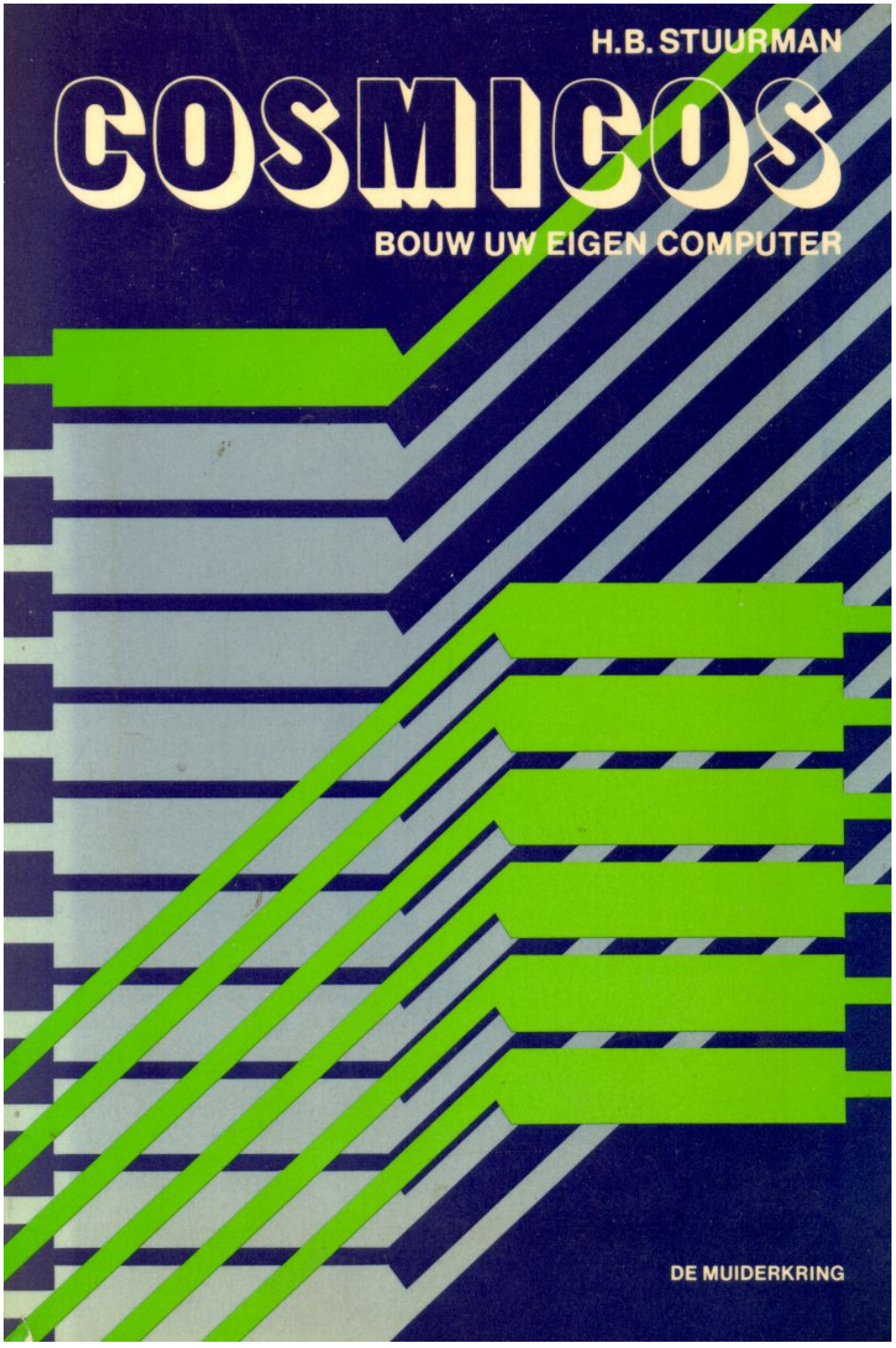

H.B. STUURMAN
COSMICOS
BOUW UW EIGEN COMPUTER
DE MUIDERKRING

H.B. STUURMAN

COSMICOS

BOUW UW EIGEN COMPUTER

DE MUIDERKRING B.V. - BUSSUM
UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

