerste byte van een instructie bevat de opcode en wordt steeds in de instructiedecoder van de CPU gebracht.
Bij een multibyte instructie (operand(en) volgt (volgen) nog één of twee oproep bewerkingen (FETCH) waarbij de data in tijdelijke registers wordt opgeslagen. Daarna volgt de uitvoeringsfase (EXECUTE).
Het geheel bestaande uit het oproepen (FETCH) en het uitvoeren (EXECUTE) van een instructie noemen we een instructie cyclus.
Een instructiecyclus is samengesteld uit één of meer machinecycli (oproepen, uitvoeren). Elk van de machine cycli zelf wordt afgehandeld in verschillende periodes van de besturingsklok van de centrale verwerkingseenheid.
Zowel het aantal machinecycli als het aantal klokperiodes is verschillend van microprocessor tot microprocessor.
Specifiek voor de 8080 microprocessor van INTEL is bijvoorbeeld de onderverdeling van een instructiecyclus in 1 tot 5 machinecycli.

Ook de onderverdeling van elke machinecyclus in 3 tot 5 periodes van de besturingsklok, is specifiek voor de 8080 microprocessor.



Naargelang het type instructie kan de behandeling minimum 4 en maximum 18 periodes van de besturingsklok duren. Als we weten dat deze klok een frequentie van bijvoorbeeld 2MHz heeft, dus een periode van 0,5µs dan is de minimum duur van een instructie 2µs en de maximum duur 9µs. Om de totale duur van een programma te schatten, bekomt men meestal een aanvaardbaar resultaat als men een gemiddelde instructieduur van 4µs aanneemt bij een besturingsklok van 2MHz.
Bij het type 8080A1 is de maximum klokfrequentie 3MHz wat overeenkomt met een periode van 330ns. De minimum instructieduur is dus 1,3µ.
Bij het type 8085A is de maximum klokfrequentie 3,125 en bij het type 8085A2 5MHz.
Bij het type 8085 hebben sommige instructies in de eerste machinecyclus (eerste FETCH) 6 klokperiodes in plaats van 5 , bijvoorbeeld PCHL 6xT in plaats van 5xT.
Andere instructies hebben een machinecyclus minder zoals bijvoorbeeld bij voorwaardelijke sprongopdrachten, wanneer aan de voorwaarde niet voldaan is en niet moet gesprongen worden, bijvoorbeeld JZ (JUMP ON ZERO) 3xM (10xT) bij de sprongvoorwaarde en 2xM (7xT) indien de accumulator ≠ 0.
Bij de 8080 is dit steeds 3xM of 10T.
De minimum toegelaten klokfrequentie is 500KHz, wat overeenkomt met T = 2µs, dit wegens de dynamische technologie.
Bij de 6500 is de minimum klokfrequentie 20KHz met een maximum van 2MHz. De instructietijd varieert van 1µs tot 3µs.
Bij de 6800 is de maximum klokfrequentie 1MHz met een minimum van 100kHz. De instructietijd varieert van 1 tot 6µs. De instructies worden in 2 tot 12 kloktijden behandeld.
Bij de 68A00 is de maximum frequentie 1,5MHz en bij de 68B00 2MHz.
Bij de Z80 is de minimum klokfrequentie 5kHz met een maximum voor het type Z80 van 2,8MHz en voor het type Z80A van 4,5MHz.
De instructietijd varieert van 1 tot 5,75µs. De instructies kunnen tot 4 bytes lang zijn en vereisen tot 23 klokimpulsen. De lage waarde van de klok is te wijten aan het feit dat deze processor van het statisch type is. Enkele inwendige registers zijn echter dynamisch zodat een minimum klok vereist is.
Alhoewel de klokfrequentie van de 6300 en de 6500 lager is dan bij de andere types, kunnen instructies soms sneller verwerkt worden wegens het gebruik van de directe adresseringsmethode (*zero page*) waardoor steeds 1 byte minder nodig is.

HELP!

The commonest theme of your letters was "help!". I will start by giving a list of answers:

For Eddie Spavin:You disable the BREAK key by POKE #5F,#84. This unfortunately disables all the other keys. To avoid that a user's machine code routine to read the key-board is needed. So if writing in BASIC, put the following code at addresses #200-#218. POKE #5F,#84 at the beginning of the program and then when some input is required POKE #5F,#C4 and CALLM #200. The machine code waits for a key to be pressed (but not

```
0200 CD BB D6 CA 00 02 FE FF CA 00 02 CD 95 D6 32 20
0210 02 C9
```

BREAK) and then stores the ASCII code in location #200. As soon as the input is in, POKE #5F,#84 again. A typical subroutine which replaces INPUT A$ with GOSUB 1000 would be:

```
1000 PRINT "?";:A$=""
1010 POKE #5F,#C4:CALLM #200:POKE #5F,#84
1020 CHAR=PEEK(#220):IF CHAR=13 THEN RETURN
1030 A$=A$+CHR$(CHAR):GOTO 1010
```

The disassembly for the code is:

```
0200 CD BB D6   CALL :D6BB   Call GETC routine:returns key in A
0203 CA 00 02   JZ   :0200   If no key (i.e.0) call it again
0206 FE FF      CPI  :FF     Is break key pressed (returns 255)
0208 CA 00 02   JZ   :0200   If so, call GETC again
020B CD 95 D6   CALL :D695   Print character in A
020E 32 30 02   STA  :0220   Store character in #220
0211 C9         RET          Return to BASIC
```

The code is located in the envelope buffer, so if your program uses envelopes you'll have to stick it somewhere else.

Eddies other questions were: how do you change the baud rate on the RS232? Answer:POKE #FF05 with the following hex numbers.

```
#81-110bd    #82-150bd    #84-300bd    #88-1200bd
#90-2400bd   #A0-4800bd   #C0-9600bd (default)
```

Those numbers generate one stop bit. (the norm) If using two stop bits, subtract #80 from each number. Other baud rates would only be available by user-written software. How do you mix text and graphics? Answer:use the DAInamic FGT package to "draw" text. How do you get super-hi-res. Use the MODE 7 and MODE 8 programs in DAInamic 13.

For Dave Fortune:To read the paddle buttons you PEEK #FD00 and then IAND the result with #10 or #20. Here is a short demo.

```
10 A%=PEEK(#FD00) IAND #30      This program when run will
20 A%=A% SHR 4:PRINT A%         print a number every half-second
30 WAIT TIME 25:GOTO 10         1 if Paddle 2 button pressed,
                                2 if Paddle 1:3 if both pressed
```

Dave also asks members for a simple subroutine to print variables with two decimal places only thereafter. Obviously N=INT(N*100)/100 will do the rounding down (../100+0.5 to round up) but the problem is that the DAI converts 0.01 to 0.09 into scientific notation. Can anyone supply a routine to overcome this, please?

HELP!

Jerry Counsell is looking for (a)quick efficient methods of writing graphs to the screen, (b)a method of dumping text and graphics to an Epson printer, (c)the location of the "keyboard-readout", (d)methods of doing very high resolution arithmetic. (e)help with use of the DAI WP, and assembler/loader package.

Well,(a)the best answer would appear to be use of FGT package, in conjunction with a BASIC program. If FGT annotation is used, then (b)any DAI to Epson screen dump program will dump graphics with text, as the text "is" graphics when printed by FGT. (c)By the "keyboard readout", I assume Jeremy means where is the memory mapped location of the keyboard. This is at #FFF1 (Bits 0-6 only) but much more useful is the keyboard buffer which is at #2BA-#2BD, and stores the last four keys pressed. The address of the last key pressed is calculated as one less than the contents of #2BE/#2BF. High resolution arithmetic is easy for large integers, but I assume that Jeremy requires arithmetic on real numbers. I don't have a routine that does this:can any member oblige? Re: the final point, again I know how to use the assembler/loader and the Word Processor, and will oblige by answering specific questions. However on the more general question of "how do you use them?" perhaps someone else could help? Hopefully, I will be able to supply translated program notes at some stage.

Geoff Hawkins has the DAI Disc system and asks how to use the system to open files and print and input from disc. In brief, the answer is to POKE #131,3 and execute normal PRINT statements to write to disc, and POKE #135,3 and use the GETC routine to read from disc. However, if anyone has got some standard disc filing routines written, or indeed any programs written with the DOS in mind, perhaps they could let me have them or send them to Geoff via me (so that I foot the airmail postage cost). Geoff also wants to see some simple machine code so that he can get started and indeed many others of you have written to me in this vein. I am trying to include simple routines where I can, but the problem with a full-blown set of articles on the subject is that most of what I would say is covered much better in texts on the subject. I will repeat that 8080 programming is mainly the same on <u>any</u> 8080 machine, so the best thing for me to do is take up DAI-specific entry-points etc, assuming a general knowledge of 8080 assembler.

Eddie Ryzman, another professional user, asks how the text editor can be used to store and edit non-DAI BASIC programs getting round the DAI syntax checker. Input to the editor is by POKE #131,2 (the input will perhaps be from keyboard, RS232 disk etc.) and then the editor is entered by RST 5 DATA #2A (EF 2A). (BASIC "EDIT" won't work as this clears the edit buffer first). The program lines are removed from the editor by POKE #135,2, but the return area for programs must not be the command line interpreter. POKE #131,3 will therefore have to be used to pass the lines to disk, or a user-defined routine (see the "lower case" program elsewhere in this issue as an example of a user-defined output routine. Eddie is also looking for a machine code WP, that stores text as an addressed block rather than as an array, and is therefore more flexible. I did start work on this, but for one reason and another, never got round to finishing it. Can any member help (they are welcome to the source listings of the bits I have done)

HELP!