DEEL 2: INTRODUCTIE IN HET PROGRAMMEREN

DEEL 3: UITBREIDING VAN COSMICOS

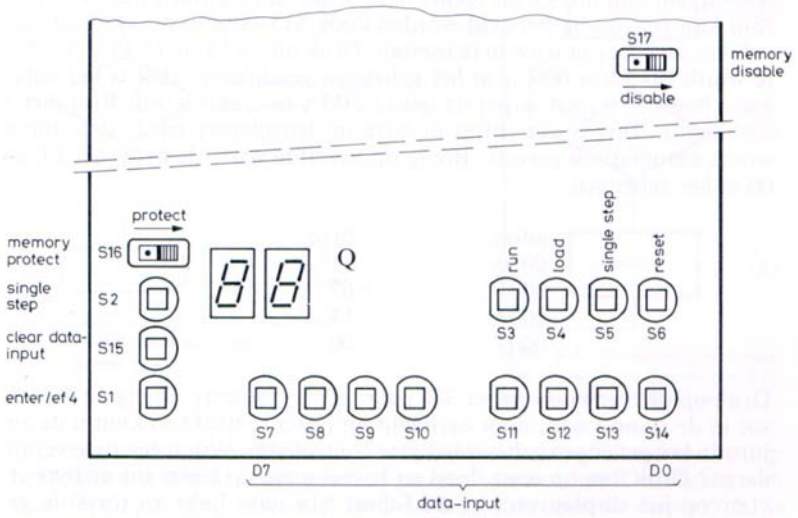
DEEL 2
introductie in het programmeren

HOOFDSTUK 6

Introductie in het programmeren 1

Bediening van Cosmicos

In afb. 6-1 ziet u het bedieningsgedeelte van de Cosmicos. Het bevat 15 drukknoppen en 2 schuifschakelaars. Eén schuifschakelaar (S17) bevindt zich een beetje achteraf. Dit is de ''memory disable'' schakelaar. Naar rechts geschoven komen de geheugen IC's in de stand - by toestand en behouden hun informatie onafhankelijk van de netspanning. De voeding wordt ontleend aan de 2 penlight batterijen. Voor normaal bedrijf moet de schakelaar naar links staan.
Schuifschakelaar S16 is de ''memory protect'' schakelaar. In de rechterstand is het writesignaal van processor naar geheugen IC's verbroken. De informatie hierin kan dan uit sluitend gelezen worden.
Het is raadzaam de ''gevaarlijke'' stand van de schakelaars S16 en S17 (naar links) te kenmerken met een stip rode verf.
De 4 drukknoppen S3 t/m S6 rechtsboven dienen voor het instellen van de 4 functie-toestanden. De bijbehorende LED brandt dan. Drukknoppen S7 t/m S14 zijn de 8 data-ingangen. Een iets grotere tussenruimte tussen S10 en S11 verdeelt ze in 2 groepjes van 4 bits overeenkomstig de hexadecimale code. De twee 7-segment display's vormen de uitgang van



Afb. 6-1: Plaats en functie van de 17 knopjes van Cosmicos.

Cosmicos. Als de processor d.m.v. S5 in de single step toestand is gebracht, veroorzaakt iedere indrukking van S2 één machinecyclus. Met S15 kan de data-input gereset worden. Alle LED's zijn dan uit en de aangeboden byte is 00_{16}. Door de met de Hexcode overeenkomstige data-input knoppen even in te drukken zet men de gewenste byte in de input-Latch.
S1 tenslotte heeft een dubbelfunctie. In de stand load wordt na indrukken een DMA-in cyclus opgewekt die de input byte in de door Register 0 aangewezen geheugen plaats schrijft. Tijdens run is S1 verbonden met externe vlaglijn 4 (EF4). Tussen S1 en de EF4-lijn is een flipflop geschakeld die bij het inschakelen van de voedingsspanning een willekeurige toestand kan innemen. Dit kan tot gevolg hebben dat in de toestand ''Load'' soms 2× op Enter moet worden gedrukt voordat de eerste byte op het display verschijnt. Als u er echter een gewoonte van maakt na het aanzetten van de computer eerst een keer op Enter te drukken heeft u nergens last van!
Als u Cosmicos terdege heeft gecontroleerd dan is het nu tijd de spanning aan te sluiten. Op het display verschijnen 2 getallen en waarschijnlijk gaan er enige data-LED's branden. Wat moét branden is de LED behorende bij reset. Schuif nu S16 en S17 naar links om het geheugen in te schakelen. Breng de processor in de load toestand d.m.v. de gelijknamige knop. De bijbehorende LED moet gaan branden en de reset LED dooft. Druk op de knop ''clear data-input''. Breng nu met de 8 data-input knoppen een byte in de ingangslatch b.v. $E0_{16}$. Gebruik hiervoor de hexadecimale tabel. Het is het gemakkelijkst de inputknoppen consequent van links naar rechts in te drukken (beginnen met D7). Een fout kan eenvoudig hersteld worden door S15 even te drukken en vervolgens de byte opnieuw in te toesten. Druk nu op ''Enter'' (S1). De byte wordt op adres 0000_{16} in het geheugen geschreven. (Dit is het adres waar Register 0 naar wijst; na iedere DMA-in cyclus wordt R(0) met 1 verhoogd). Tevens verschijnt de byte op het display en de data-input wordt automatisch gereset. Breng op dezelfde wijze de bytes 67, FF en 00 in het geheugen.

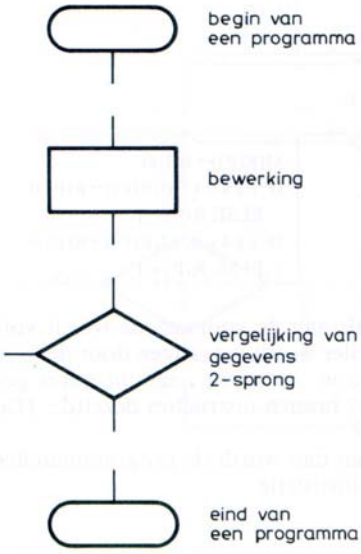
adres	byte
0000	E0
0001	67
0002	FF
0003	00

Druk op de resetknop en zet S16 naar rechts (protect). Breng de processor in de stand Load; door herhaald op Enter te drukken kunt u de zojuist in het geheugen gebrachte bytes controleren. Wilt u een byte veranderen? Druk dan op reset, load en zoveel maal op Enter tot de byte ervoor op het display verschijnt. Schuif S16 naar links en toets de gewenste byte in. Vergeet niet S16 weer naar rechts te schuiven als u vol-

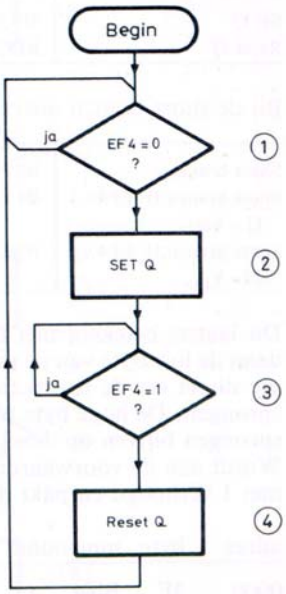
gende geheugenplaats wilt controleren. Klopt het? Druk dan op reset, schuif S16 naar links en druk op run. Verschijnt er FF op het display?

Stroomdiagrammen

Een stroomdiagram is een verzameling blokken verbonden door pijlen. Ieder blok bevat de beschrijving van één of meer door de processor te verrichten handelingen. Zo'n handeling kan b.v. zijn het optellen van 2 getallen; het naar rechts schuiven van een getal of het controleren of aan een bepaalde voorwaarde is voldaan. Door voor verschillende soorten handelingen de blokken verschillend van vorm te maken kan d.m.v. deze blokken al grofweg worden aangegeven hoe de afloop vam een programma zich dient te voltrekken. Hoewel er meer symbolen bestaan, zijn voor het ogenblik alleen de in afb. 6-2 getoonde van belang. Laten we eens proberen m.b.v. van de blokken een eenvoudig programma op te stellen.



Afb. 6-2: De belangrijkste symbolen waaruit een stroomdiagram wordt opgebouwd.



Afb. 6-3: Stroomdiagram van een programma om de Q-LED te bedienen met S1 (Enter).

Bij de aansluitingen in Hfdst. 3 is vermeld dat de Cosmac een Q-flipflop bevat die met 2 instructies geset of gereset kan worden. De uitgang van deze flipflop is verbonden met de Q-LED (rechts naast de display). S1 is verbonden met externe vlaglijn 4 (EF4). Deze vlag kan d.m.v. sprong-

instructies afgetast worden. Hoe maken we nu een programma dat de Q-LED laat branden als de knop wordt ingedrukt en uitdoet als de knop wordt losgelaten.
In afb. 6-3 ziet u een mogelijke oplossing. Direct na het begin tast de processor EF4 af. Als deze niet ingedrukt is wordt terug-gesprongen naar begin, en dat blijft doorgaan met een snelheid van zo'n 100.000 maal per seconde tot de knop wordt ingedrukt. Nu wordt Q geset en onmiddellijk hierop volgt de vraag of de knop is gelaten. Zolang dat niet het geval is blijft de processor hier kringetjes (loops) draaien. Als de knop wordt losgelaten wordt Q gereset en vervolgens teruggesprongen naar het begin. In bijlage A vindt u het complete instructie repertoire van de Cosmac microprocessor.
Bij de controle instructions vindt u

instruction	mnemonic	opcode	operation
Set Q	SEQ	7B	1→Q
Reset Q	REQ	7A	0→Q

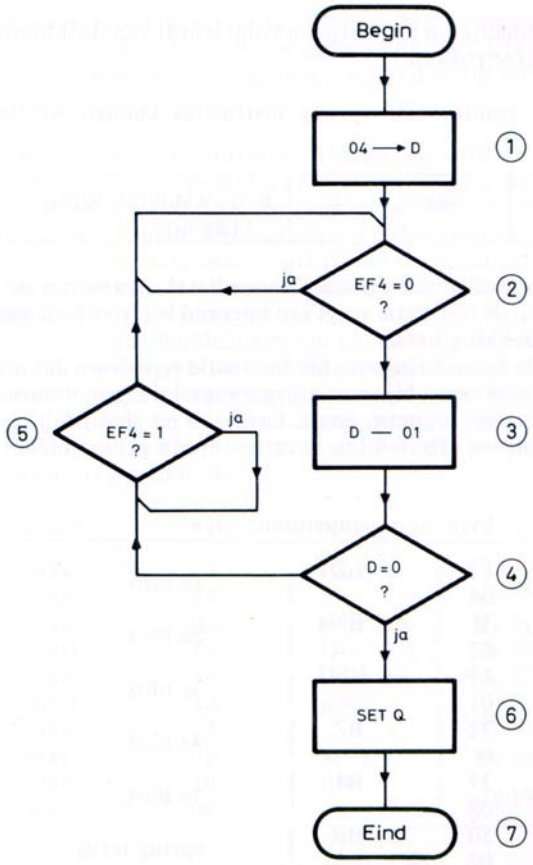
Bij de short branch instruction vindt u

Short branch	BR	30	M(R(P))→R(P)0
Short branch IF EF4 = 1 (1 = Vss)	B4	37	IF EF4 = 1, M(R(P))→R(P).0 ELSE R(P) + 1
Short branch IF EF4 = 0 ($0 = V_{DD}$	BN4	3F	IF EF4 = 0, M(R(P))→R(P).0 ELSE R(P) + 1

Dit laatste betekent niet anders dat als aan de voorwaarde wordt voldaan de low byte van de programmateller wordt vervangen door de byte die direct op de instructie volgt. M.a.w. er wordt naar dit adres gesprongen. De hoge byte blijft bij short branch instructies dezelfde. (De sprongen blijven op dezelfde pagina)
Wordt aan de voorwaarde niet voldaan dan wordt de programmateller met 1 verhoogd en pakt de volgende instructie

adres	byte	mnemonic	
0000	3F	BN4	} 1e blok
0001	00		}
0002	7B	SEQ	} 2e blok
0003	37	B4	} 3e blok
0004	03		}
0005	7A	REQ	} 4e blok
0006	30	BR	} spring naar 00
0007	00		}

Begrijpt u hoe het werkt. Breng het op de eerder beschreven manier in het gebeuren. Vergeet niet run in te drukken!
Werkt het? Wat zeggen uw huisgenoten ervan? Oh, ze weten wel een eenvoudiger manier om een lampje aan en uit te schakelen. Stel nu dat we het D-register vullen met een bepaald getal, b.v. 4. Door EF4 in te drukken trekken we er 1 af. Als de inhoud van het D-register 0 is geworden moet de Q-LED gaan branden; dus na 4× drukken. In afb. 6-4 ziet u een mogelijke oplossing. Het dataregister kunnen we laden met instructie



Afb. 6-4: Na 4× drukken op S1 gaat de Q-LED branden.