John Mitchell has had problems positioning the FGT characters on the screen: this is done by setting the X and Y variables in the BASIC control subroutine that accompanies every FGT program. You must use this subroutine to call FGT:don't just CALLM #300,A$. I have written a short demonstrator which you will find elsewhere in this newsletter. John has also had repeat trouble with some keys on his keyboard. Lack of key debounce is a known bug in BASIC V1.0 but from what John describes, he could have a sticky key. I had a sticky 'I' key at one point (excessive wear!?) and I found that removing the keycap and cleaning the exposed switch helped. John offers a hint for AMD 9511 owners. Some programs run too fast(!) with the maths chip installed, viz tunes etc, so to disable it, POKE #D4,0. Dave Blackadder asks for more info. on the 136 colours promised in the last issue. I am not able to give you anything further yet, but as they say, watch this space.

HINTS & TIPS

An abbrieviated issue this time round because of shortage of space (well, time actually!). Those of you who have sent me useful information will see their contributions published next time, which will hopefully be a couple of months away, and certainly not as long as last time. POKE #131,3 for non-disc users is pretty interesting. It jumps to #2DD-#2DF to direct its output, where a user defined jump can be made. I have stuck a bit of code in #200, the envelope buffer (very useful place for short routines if you aren't using envelopes) as follows:

```
0200 B7 FE 41 DA 0D 02 FE 5B D2 0D 02 C6 20 EF 03 C9
```

and then altered the #2DD-#2DF code to C3 00 02 and a user-defined routine exists. You put the code in with the S command. Now do that, while a BASIC program is in memory and type POKE #131,3 and LIST.

TAILPIECE

I have noticed that a firm called Ikon who sell DCR's for the BBC Micro under the name 'Hobbit' appear to be selling DCR tapes at about £20 a box, several pounds cheaper than any other supplier (unless of course, you know different!) Also a firm called Work Force in Luton sell Epson ribbons at £10 for four. These are without cartridges, so you have to refit them to the old cartridge yourself.

Next time I hope to include more on machine code, including the printer RHJ routines, a selection of translated documentation from published programs, and maybe a couple of printer and pssibly an RGB Monitor review. Keep the stuff coming! Professional users: please pass on your experiences too, especially in the field of hardware expansion/connections. Anyone know where to get 1v composite video out?

Dave Atherton

HIGH SPEED DATA LOADER

(HSDL)

Heeft U ook wel eens het probleem dat U juist een lang programma van band hebt ingelezen en dat de DAI zich dan "ophangt" en U weer opnieuw moet beginnen, of dat U eerst koffie kunt gaan drinken wanneer U een groot programma van zo'n 12K van een audio band moet inlezen en dan terug kerende moet ervaren dat er iets fout ging en men een tweede kop koffie kan gaan drinken of dat op zondagmorgen Uw zoontje U wakker komt maken omdat hij space invader wilt spelen maar het programma er niet in kan krijgen.

Nu Wij wel. Dan hebben Atari, Commodore 64 en Philips P2000 bezitters en nog vele anderen het een stuk eenvoudiger. ROM module met het gewenste programma erin, een of twee commando's en het loopt. Zoiets zou ook wel wat zijn voor de DAI dachten wij. Het idee was daar en na veel praten en knutselen met hard- en software hadden we een doosje in elkaar wat we High Speed Data Loader noemden.

Waarom high speed? Wel, we kunnen data inlezen met een snelheid van iets meer dan 10 kilobyte/sec. Dit betekent dat b.v. SPL niet langer duurt dan één seconde. Dus geen tijd meer voor koffie. De soft/hardware is zo gemaakt dat de HSDL eventueel zonder problemen met Uw DCR's gebruikt kan worden (disc weten we niet omdat we deze niet bezitten).

De HSDL is in staat om basic, MLP en gecombineerde programma's in te lezen. Al naar gelang van wat er in de EPROMS staat.

Wij zijn uitgegaan van een EPROM kaartje waarop 4 EPROM's gaan van de volgende types:

2716, 2732 en 2764 wat ons een maximale programma grootte geeft van resp. 8, 16 en 32 Kbyte.

Dit EPROM kaartje komt met een connector op de HSDL welke via een flatkabel is aangesloten op de DCE connector van de DAI.

Als het is aangesloten, is de rest eenvoudig n.l. zet de DAI aan en type onder basic RDL1 ... 4, return. Wanneer de cursor naar de volgende regel springt is het programma ingelezen. Voor een basic of kombinatie van basic met MLP is de "RUN" commando voldoende en het loopt.

Voor MLP is het iets moeilijker. Ga eerst naar UT, type Z3 en dan Gxxxx de locatie waar het programma start en dit loopt ook. Omdat alle programma's geen 8 resp. 16 K groot zijn hebben we het mogelijk gemaakt om meerdere programma's op een kaartje te zetten.

Iedere EPROM is bereikbaar met het commando RDL 1...4. 1 is de eerste EPROM en 4 is de 4de EPROM.

.../...

Wanneer een programma groter is dan wat in één of meerdere EPROMS past en men roept deze EPROM(S) aan dan krijgt men de foutmelding "NOT AVAILABLE". De ruimte in een EPROM welke niet gebruikt wordt door het aanwezige programma gaat helaas verloren. Alleen selectie van gehele EPROMS is mogelijk.

De high speed data loader is opgebouwd uit de volgende delen:-

1) EPROM kaartje op de X bus waarop een EPROM met besturingssoftware.

2) De data loader waarin de hardware zit, hierop zit een connector waarin het EPROM kaartje gestoken wordt.

3) Het betreffende EPROM kaartje met programma(s).

Er zijn momenteel twee prototypes gebouwd en wij zijn bezig de definitieve print te tekenen.

Bij voldoende belangstelling willen we een kleine productie beginnen en met Dainamic bespreken of het mogelijk is om programma's in PROM of EPROM in hun assortiment op te nemen, eventueel kompleet met printplaatje en connector.

Heeft men eenmaal de basisset dan is alleen het EPROM printje met programma's noodzakelijk. Het belangrijkste is wat gaat dat kosten. We hebben een voorlopige schatting gemaakt en komen op een bedrag dat rond de f 275,-- zal zijn, voor een basisset bestaande uit:-

EPROM met operating software;
high speed data loader;
EPROM kaartje met een demo programma in een EPROM.
Voor eventuele extra's zoals xbus print, flatcable met connectoren, EPROM kaartjes, etc... moeten we nog prijzen vaststellen.

Na een gesprek wat wij hadden met Dianamic hebben we nog een paar opmerkingen en een aantal vragen aan U.

1) Gebruikt U Uw DAI voor een bepaald doel b.v. SPL, DNA, Word Processing, VIDITEL of iets anders waarvoor maar 1 of 2 programma's gebruikt worden dan is een HSDL hier uitermate geschikt voor. Eenmaal in EPROM en geen "LOAD" problemen meer.

2) Dainamic levert op één cassette (normaal of DCR) meestal meerdere programma's voor een relatief lage prijs. Dit is met EPROMS niet mogelijk,dus zijn de kosten hiervan hoger dan van welke tape ook. Waarop dan onze vragen:
   a) Welke programma's zou U in EPROM willen zien?
   b) Wilt U de losse EPROMS of kompleet gemonteerd?
   c) Wat wilt U ervoor betalen? Buiten de vaste kosten van de EPROMS, aanmaak etc... komen hierbij de kosten van de programma's zelf (vaste kosten hardware ± f 80,-- 16KByte). Daarom geeft ons een reeële prijs indicatie. Kijkt U maar eens naar de programma's van de speel computers.

Mocht U interesse hebben voor een High Speed Data Loader laat het ons weten. Daarom vragen wij; stuur ons een briefkaartje met al Uw vragen en wensen, dan kunnen wij in één van de komende DAInamics daarop verder ingaan.

H. KOP,
Burg. Smeetsstraat 39,
6151 GM MUNSTERGELEEN NL.

H. RISON,
Luxemburgstr. 17,
6164 BS GELEEN NL.

```
1     REM
2     REM
3     REM [[[[[[[[        M A R Z I A N E T T O     ]]]]]]]]
4     REM
5     REM
6     REM ... A FIRST BASIC (SIC) STEP TO MOVE
7     REM A GREEN ALIEN ON THE SCREEN ...
8     REM
9     REM
10    REM Gianni Uliana
11    REM Via Zuccoli 40
12    REM 00137  ROMA  ITALY
15    COLORG 0 15 0 0
20    DIM A%(15.0)
30    FOR I=0.0 TO 15.0
40    READ A%(I)
50    NEXT
55    PRINT CHR$(12):PRINT :PRINT "MARZIANETTO":PRINT
56    PRINT " USE THE CURSOR KEYS TO MOVE A GREEN ALIEN "
58    WAIT TIME 200
60    MODE 3
70    B%=#BFEB
75    H=H+8.0:IF H=16.0 THEN H=0.0
77    SOUND 1 1 1 2 FREQ(50.0+200.0*H)
80    FOR I=0.0 TO 7.0
90    POKE B%-1,#50
100   POKE B%,A%(I+H)
110   B%=B%-46
120   NEXT
125   SOUND OFF
130   B%=B%+368
140   Z%=B%
150   C=GETC:IF C=0.0 THEN GOTO 150
160   IF C=19.0 AND X=19.0 THEN GOTO 150
170   IF C=18.0 AND X=0.0 THEN GOTO 150
180   IF C=17.0 AND Y=15.0 THEN GOTO 150
190   IF C=16.0 AND Y=0.0 THEN GOTO 150
200   IF C=19.0 THEN X=X+1.0:B%=B%-2
210   IF C=18 THEN X=X-1.0:B%=B%+2
220   IF C=17 THEN Y=Y+1.0:B%=B%-368
230   IF C=16 THEN Y=Y-1.0:B%=B%+368
240   FOR I=0.0 TO 7.0
250   POKE Z%,#0
260   Z%=Z%-46
270   NEXT
280   GOTO 75
500   DATA #00,#18,#3C,#5A,#7E,#42,#C3,#00
510   DATA #18,#18,#3C,#DB,#7E,#42,#42,#42
```