Load immediate	LDI	F8	M(R(P))→D; R(P) + 1

De byte die onmiddellijk op de instructie volgt komt in het data- (D-) register.

We kunnen een getal van het D-register aftrekken met instructies.

Subtract memory immediate	SMI	FF	D—M(R(P))→ DF,D; R(P) + 1

De byte die onmiddellijk op de instructie volgt wordt van de inhoud van het data register afgetrokken

Met de volgende conditionele sprong instructies kunnen we onderzoeken of D = 0

Short branch IF D = 0	BZ	32	IF D = 0,M(R(P))→R(P).0 ELSE R(P) + 1

Misschien is u opgevallen dat bij alle ''immediate'' instructies de byte die onmiddellijk op de instructie volgt een operand is (grootheid waarop een handeling betrekking heeft).
Verder ziet u bij de bestudering van het instructie repertoire dat alle instructies overzichtelijk soort bij soort zijn gerangschikt (lange sprong instructies, rekenkundige, logische, enz.). Laten we nu eens proberen het stroomdiagram volgens afb. 6-4 om te zetten in een programma.

adres	byte	mnemonic	
0000	F8	LDI	1e blok
0001	04		
0002	3F	BN4	2e blok
0003	02		
0004	FF	SMI	3e blok
0005	01		
0006	32	BZ	4e blok
0007	0C		
0008	37	B4	5e blok
0009	08		
000A	30	BR	spring terug
000B	02		
000C	7B	SEQ	6e blok
000D	00	IDL	7e blok

De laatste instructie IDLE heeft tot gevolg dat de processor stopt in afwachting van een interrupt- of DMA signaal. Dit komt echter niet. In het algemeen is het verstandiger de processor te stoppen d.m.v. een onconditionele sprong

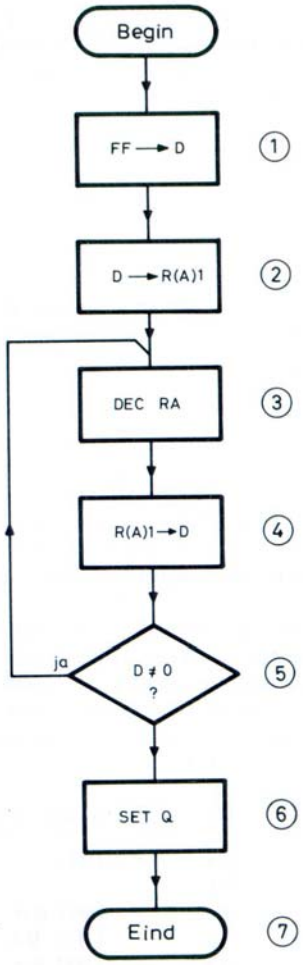
adres	byte	mnemonic
000D	30	BR
000E	0D	

We hoeven natuurlijk niet te stoppen. Het is eenvoudig om het programma uit te breiden met een gedeelte dat de LED doet uitgaan nadat EF 4 één of meer keer is ingedrukt, waarna wordt teruggesprongen naar het begin.

De Cosmac bevat 16 universele 16-bits registers. Deze register kunnen gebruikt worden als pointers naar een geheugenplaats; als programmatellers of om gewoon twee bytes in te bewaren. Elk register kan met 1 verhoogd (Increment; INC) of verlaagd (Decrement; DEC) worden. Het is een zeer eenvoudig deze mogelijkheid te benutten om een vertragingstijd op te wekken. Het stroomdiagram volgens afb. 6-5 geeft een mogelijkheid.
Merk op dat alle handelingen via het dataregister gaan. Als telregister kan ieder register gebruikt worden (behalve natuurlijk de programmateller (R0). We nemen register A; dit wordt met 1 verlaagd door in de instructie 2N de letter N te vervangen door A; op-code 2A. We stoppen een getal in de high byte van RA met instructie put high reg N; in ons geval wordt de op-code BA

adres	byte	mnemonic	
0000	F8	LDI	1e blok
0001	FF		
0002	BA	PHI RA	2e blok
0003	2A	DEC RA	3e blok
0004	9A	GHI RA	4e blok
0005	3A	BNZ	5e blok
0006	03		
0007	7B	SEQ	6e blok
0008	30	BR	7e blok
0009	08		

Waarom stoppen we het getal in R(A)1. Wel, als we het in R(A) 0 stoppen is de vertragingstijd zo kort dat de LED bijna meteen brandt na het indrukken van Run.



Afb. 6-5:

Opwekken van een vertragingstijd.

Hoelang is de vertragingstijd? Het decoderen en het uitvoeren van een instructie duurt 2 machinecyclussen. Iedere machinecyclus duurt 8 clockpulsen. De clockfrequentie is 1,75 MHZ; een machinecyclus duurt dus bij benadering:

$$\frac{1}{1{,}75} \times 8 = 4{,}57\ \mu s$$

De tijdlus bestaat uit 3 blokken (3, 4 en 5) dus 6 machinecyclussen. Aannemend dat de R(A) 0 gevuld was met 00 wordt de lus 65280_{10} maal. doorlopen ($FF00_{16}$). De vertragingstijd is

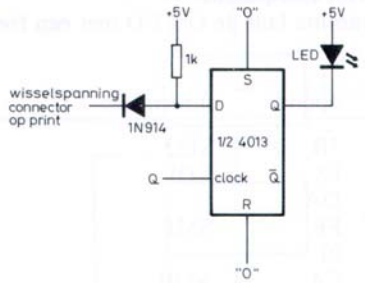
$65280 \times 6 \times 4{,}57\mu s = 1{,}79s$

Afregeling van de clockfrequentie
Onderstaand programma laat de Q-LED met een frequentie van 50 HZ knipperen

adres	opcode	mnemonic
0000	7B	SEQ
1	F8	LDI
2	DA	
3	FF	SMI
4	01	
5	C4	NOP
6	C4	NOP
7	3A	BNZ
8	03	
9	E2	
A	E2	
B	7A	REQ
C	F8	LDI
D	DA	
E	FF	SMI
F	01	
10	C4	NOP
1	C4	NOP
2	3A	BNZ
3	0E	
4	E2	
5	30	BR
6	00	

Instructie C4 is een no operation instruction. Deze heeft geen invloed op het programma behalve dat hij 3 machinecyclussen duurt.
Instructie E2 is een instructie die één van de 16 universele register tot het X-register maakt. In dit programma is dat niet van belang en wordt de E2 instructie alleen gebruikt omdat hij 2 machinecyclussen duurt. C4 en E2 zijn gebruikt om de juiste tijdsduur te verkrijgen.
Als u een 2-kanaalsoscilloscoop heeft kunt u één kanaal m.b.v. het busprintje op de Q-uitgang aansluiten; het andere kanaal sluit u aan op een pool van de trafo-connector. Hierop staat een enkelfazige gelijkspanning van 50Hz; de netfrequentie. Regel nu de spoel af tot de

beelden t.o.z. van elkaar stilstaan. Als u een enkel straals oscilloscoop heeft kunt u één signaal aansluiten op de X-ingang en regelt u de spoel af tot een lissajou-figuur wordt verkregen. Heeft u geen oscilloscoop, gebruik dan de schakeling volgens afb. 6-6. De D-flipflop is een fazedetector en de spoel wordt afgeregeld tot de knipperfrequentie van de LED zo laag mogelijk is.



Afb. 6-6: Bij gebrek aan een oscilloscoop kan een D-fipflop worden gebruikt om de clockfrequentie af te regelen.

Input en Output-instructies.
Een gedeelte van het instructie repertoire heeft betrekking op input- en output instructies (input-output byte transfer).
De data-input van de Cosmicos wordt geadresseerd met: INPUT 7 (6F).
Het display met OUTPUT 7 (67).

Input 7	INP 7	6F	BUS → M(R(X)); BUS → D; N LINES = 7
Output 7	OUT 7	67	M(R(X)) → BUS; R(X) + 1; N LINES = 7

De handeling die instructie 6F teweegbrengt is:
de byte die op de bus staat wordt in de geheugenplaats geschreven waar register X haar wijst; de byte komt ook in het data-register. Output-instructie 67 heeft als resultaat:
de byte op de geheugenplaats waar register X haar wijst wordt op de bus gezet waarna register X met 1 wordt verhoogd.
Laten we een programma maken dat de byte op de data-input laat zien op het display na indrukken van EF4.
Voor de geheugenplaats kiezen we 00FF en register 2 en maken we X-register.

adres	byte	mnemonic	
0000	F8	LDI	vul R2 met 00FF
1	00		
2	B2	PHI R2	
3	F8	LDI	
4	FF		
5	A2	PLO R2	
6	E2	SEX R2	R2 = RX
7	3F	BN4	
8	07		
9	6F	INP7	
A	67	OUT7	
B	22	DEC R2	
C	37	B4	
D	0C		
E	30	BR	
F	07		

Begrijpt u hoe het programma werkt. Maak er maar een stroomdiagram van. Door de outputinstructie is R(X) met 1 verhoogd. Hij moet echter op het oorspronkelijke adres blijven wijzen, daarom verminderen we hem met 1 d.m.v. DEC R2. Merk op dat het vullen en R(X) maken van R2 slechts éénmaal hoeft te gebeuren. Dit noemt men initialiseren. Als we nu programmateller R(X) maken wat betekent dit dan voor een outputinstructie. Wel, na het binnenhalen van de instructie wijst de programmateller naar de geheugenplaats volgend op de instructie. Omdat hij tevens register X is, is dit geheugenplaats M(R(X)). De byte direct na de instructie verschijnt op het display. R(X) wordt automatisch met 1 verhoogd en pakt als programmateller netjes de volgende instructie. Ga maar eens terug naar de 4 eerste bytes die u in Cosmicos heeft getikt.