CATALOGUS NEDERLAND

CODE	TITEL	PRIJS (in guldens)
G1	GAMES COLLECTION 1	22,50
G2	GAMES COLLECTION 2	22,50
G3	GAMES COLLECTION 3	22,50
G4	GAMES COLLECTION 4	45,--
G5	GAMES COLLECTION 5	22,50
G6	GAMES COLLECTION 6	42,50
G7	GAMES COLLECTION 7	42,50
G8	GAMES COLLECTION 8	42,50
G9	GAMES COLLECTION 9	42,50
G10	GAMES COLLECTION 10	42,50
G11	GAMES COLLECTION 11	42,50
CTP	CENTIPEDE	35,--
DRI	DRIVER	35,--
SI	SPACE INVADER	45,--
SUI	SUPER INVADER	35,--
DAPA	DAIPANIC	45,--
DN	DAINIBBLE	45,--
ACR	ACROBATES	35,--
CH	SARGON CHESS	85,--
TK1	TOOLKIT 1	55,--
TK2	TOOLKIT 2	55,--
TK3	TOOLKIT 3	55,--
TK4	TOOLKIT 4	55,--
TK5	TOOLKIT 5	55,--
DNA	DNA ASSEMBLY PACK	62,50
SPL	SPL MACRO-ASSEMBLER	62,50
DTP	DAI TINY PASCAL	55,--
DTX	DAINATEXT	115,--
FGT	FAST GRAF TEXT	55,--
FGTA	FGT-APPLICATIONS	55,--
SFGT	SUPER FGT	55,--
GT	GRAPHIC TABLET	55,--
GH	GRAFISCHE HULP	27,50
ML	MAILING LIST	55,--
PE1	PRIMARY EDUCATION 1	55,--
SE1	SECUNDARY EDUCATION 1	55,--
SE2	SECUNDARY EDUCATION 2	55,--
WP	WORDPROCESSOR	55,--
EGT	ENGLISH-GERMAN TRAINER	55,--
JR	MICRO'S-ONDERWIJS	57,50
TT1	TAAL 1	42,50
FB	FAMILIEBUDGET	27,50
W3	WISKUNDE 3	42,50
F1	FYSICA 1	42,50
DD	DAI DEMO + BASIC TUTOR	27,50
T80	TAPE 80-81	47,50
N10	NEWSLETTER 10	27,50
N11	NEWSLETTER 11-12	37,50
N13	NEWSLETTER 13-14-15	37,50
M1	MUSIC COLLECTION 1	17,--
M2	MUSIC COLLECTION 2	17,--
M3	MUSIC COLLECTION 3	17,--

	HARDWARE & PUBLICATIONS	
PCS	DAIpc SCHEMATICS	47,50
BOD	BEST of DAInamic	27,50
SNG	SUPER NOISE GENERATOR	85,--
NC	NEW CHARACTER GENERATOR	55,--
DCE	DCE-INTERFACE-CARDS	140,--
N8	NEWSLETTERS 8-13 (1982)	27,50
	LIDMAATSCHAP	50,--

Op DCR zijn de programma's 8 gulden duurder. De banden kunt U bestellen door het

bedrag over te maken op onderstaande rekening. Bij de medelingen moet U de code van de door U gewenste band(en) vermelden en of U de programma's op AUDIO of DCR

wilt hebben. Als U geld over maakt voor het lidmaatschap wilt U dan vermelden: LIDMAATSCHAP en het jaartal?

GIRO: 4083817
t.n.v.: J.F. van Dunne'
HOFLAAN 70
3062 JJ ROTTERDAM
TEL: (010) 144802

UNIEKE aanbieding voor DAI-gebruikers !!!

Een raspaardje onder de processorkaarten voor definitieve toepassingen:

UNIDATA

Om voor al uw ideeen m.b.t. automatisering van reselingen en/of besturingen een oplossing te vinden is er nu een universele processorkaart ontwikkeld.

Deze eurokaart heeft standaard de volgende eigenschappen:

- 8085 CPU (6.144Mhz)
- interne interrupt controller
- 1/4 K RAM + 2 sockets voor uitbreiding naar 1 1/4 K RAM
- 2 EPROM-sockets voor 2716 en/of 2732 (dus maximaal 8 K ROM)
- programmeerbare timer
- universele I/O aansluitbus (DAI-DCE compatible gemaakt)-- 8155
- 20 mA current-loop (voor direkte aansluiting teletype TTY e.d.)
- RS 232 (voor aansluiting modem of bijv. DAI computer)
- printbanen voor eigen hardware uitbreiding
- e'e'n voedingsspanning (+5V bij current-loop)
- uitgebreide nederlandstalige handleiding

EXTRA leverbaar:

TTY-monitor (1 K in 2716 en SDK-85 compatible) inclusief gegevens belangrijke entry points.

De prijs ? ? ?

Vanwege de aanmaak van proffesionele eurokaarten kan deze processorkaart bij VOLDOENDE belangstelling geleverd worden voor:

UNIDATA-1.....fl 189,-
TTY-monitor...fl 45,- prijzen incl BTW

T.Groeneveld
J.P.Wiersmawei 7
8915 HT leeuwarden
DAI-gebruiker

Indien u belangstelling heeft en deze kaart wilt bestellen, dan dient u schriftelijk kontakt op te nemen met:

2. DE DCE-MICRO-COMPUTER

Het doel van deze cursus is, om ervaring op te doen met het programmeren van een micro-computer.
De DCE-micro-computer is daarbij een hulpmiddel. Op deze computer kunt u n.l. uw programma's invoeren, testen en debuggen.
Het is niet de bedoeling, dat u in deze cursus de werking van elk onderdeel van de DCE-micro-computer leert kennen.
Wilt u er alles van weten, dan verwijzen we naar de literatuur.
In de komende lesen bespreken we alleen datgene wat u van de DCE-micro-computer moet kennen om uw programma's te kunnen invoeren, testen en debuggen.

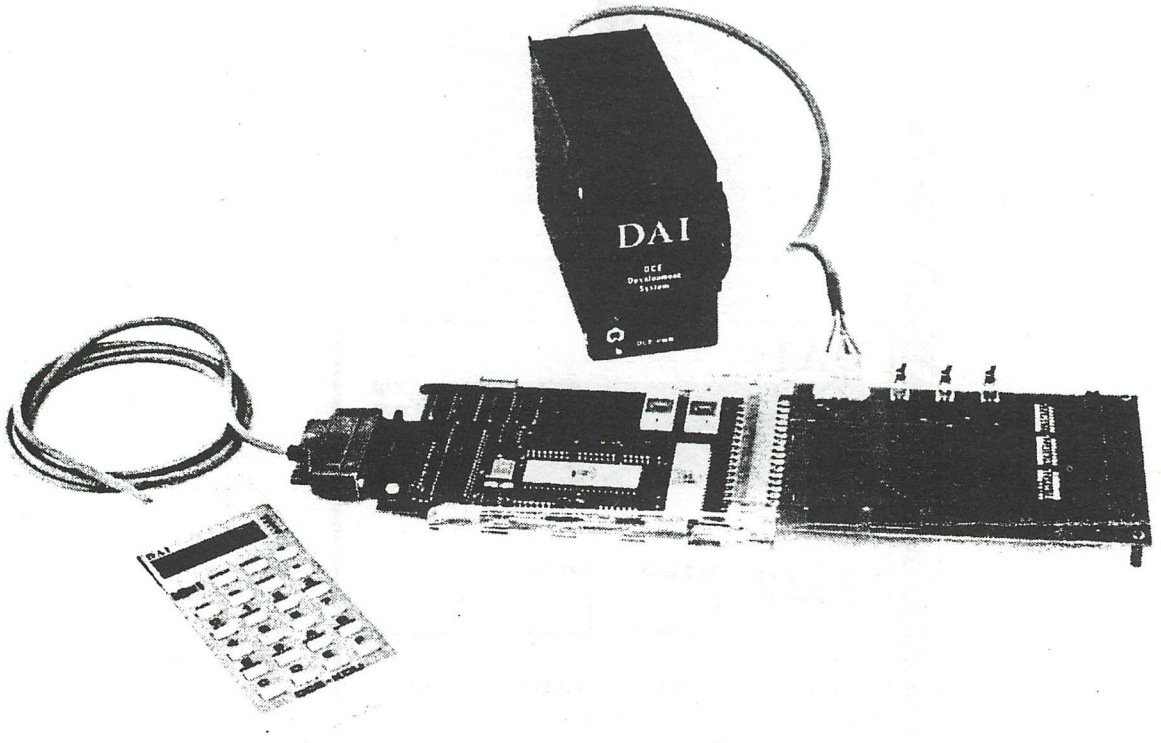


fig.1

De micro-computer uit fig. 1 bestaat, ruimtelijk gezien, uit de volgende delen.