Het volgende programma laat de Q-LED knipperen met een frequentie die bepaald wordt door de byte op adres 0003. Hoe hoger de byte des te langer de frequentie.

adres	byte	mnemonic
0000	E2	(2 machinecy-clussen voor symme-trie)
1	7A	REQ
2	F8	LDI
3	50	
4	BA	PHI RA
5	2A	DEC RA
6	9A	GHI RA
7	3A	BNZ
8	05	
9	31	BQ
A	00	
B	7B	SEQ
C	30	BR
D	02	

Het is niet moeilijk het programma te veranderen zodat de frequentie wordt bepaald door de byte op de data-input. We hebben dan een programmeerbare blokgolfgenerator gekregen.

adres	byte	mnemonic
0000	90	GHI R0 (=00)
1	B2	PHI R2
2	F8	LDI
3	FF	
4	A2	PLO R2
5	E2	SEX R2
6	7A	REQ
7	6F	INP7
8	BA	PHI RA
9	2A	DEC RA
A	9A	GHI RA
B	3A	BNZ
C	09	
D	31	BQ
E	05	
F	7B	SEQ
10	30	BR
1	07	

Het is mischien verstandig in dit stadium Hfdst. 3 nog eens door te lezen om vertrouwd te raken met de architectuur van de Cosmac microprocessor. In het volgende hoofdstuk zullen we een eenvoudig operating system (MOPS) bespreken en tevens een aantal leuke programma's waaronder een universele muziekdoos!

HOOFDSTUK 7

Introductie in het programmeren 2

Tellen met de microprocessor

Het heeft weinig zin om in 't wilde weg codes in het geheugen van Cosmicos te stoppen. Een bekende Engelse zegwijze luidt: ''structure is the essence of programming''. En zo is het precies!

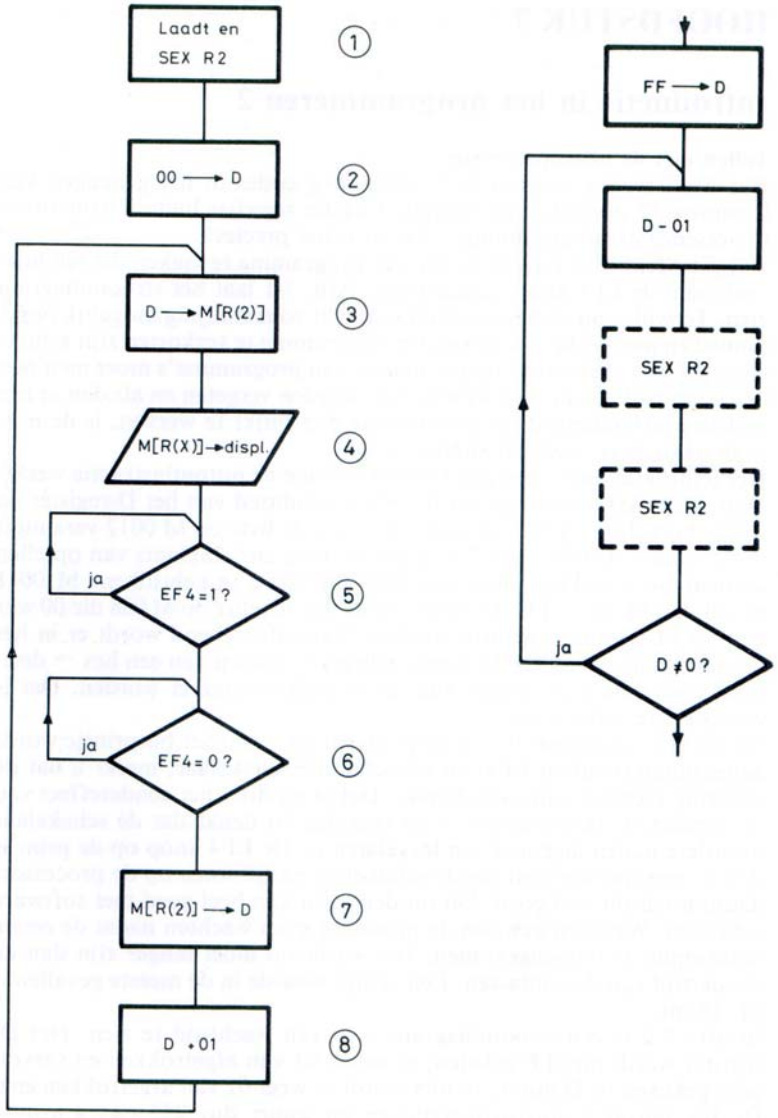
Het is betrekkelijk eenvoudig om een programma te maken dat telt hoeveel maal de EF4 knop is ingedrukt. Afb. 7-1 laat het stroomdiagram zien. Terwille van de begrijpelijkheid is dit zo rechtlijnig mogelijk opgebouwd en mogelijke trucjes om het programma te verkorten zijn achterwege gelaten. Trouwens bij het maken van programma's moet men niet al te slim willen zijn. Die lepigheidjes worden vergeten en als dan in een andere omstandigheid het programma niet blijkt te werken, is de oorzaak maar al te vaak het slimme trucje.

Het telprogramma is een goed voorbeeld hoe de outputinstructie werkt. D.m.v. de ADI-instructie wordt er bij de inhoud van het D-register na iedere keer drukken 01 opgeteld. Als men de byte op M 0012 verandert in 02 springt de teller met 2 tegelijk omhoog enz. Inplaats van optellen kunnen we aftrekken door een SMI-instructie te gebruiken. M 0011 wordt dan FF i.p.v. FC. De beginstand van de teller op M 008 die 00 was zou nu FF gemaakt kunnen worden. Vanzelfsprekend wordt er in het hexadecimale stelsel geteld. Door gebruik te maken van een hex → decimaal conversie programma kan het resultaat omgezet worden. Dat is wat prettiger afleesbaar.

Als nu een schakelaar tussen de punten 1 en 29 op het busprintje wordt aangesloten (Gnd en EF4) en u bedient de schakelaar, merkt u dat de uitlezing vreemde capriolen maakt. Dat komt door het dendereffect van de contacten. De processor is zo snel dat hij denkt dat de schakelaar meerdere malen ingedrukt en losgelaten is. De EF4 knop op de print is d.m.v. een speciale anti-denderschakeling aangesloten op de processor; daarom telt die wel goed. Dit ontdenderen kan heel goed met software gebeuren. We laten gewoon de processor even wachten nadat de eerste schakelpuls is binnengekomen. Die wachttijd moet langer zijn dan de dendertijd van de contacten. Een veilige waarde in de meeste gevallen is ca. 18 ms.

In afb. 7-2 is een stroomdiagram voor een wachttijd te zien. Het D-register wordt met FF geladen; er wordt 01 van afgetrokken en vervolgens gekeken of D nul is; zo niet wordt er weer 01 van afgetrokken enz. De lus omvat 2 instructiecyclussen en duurt dus $255 \times 4 \times 4{,}6\ \mu s =$ ca. 4,5 ms.

Dit is iets tekort en we maken de lus $2\times$ langer met 2 nep-instructies SEX R2; (R2 was al 'gesext'). We hadden ook de NOP-instructie C4 kunnen gebruiken die 3 machinecyclussen duurt. Als we programma 7-1



Afb. 7-1: (links) Stroomdiagram van een programma waarmee wordt geteld (Programma 7-1). Merk op dat een input- en outputhandeling wordt weergegeven met een trapeziumblokje.

Afb. 7-2: (rechts) Stroomdiagram voor een wachttijd.

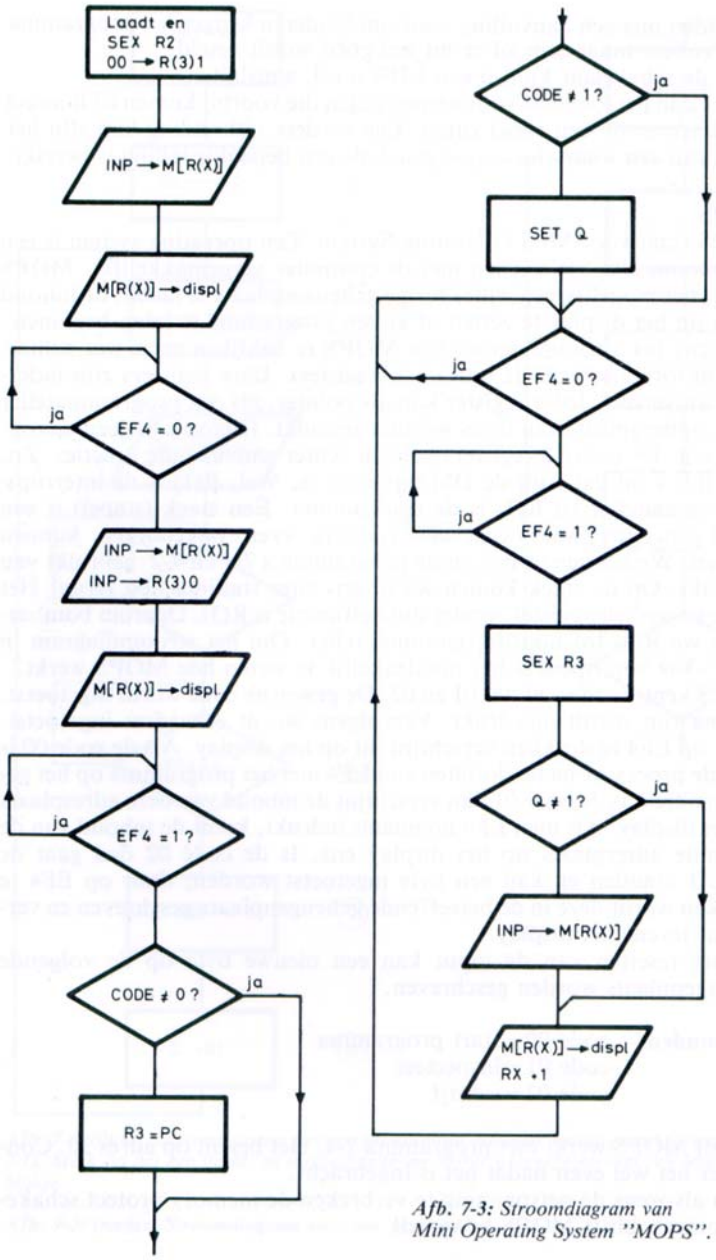
uitbreiden met een aanvulling voor ontdenderen krijgen we programma 7-2. Probeer maar eens of er nu wel goed wordt geteld.
I.p.v. de schakelaar kunt u een LDR o.i.d. aansluiten.
U kunt dan personen of voorwerpen tellen die voorbij komen of hoeveel windingen er op een spoel zitten. Een verdere uitbreiding kan zijn het geven van een waarschuwingssignaal als een bepaalde telling is bereikt.