1. De DCE-print (DCE = Digital Control Element).
   Op deze print bevinden zich de 8080, de I/O-modules en het centrale geheugen.
2. De extender (verlengstuk) waarop zich een aantal schakelaars en LED's bevinden om input- en outputsignalen op te wekken of na te bootsen.
3. Het toetsenbord en de display, met behulp waarvan de communicatie tussen de programmeur en de micro-computer verloopt.
4. De voeding.

In deze les gaan we uitgebreid in op de DCE en de extender. Het toetsenbord behandelen we in de volgende les. (Op de voeding gaan we in deze cursus niet in.).

3. BLOKSCHEMA VAN DE DCE

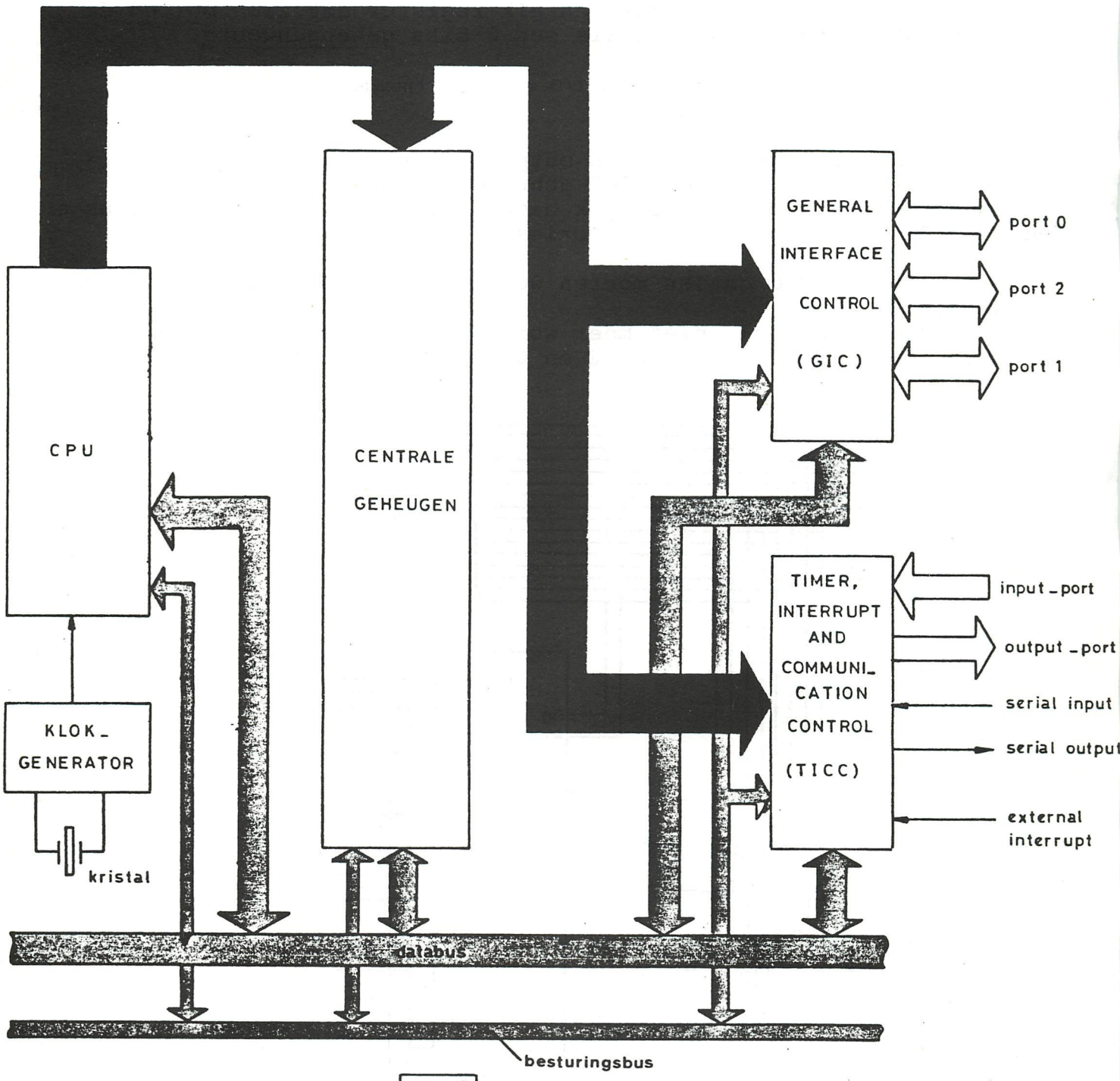


fig.2

In fig. 2 is het blokschema van de DCE weergegeven.
Rondom de micro-processor bevinden zich de volgende eenheden.
1. Het centrale geheugen, bestaande uit RAM's en EPROM's.
2. De klokgenerator.
3. De General Interface Controller (GIC).
4. De Timer, Interrupt and Communications Controller (TICC).

In de volgende paragrafen bespreken we de GIC en de TICC en geven we een overzicht van de bezetting van de geheugencapaciteit.
Wat betreft de werking van de CPU, de klokgenerator en het centrale geheugen, verwijzen we naar de betreffende lessen uit de basiscursus.

4. MEMORY MAPPED I/O

De DCE werkt volgens het systeem van memory mapped I/O. Dit houdt in, datelke in- en outputpoort en elk register dat zich buiten de CPU bevindt, wordt behandeld als een 8-bits geheugenwoord.

Vraag 1: Bij memory mapped I/O zijn poortnummers vervangen door .....

De poortnummers, waarmee in- en outputpoorten werden geadresseerd, worden vervangen door "normale" geheugenadressen.
IN- en OUT-instructies komen ook niet meer voor. Ze worden vervangen door LOAD, STORE en MOVE-instructies.

Hardware-technische gezien moeten wel enkele kunstgrepen worden uitgehaald.
Een I/O-module zal nl. niet meer worden aangestuurd door IN- en OUT-signalen, maar door READ, WRITE en CHIP-SELECT-signalen.

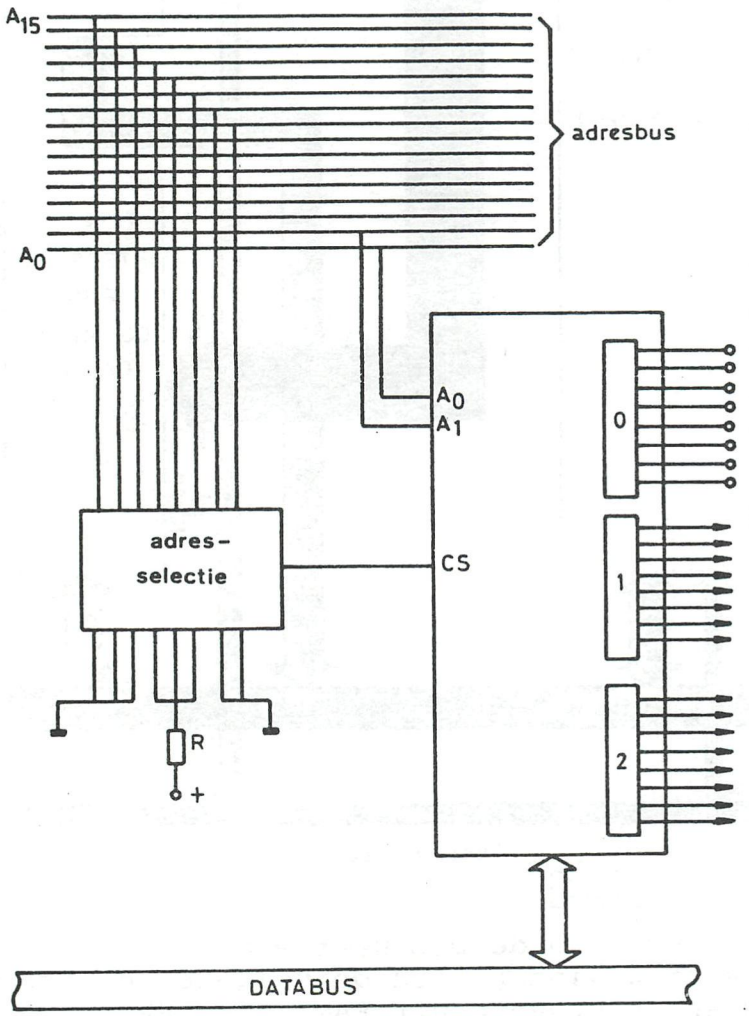


fig.3

Antw.1: geheugenadressen.

In fig. 3 is weergegeven hoe we een I/O-module als geheugenbouwsteen kunnen opnemen.
De I/O-module in fig. 3 bestaat uit 2 output-poorten en 1 input-poort. Voor de selectie van deze poorten zijn 3 ingangen aanwezig: 2 adres-ingangen en 1 chip-select (CS)-ingang. Wanneer we nl. de chip (de I/O-module) hebben geselecteerd, moeten we alleen nog 1 van de poorten binnen die chip adresseren. Dit kan m.b.v. 2 bits. Hiermee zijn immers 4 combinaties mogelijk, nl. $00_2 = 0$; $01_2 = 1$; $10_2 = 2$ en $11_2 = 3$.
De chip selecteren we, door b.v. de 8 hoogste adresbits m.b.v. een comparator te vergelijken met een vooraf ingestelde waarde.

Vraag 2: In fig. 3 worden de 8 hoogste adresbits vergeleken met $00111100_2/00011100_2$.

In fig. 3 is dit de waarde $00011100_2 = 1C_{16}$.

Vraag 3: De adressen van de 3 poorten in fig. 3 zijn:
$\ldots\ldots_{16}$, $\ldots\ldots_{16}$ en $\ldots\ldots_{16}$.

De adressen van de 3 poorten in fig. 3 zijn dus $1C00_{16}$, $1C01_{16}$ en $1C02_{16}$.

We zien nu meteen een groot nadeel opdoemen van memory mapped I/O.
De I/O-module in fig. 3 neemt nl. een veel groter adresbereik in beslag dan de 3 adressen die voor de poorten nodig zijn.
Immers, bij alle adressen die beginnen met 1C.... wordt de I/O-module aangesproken. We mogen dus stellen, dat de adressen 1C04 t/m $1CFF_{16}$ niet meer voor andere doeleinden gebruikt mogen worden.
In de meeste gevallen is dit niet zo'n groot bezwaar, omdat de volle 64k-geheugenruimte toch niet nodig is.
Is het benutten van alle geheugencapaciteit wel nodig, dan moeten we werken met grotere comparators. In fig. 3 zullen we dan de adresbits A_7 t/m A_2 ook moeten vergelijken met een vooraf ingestelde waarde.

Een groot voordeel van memory mapped I/O is, dat bij een input- en output-operatie de bestemming resp. de bron niet meer persé de accumulator behoeft te zijn.
Vullen we b.v. registerpaar H,L met $1C01_{16}$, dan kunnen we de inhoud van register D direct naar output-poort 1 brengen met de instructie MOV M,D.

20-01 Antw.2: 00011100_2. Antw.3: $1C00_{16}$; $1C01_{16}$ en $1C02_{16}$.

5. DE GIC

De DCE beschikt over 2 I/O-modules, de GIC en de TICC.
De GIC (General Interface Controller) wordt gevormd door de 8255 van Intel.
Deze IC beschikt over 3 poorten, welke onafhankelijk van elkaar als input-poort of als output-poort kunnen fungeren.

M.b.v. een 8-bits woord, dat we naar de z.g. mode-controller van de GIC zenden, bepalen we in welke mode (input of output) we elk van de 3 poorten schakelen.
M.a.w. de mode van de 3 poorten is programmeerbaar.
(Een betere aanduiding voor de GIC zou dus eigenlijk zijn: Programmable Peripheral Interface (PPI)).

In fig. 4 is een blokschema van de GIC weergegeven.
De mode-controller van de GIC heeft, net als de 3 poorten, een adres.
De adressen zijn in fig. 4 tussen haakjes weergegeven.

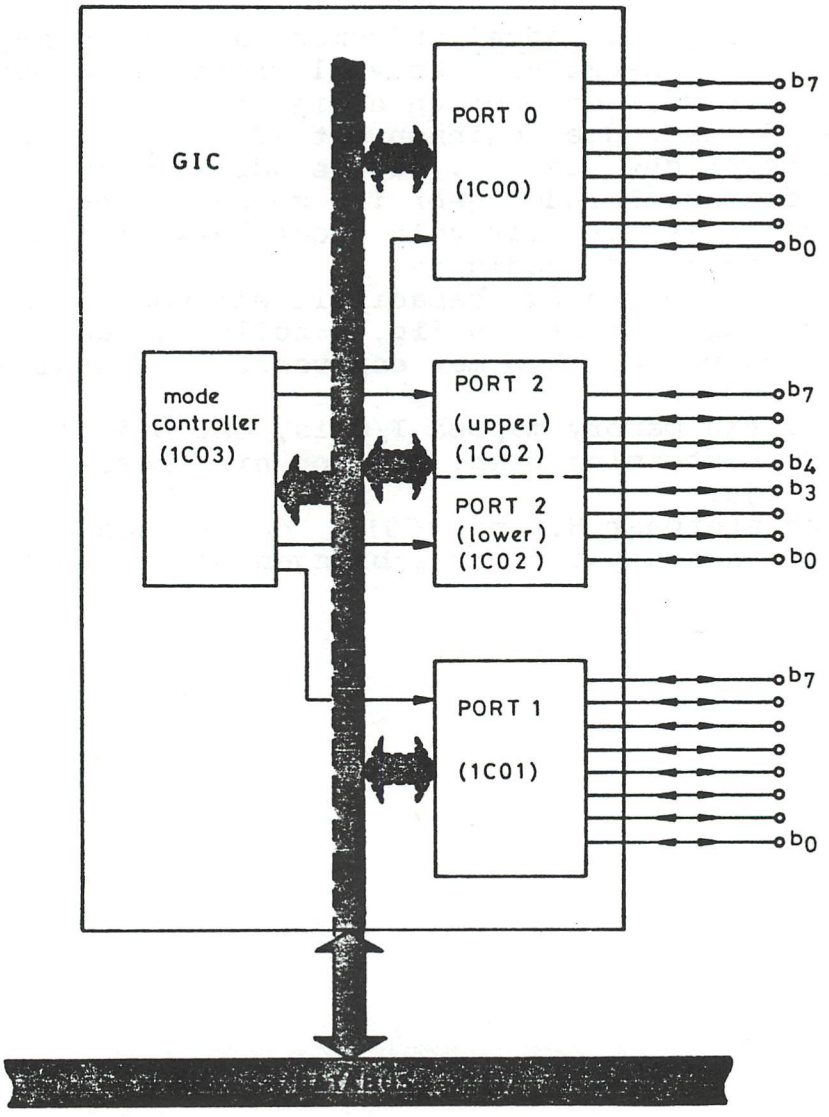


fig.4

Vraag 4: Het adres van de mode controller is ....._16.

We kunnen de mode controller dus beschouwen als een geheugenwoord met adres $1C03_{16}$.
De mode controller decodeert het 8-bits codewoord dat hij ontvangt, en schakelt dan de GIC-poorten in de gewenste mode.
We merken nog op dat port 2, wat betreft de mode, in 2 delen kan worden gesplitst, nl. port 2 upper, waartoe b_4 t/m b_7 behoren en port 2 lower, gevormd door b_0 t/m b_3.

In fig. 5 is weergegeven hoe het codewoord, waarmee we de modes van de 3 poorten aangeven, is opgebouwd.

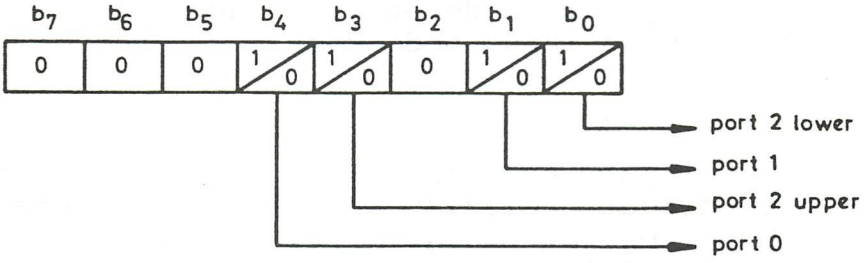


fig.5

We zien, dat de bits b_7, b_6, b_5 en b_2 niet worden gebruikt. Deze bits zijn altijd 0.
Met b_0 geven we de mode van port 2 (lower) aan. Is $b_0 = 0$, dan werkt port 2 (lower) als output-poort. Is $b_0 = 1$, dan werkt port 2 (lower) als input-poort.
Op dezelfde manier geven we met b_1, b_3 en b_4 de modes van resp. port 1, port 2 (upper) en port 0 aan.
Een 0 komt dus overeen met output; een 1 komt overeen met input.

Voorbeeld 1:

Gevraagd:
Welk codewoord moeten we naar adres $1C03_{16}$ sturen om de poorten van de GIC in de volgende modes te schakelen ?
port 0 = input-poort
port 2 (upper en lower) = input-poort
port 1 = output-poort

Oplossing:
b_0 moet 1 worden, omdat port 2 (lower) als input-poort moet werken.
b_1 moet 0 worden, omdat port 1 als output-poort moet werken.
b_3 en b_4 moeten beide ook 1 worden, omdat port 2 (upper) en port 0 als input-poort moeten werken.
Het code-woord wordt dus $00011001_2 = 19_{16}$.
M.b.v. de 2 nevenstaande instructies kunnen we dit woord naar de mode controller zenden.

```
MVI  A,19
STA  1C03
```


20-01 Antw.4: $1C03_{16}$.

Voorbeeld 2:

Gegeven:
De inhoud van de accumulator is 13_{16}.

Gevraagd:
In welke modes komen de poorten na uitvoering van de instructie STA 1C03 ?

Oplossing:
Met de instructie STA 1C03 sturen we de inhoud van de accumulator naar adres $1C03_{16}$, d.w.z. naar de mode controller.
Deze ontvangt dan het woord $13_{16} = 00010011_2$. b_0 en b_1 zijn 1, zodat port 2 (lower) en port 1 in de input mode komen.
$b_3 = 0$, zodat port 2 (upper) in de output mode komt.
$b_4 = 1$, zodat port 0 in de input mode komt.

Opmerking:

Behalve de "normale" input- en output-mode is m.b.v. de 8255 ook handshake I/O en bi-directionele I/O mogelijk. Deze modes worden bepaald door de bits b7, b6, b5 en b3.
In deze cursus maken we van deze mogelijkheden echter geen gebruik, zodat deze bits in onze gevallen altijd 0 zijn.

SAMENVATTING 1

1. De DCE (Digital Controller Element) bestaat uit:
   a. een 8080 micro-processor.
   b. RAM's en EPROM's.
   c. 2 I/O-modules, de GIC en de TICC.

2. De DCE werkt volgens memory mapped I/O.
   Dit houdt in, dat elk register buiten de CPU als geheugenwoord wordt beschouwd.

3. GIC betekent General Interface Controller.
   De GIC beschikt over 3 I/O-poorten die m.b.v. een codewoord onafhankelijk als input- of outputpoort geschakeld kunnen worden.

MICRO FAST GRAPHICS

```
PAGE 01     MICRO FAST GRAFFICS  *** DAINAMIC ***

002
003
004                    ************************************************
005                    * THIS UTILITY PROVIDES A FAST DRAWING FACILITY  *
006                    * APPLICABLE FOR EACH MODE EXEPT MODE 0.         *
007                    * IT IS TAILERED TO BE USED TOGETHER WITH BASIC. *
008                    * THIS MEANS THAT PARAMETERS ARE PASSED VIA BA-  *
009                    * SICS SYMBOL TABLE. YOU HAVE TO SPECIFY 4 VARIA- *
010                    * BLES I.E. - X (ABSIS OF THE GRAFFICS SCREEN)   *
011                    *           - Y (ORDINATE OF THE GRAFFICS SCREEN) *
012                    *           - C (DESIRED COLOR )                *
013                    *           - E (ENTRY OF A TABLE WITH THE NE-  *
014                    *           CESSARY INFO TO DRAW A PICTURE)     *
015                    *               SEE FUTHER.                     *
016                    *  X,Y SPECIFY THE STARTPOSITON OF THE PICTURE ON *
017                    *  THE SCREEN                                   *
018                    * BEFORE CALLING THE PROCEDURE ONE MUST PASS THE  *
019                    * NECESSARY INFORMATION REFFERING TO THE MENTIONED*
020                    * VARIABLES.                                     *
021                    * THIS CAN BE DONE IN THIS WAY (I = INTEGER)     *
022                    * I=VARPTR(X)+2:POKE#2F0,I MOD 256:POKE#2F1,I/256 *
023                    * I=VARPTR(Y)+3:POKE#2F2,I MOD 256:POKE#2F3,I/256 *
024                    * I=VARPTR(C)+3:POKE#2F4,I MOD 256:POKE#2F5,I/256 *
025                    * I=VARPTR(E)+2:POKE#2F6,I MOD 256:POKE#2F6,I/256 *
026                    * PICT=#300, TO ACTIVATE PROCEDURE BY CALLM PICT .*
027                    * THE USER MUST CREATE HIS SPECIFIC TABEL CONTAI- *
028                    * NING ALL NECESSARY INFORMATION TO DRAW THE RE-  *
029                    * QUIRED PICTURE (S).                            *
030                    * THE TABLE CONSISTS OF ONE OR MORE ENTRIES, ONE  *
031                    * FOR EACH PICTURE. EACH ENTRY CONSISTS OF SEVERAL*
032                    * ELEMENTS(ONE FOR EACH DOT,DRAW OF FILL FUNCTION)*
033                    * EACH ELEMENT CONSISTS OF 5 BYTES: M,x1,y1,x2,y2 *
034                    * THE LAST ELEMENT HOWEVER CONSISTS OF ONLY ONE   *
035                    * BYTE (#FF) DENOTING THE END OF THE ENTRY.       *
036                    * - M = OPERATOR: #1E/DOT, 21/DRAW, 24/FILL       *
037                    * - x1,x2 ABCIS IN A 256x256 MATRIXS (IF MODE 6)  *
038                    * - y1,y2 ORDINATES IN THE SAME MATRIX            *
039                    *   x AND y ARE OFFSETS OF X AND Y               *
040                    *   NOTE THAT x2,y2 ARE DON'T CARE IN CASE OF THE *
041                    *   DOT OPERATION.                               *
042                    ************************************************rt
043
044
045                    *
046                    * SYMBOL TABLE #2F0...2F8
047                    *
048                            ORG    :2F0
049                                                  **** USER DEFINABLE
050 02F0 0000          X       DBL    0             +VARPTR(X-COORD)+2
051 02F2 0000          Y       DBL    0             +VARPTR(Y-COORD)+3
052 02F4 0000          COLOR   DBL    0             +VARPTR(COLOR)+3
053 02F6 0000          ENTRY   DBL    0             +VARPTR(ENTRY)+2
054                                                  **** SYSTEM TEMPORARY STORAG
```

```
055 02F8 00          KLEUR   DATA  0
056                  *
057                  * START PROGRAM
058                  *
059                          ORG   :300
060 0300 E5          START   PUSH  H          PUSH ALL
061 0301 D5                  PUSH  D
062 0302 C5                  PUSH  B
063 0303 F5                  PUSH  PSW
064                  *
065                  * GET ENTRY
066                  *
067 0304 2AF602              LHLD  ENTRY      LOAD ADDRESS OF
068 0307 56                  MOV   D,M        TABLE ENTRY (2 BYTES )
069 0308 23                  INX   H
070 0309 5E                  MOV   E,M
071                  *
072                  * GET COLOR
073                  *
074 030A 2AF402              LHLD  COLOR      GET COLOR
075 030D 7E                  MOV   A,M
076 030E 32F802              STA   KLEUR      STORE TEMPORARY
077 0311 EB                  XCHG
078                  *
079                  * PICK UP OPERATOR (FILL,DRAW,DOT)
080                  *
081 0312 7E          MFGTO   MOV   A,M        FIRST ELEMENT OF ENTRY
082 0313 23                  INX   H          INDICATES FILL, DOT, DRAW
083 0314 FEFF                CPI   :FF
084 0316 CA5903              JZ    END
085 0319 324D03              STA   LABEL+1    FILL IN OPERATOR
086                  *
087                  * PICK UP X1,Y1,X2,Y2
088                  *
089 031C 0603                MVI   B,3        REPEAT 2 TIMES FOLLOWING
090                                           LOOP
091 031E 05          MFGT1   DCR   B
092 031F CA4103              JZ    MFGT2
093                  *
094                  * CALCULATE X1 = X + x1 AND X2 = X + x2
095                  *
096 0322 5E                  MOV   E,M        LOAD x
097 0323 23                  INX   H
098 0324 1600                MVI   D,0
099 0326 E5                  PUSH  H
100 0327 D5                  PUSH  D
101 0328 2AF002              LHLD  X          LOAD CONTENTS OF X
102 032B 56                  MOV   D,M        MS BYTE
103 032C 23                  INX   H
104 032D 5E                  MOV   E,M        LS BYTE
105 032E E1                  POP   H
106 032F 19                  DAD   D          CALCULATE X + x
107 0330 EB                  XCHG
```

```
108 0331 E1                  POP   H          CURRENT TABEL POINTER
109 0332 D5                  PUSH  D          PUSH X1 OR X2 ON STACK
110                  *
111                  * CALCULATE Y1 = y1 + Y OR Y2 = y2 + Y
112                  *
113 0333 4E                  MOV   C,M        GET y1 OR y2
114 0334 23                  INX   H
115 0335 E5                  PUSH  H
116 0336 2AF202              LHLD  Y          LOAD CONTENTS OF Y
117 0339 7E                  MOV   A,M        (ONLY ONE BYTE)
118 033A 81                  ADD   C          CALCULATE Y+y
119 033B 4F                  MOV   C,A
120 033C E1                  POP   H
121 033D C5                  PUSH  B          PUSH Y1 OR Y2 ON STACK
122 033E C31E03              JMP   MFGT1
123                  *
124                  * PERFORM OPERATION
125                  *
126 0341 E3          MFGT2   XTHL             FETCH Y2
127 0342 45                  MOV   B,L
128 0343 E1                  POP   H
129 0344 D1                  POP   D          FETCH X2
130 0345 E3                  XTHL             FETCH Y1
131 0346 4D                  MOV   C,L
132 0347 E1                  POP   H
133 0348 E3                  XTHL             FETCH X1
134 0349 3AF802              LDA   KLEUR      FETCH COLOR
135 034C EF          LABEL   RST   5          ACTIVATE DRAW, FILL OR DOT
136 034D 21                  DATA  :21        1E:DOT;21:DRAW;24:FILL
137 034E E1                  POP   H          FETCH TABELPOINTER
138 034F DA5503              JC    ERROR      CHECK FOR ERROR
139 0352 C31203              JMP   MFGTO
140 0355 3E45        ERROR   MVI   A,'E'      IN CASE AN ERROR IS DE-
141 0357 EF                  RST   5          TECTED, A 'E' IS PRINTED
142 0358 03                  DATA  3
143 0359 F1          END     POP   PSW        POP ALL
144 035A C1                  POP   B
145 035B D1                  POP   D
146 035C E1                  POP   H
147 035D C9                  RET
148 035E                     END
```

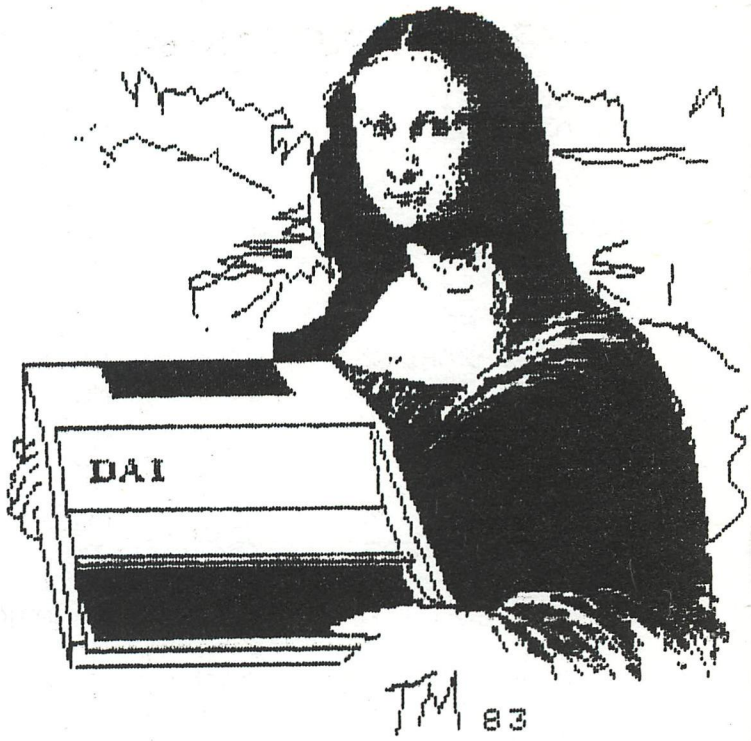
```
****************************
* S Y M B O L   T A B L E *
****************************

COLOR  02F4   END    0359   ENTRY  02F6   ERROR  0355
KLEUR  02F8   LABEL  034C   MFGTO  0312   MFGT1  031E
MFGT2  0341   START  0300   X      02F0   Y      02F2
```

```
0300 E5 D5 C5 F5 2A F6 02 56 23 5E 2A F4 02 7E 32 F8
0310 02 EB 7E 23 FE FF CA 59 03 32 4D 03 06 03 05 CA
0320 41 03 5E 23 16 00 E5 D5 2A F0 02 56 23 5E E1 19
0330 EB E1 D5 4E 23 E5 2A F2 02 7E 81 4F E1 C5 C3 1E
0340 03 E3 45 E1 D1 E3 4D E1 E3 3A F8 02 EF 21 E1 DA
0350 55 03 C3 12 03 3E 45 EF 03 F1 C1 D1 E1 C9
```

Videes graphics

TOMISLAV MIKULIC
DAKICEV TRG 3/5
YU - 41000 ZAGREB
YUGOSLAVIA
tel. (041) 535 490



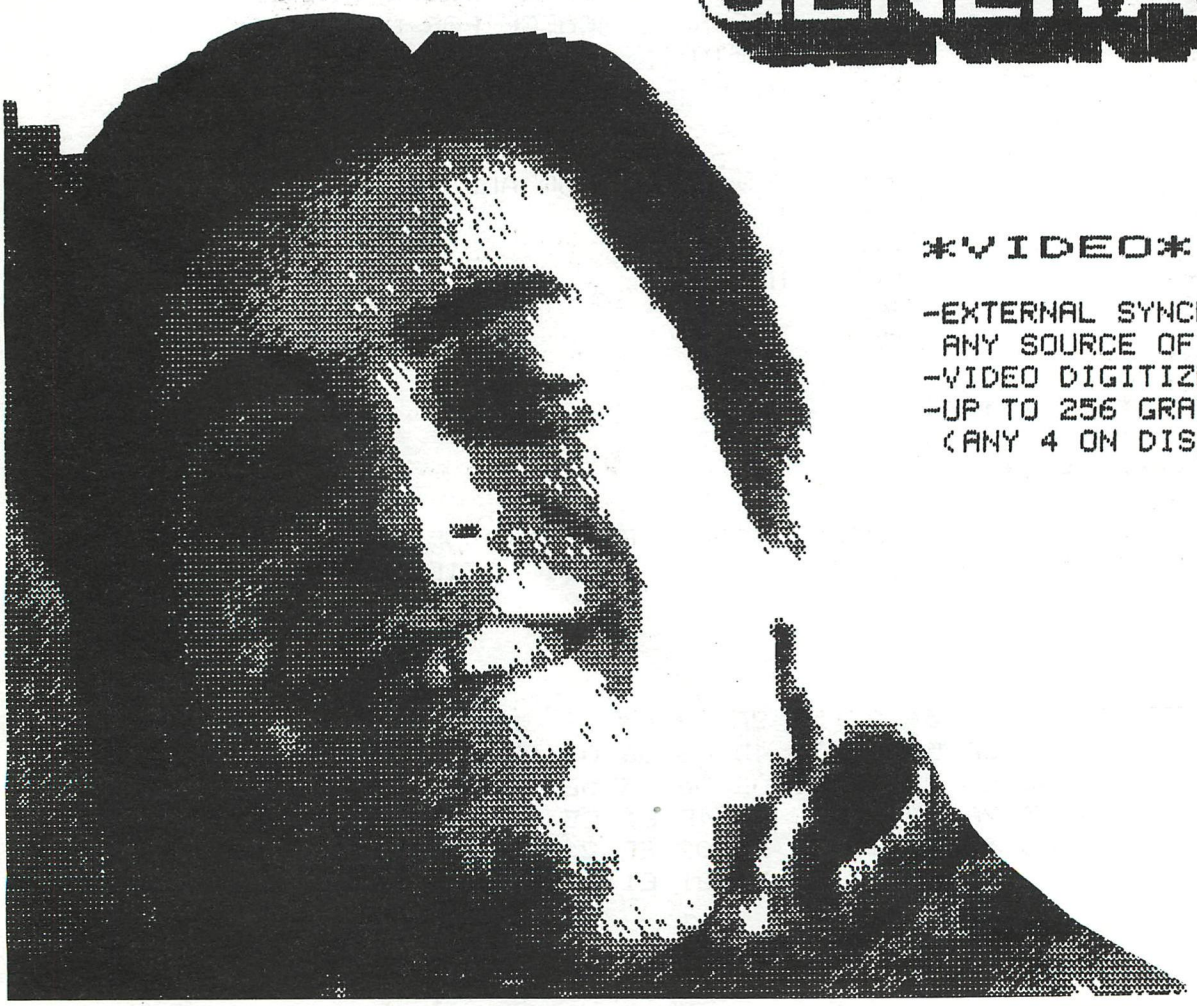
GRAPHICS TABLET

-SCREEN MENU OVERLAY
-HAND DRAWING
-ZOOM, SHRINK
-SPECIAL EFFECTS
-FILL AREA
-GEOMETRIC SHAPES
-USER BRUSHES, etc.

CHARACTER GENERATOR

-FULL GRAPHICS OVERLAY
-ANY STYLE FONTS
-HELVETICA IN VARIOUS SIZES
-USER DEFINED SYMBOLS
-VARIOUS SHADOWS AND EDGES

CHARACTER GENERATOR



VIDEO

-EXTERNAL SYNCHRONIZATION WITH ANY SOURCE OF COMPOSITE VIDEO
-VIDEO DIGITIZER
-UP TO 256 GRAY LEVELS (ANY 4 ON DISPLAY)

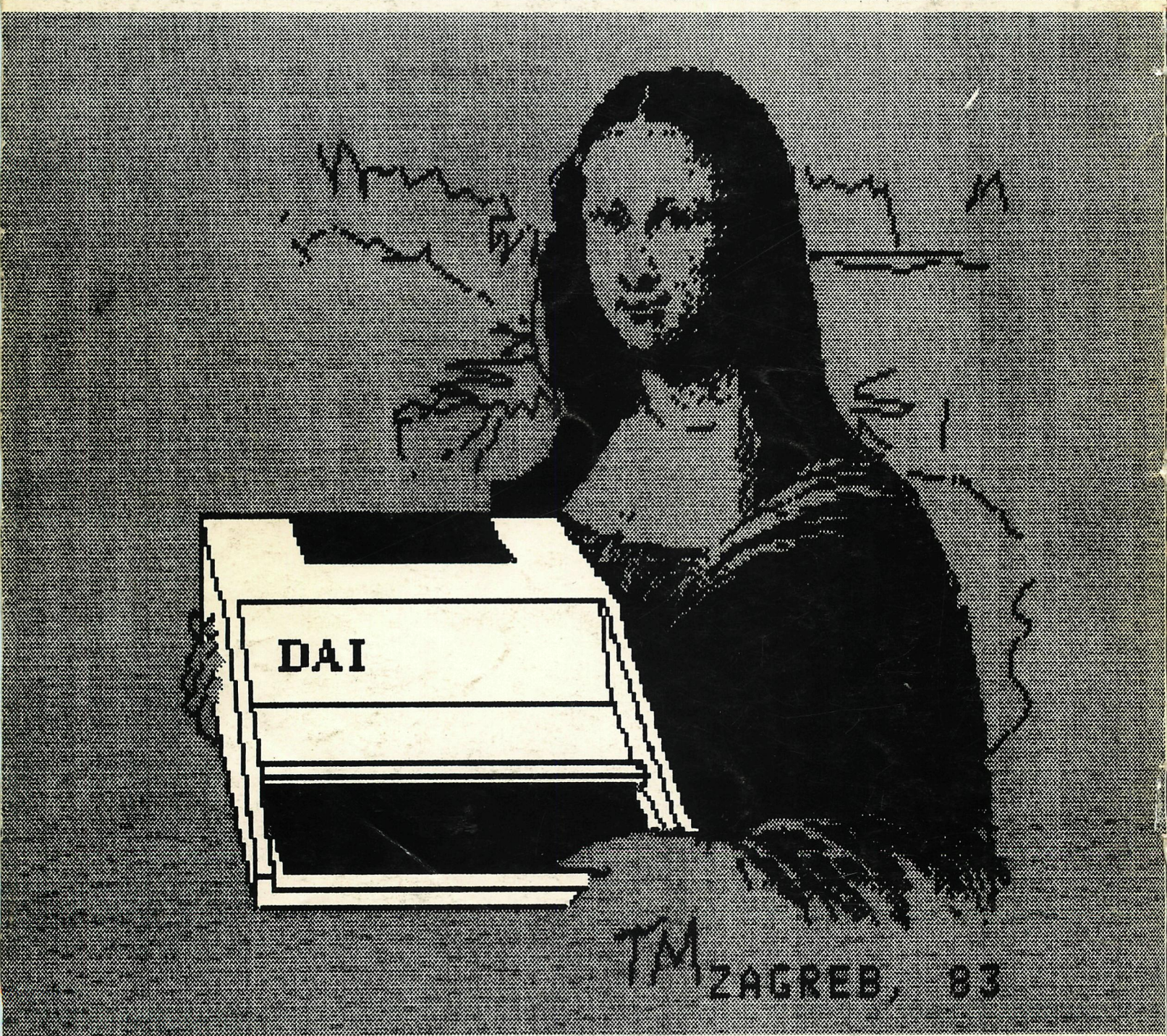
IF YOU ARE SERIOUS ABOUT DAI VIDEO GRAPHICS CONTACT ME ON ABOVE ADDRESS

8080 instructionset

INSTRUCTION	FUNCTION	HEX
MOVE GROUP	reg	A B C D E H L M
MOV A, reg	(A)←(reg)	7F 78 79 7A 7B 7C 7D 7E
MOV B, reg	(B)←(reg)	47 40 41 42 43 44 45 46
MOV C, reg	(C)←(reg)	4F 48 49 4A 4B 4C 4D 4E
MOV D, reg	(D)←(reg)	57 50 51 52 53 54 55 56
MOV E, reg	(E)←(reg)	5F 58 59 5A 5B 5C 5D 5E
MOV H, reg	(H)←(reg)	67 60 61 62 63 64 65 66
MOV L, reg	(L)←(reg)	6F 68 69 6A 6B 6C 6D 6E
MOV M, reg	(M)←(reg)	77 70 71 72 73 74 75 --
ACCUMULATOR GROUP		
ADD reg	(A)←(A)+(reg) *	87 80 81 82 83 84 85 86
ADC reg	(A)←(A)+(reg)+(CY) *	8F 88 89 8A 8B 8C 8D 8E
SUB reg	(A)←(A)-(reg) *	97 90 91 92 93 94 95 96
SBB reg	(A)←(A)-(reg)-(CY) *	9F 98 98 9A 9B 9C 9D 9E
ANA reg	(A)←(A)∧(reg) *	A7 A0 A1 A2 A3 A4 A5 A6
XRA reg	(A)←(A)∀(reg) *	AF A8 A9 AA AB AC AD AE
ORA reg	(A)←(A)∨(reg) *	B7 B0 B1 B2 B3 B4 B5 B6
CMP reg	(A)-(reg) *	BF B8 B9 BA BB BC BD BE
INCREMENT/DECREMENT REGISTER		
INR reg	(reg)←(reg)+1 **	3C 04 0C 14 1C 24 2C 34
DCR reg	(reg)←(reg)-1 **	3D 05 0D 15 1D 25 2D 35
REGISTER PAIR GROUP	rp	B D H SP PSW
INX rp	(rp)←(rp)+1	03 13 23 33 --
DCX rp	(rp)←(rp)-1	0B 1B 2B 3B --
LDAX rp	(A)←((rp))	0A 1A -- -- --
STAX rp	((rp))←(A)	02 12 -- -- --
DAD rp	(H,L)←(H,L)+(rp) ***	09 19 29 39 --
PUSH rp	((SP)-1)←(rh),((SP)-2)←(rl), (SP)←(SP)-2	C5 D5 E5 -- F5
POP rp	(rl)←((SP)),(rh)←((SP)+1), (SP)←(SP)+2	C1 D1 E1 -- F1 *
DIRECT ADDRESS GROUP		
LDA addr	(A)←(addr)	3A al ah
STA addr	(addr)←(A)	32 al ah
LHLD addr	(L)←(addr),(H)←(addr+1)	2A al ah
SHLD addr	(addr)←(L),(addr+1)←(H)	22 al ah
IMMEDIATE GROUP		
MVI A, data	(A)←data	3E dd
MVI B, data	(B)←data	06 dd
MVI C, data	(C)←data	0E dd
MVI D, data	(D)←data	16 dd
MVI E, data	(E)←data	1E dd
MVI H, data	(H)←data	26 dd
MVI L, data	(L)←data	2E dd
MVI M, data	(M)←data	36 dd
ADI data	(A)←(A)+data *	C6 dd
ACI data	(A)←(A)+data+(CY) *	CE dd
SUI data	(A)←(A)-data *	D6 dd
SBI data	(A)←(A)-data-(CY) *	DE dd
ANI data	(A)←(A)∧data *	E6 dd
XRI data	(A)←(A)∀data *	EE dd
ORI data	(A)←(A)∨data *	F6 dd
CPI data	(A)-data *	FE dd
LXI B, addr	(B)←ah,(C)←al	01 al ah
LXI D, addr	(D)←ah,(E)←al	11 al ah
LXI H, addr	(H)←ah,(L)←al	21 al ah
LXI SP,addr	(SP_h)←ah,(SP_l)←al	31 al ah

INSTRUCTION	FUNCTION	HEX
JUMP GROUP		
JMP addr	(PC)←addr	C3 al ah
JNZ addr	If Z=0, (PC)←addr	C2 al ah
JZ addr	If Z=1, (PC)←addr	CA al ah
JNC addr	If CY=0,(PC)←addr	D2 al ah
JC addr	If CY=1,(PC)←addr	DA al ah
JPO addr	If P=0, (PC)←addr	E2 al ah
JPE addr	If P=1, (PC)←addr	EA al ah
JP addr	If S=0, (PC)←addr	F2 al ah
JM addr	If S=1, (PC)←addr	FA al ah
PCHL	(PC_h)←(H), (PC_l)←(L)	E9
CALL GROUP		
CALL addr	(TOS)←(PC),(PC)←addr	CD al ah
CNZ addr	If Z=0, (TOS)←(PC),(PC)←addr	C4 al ah
CZ addr	If Z=1, (TOS)←(PC),(PC)←addr	CC al ah
CNC addr	If CY=0,(TOS)←(PC),(PC)←addr	D4 al ah
CC addr	If CY=1,(TOS)←(PC),(PC)←addr	DC al ah
CPO addr	If P=0, (TOS)←(PC),(PC)←addr	E4 al ah
CPE addr	If P=1, (TOS)←(PC),(PC)←addr	EC al ah
CP addr	If S=0, (TOS)←(PC),(PC)←addr	F4 al ah
CM addr	If S=1, (TOS)←(PC),(PC)←addr	FC al ah
	N.B. (TOS)←(PC) designates the following:- ((SP)-1)←(PC_h),((SP)-2)←(PC_l),(SP)←(SP)-2	
RETURN GROUP		
RET	(PC)←(TOS)	C9
RNZ	If Z=0, (PC)←(TOS)	C0
RZ	If Z=1, (PC)←(TOS)	C8
RNC	If CY=0,(PC)←(TOS)	D0
RC	If CY=1,(PC)←(TOS)	D8
RPO	If P=0, (PC)←(TOS)	E0
RPE	If P=1, (PC)←(TOS)	E8
RP	If S=0, (PC)←(TOS)	F0
RM	If S=1, (PC)←(TOS)	F8
	N.B. (PC)←(TOS) designates the following:- (PC_l)←((SP)),(PC_h)←((SP)+1),(SP)←(SP)+2	
RESTART GROUP		
RST 0	(TOS)←(PC),(PC)←00_{16}	C7
RST 1	(TOS)←(PC),(PC)←08_{16}	CF
RST 2	(TOS)←(PC),(PC)←10_{16}	D7
RST 3	(TOS)←(PC),(PC)←18_{16}	DF
RST 4	(TOS)←(PC),(PC)←20_{16}	E7
RST 5	(TOS)←(PC),(PC)←28_{16}	EF
RST 6	(TOS)←(PC),(PC)←30_{16}	F7
RST 7	(TOS)←(PC),(PC)←38_{16}	FF
ROTATE/CONTROL/SPECIAL GROUP		
RLC	(A_{n+1})←(A_n),(A_0)←(A_7),(CY)←(A_7) ***	07
RRC	(A_n)←(A_{n+1}),(A_7)←(A_0),(CY)←(A_0) ***	0F
RAL	(A_{n+1})←(A_n),(A_0)←(CY),(CY)←(A_7) ***	17
RAR	(A_n)←(A_{n+1}),(A_7)←(CY),(CY)←(A_0) ***	1F
NOP	No operation	00
HLT	Processor stopped until interrupt or reset	76
DI	Interrupts disabled	F3
EI	Interrupts enabled after next instruction	FB
XTHL	(L)↔((SP)),(H)↔((SP)+1)	E3
SPHL	(SP_h)←(H),(SP_l)←(L)	F9
XCHG	(H)↔(D),(L)↔(E)	EB
DAA	Decimal adjust accumulator *	27
CMA	(A)←$\overline{(A)}$	2F
STC	(CY)←1 ***	37
CMC	(CY)←$\overline{(CY)}$ ***	3F
OUT port) IN port)	Not used in DCE Systems	D3 port DB port

DAI graphics



SEND YOUR DRAWINGS TO DAINAMIC EDITOR