MOPS

MOPS staat voor Mini OPerating System. Een operating system is een programma dat het werken met de computer vergemakkelijkt. MOPS maakt het mogelijk een willekeurige geheugenplaats te laden, de inhoud ervan op het display te zetten of er een programma te laten beginnen.
Alvorens het stroomdiagram van MOPS te bekijken eerst wat achtergrondinformatie over de universele registers. Deze registers zijn inderdaad universeel. Ieder register kan als pointer, als een programmateller of als opbergplaats van bytes worden gebruikt. Hierop zijn geen uitzonderingen. De eerste 3 registers hebben echter aanvullende functies. Zo, weet u b.v. al dat R(0) de DMA-pointer is. Wel, R(1) is de interruptprogrammateller en R(2) is de stackpointer. Een stack (stapel) is een RAM-geheugen waar gegevens tijdelijk even opgeborgen kunnen worden. We hebben er b.v. in de programma's 7-1 en 7-2 gebruikt van gemaakt. Op de stack komen we in een later stadium nog terug! Het eerste universele register zonder dubbelfunctie is R(3). Daarom bombarderen we R(3) tot hoofdprogramma-teller. Om het stroomdiagram in afb. 7-3 te begrijpen, is het noodzakelijk te weten hoe MOPS werkt.
MOPS kent 3 codes, nl. 00,01 en 02. De gewenste code wordt ingetoetst, **waarna** run wordt ingedrukt. Vervolgens wordt een adres ingetoetst. Door op EF4 te drukken verschijnt dit op het display. Als de code 00 is start de processor na het loslaten van EF4 met het programma op het gekozen adres. Is de code 01 dan verschijnt de inhoud van deze adresplaats op het display. Als men EF4 nogmaals indrukt, komt de inhoud van de volgende adresplaats op het display enz. Is de code 02 dan gaat de Q-LED branden en kan een byte ingetoetst worden; door op EF4 te drukken wordt deze in de betreffende geheugenplaats geschreven en verschijnt tevens het display.
Na het resetten van de input kan een nieuwe byte op de volgende geheugenplaats worden geschreven.

Onthouden/ code 00 = start programma
code 01 = inspecteer
code 02 = schrijf

Test of MOPS werkt met programma 7-4. Het begint op adres 30. Controleer het wel even nadat het is ingebracht.
Als u alvorens de netspanning te verbreken de memory protect schakelaar omzet, blijft MOPS bewaard!

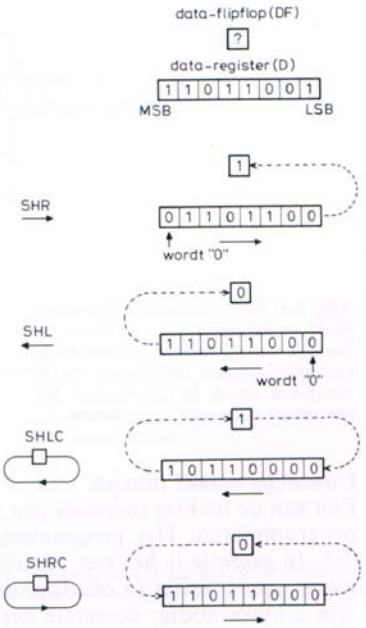


Afb. 7-3: Stroomdiagram van Mini Operating System "MOPS".

Schuiven en de dataflag (DF)
De Cosmac kent 4 schuifinstructies, nl.:

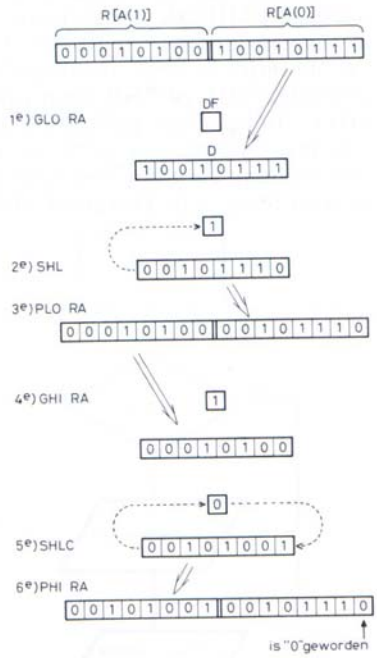
operation	mnemonic	op code
shift right	SHR	F6
shift left	SHL	FE
shift right with carry	SHRC	76
(ring shift right)	(RSHR)	
shift left with carry	SHLC	7E
(ring shift left)	(RSHL)	

De schuifinstructies worden gebruikt om te delen, te vermenigvuldigen, bit en byte manipulatie, en om te testen. Een vermenigvuldiging met 2 wordt verkregen met instructie SHL; terwijl een deling door 2 bereikt wordt met SHR. (In het decimale stelsel wordt met 10 vermenigvuldigd of gedeeld door de komma één plaats te verschuiven). Het verschuiven van de bitjes in beide richtingen kan worden gedaan door de schuifinstructies SHL of SHR maar ook door de schuifinstructie SHRC of SHLC. In het eerste geval (SHL of SHR) wordt het D-register aangevuld met ''nullen'', en de bits die uit de dataflipflop worden geschoven zijn verloren. De schuifinstructies met carry schuiven de bitjes door DF en weer terug in de D-register. Alle bits blijven behouden (afb. 7-4).



Afb. 7-4:
Duidelijker dan woorden geven deze tekeningen het gedrag van de Data flipflop (DF) aan m.b.t. de schuifinstructies.

De toestand van DF kan worden getest met de twee spronginstructies BDF (33) en BNF (3B) en door de twee longskip instructies LSDF (CF) en LSNF(C7). Bij een longskip instructie worden de twee bytes volgend op de longskip instructie overgeslagen als aan de voorwaarden wordt voldaan (to skip is overslaan).
In het MOPS programma wordt de code op de volgende wijze onderzocht. Eerst wordt getest of D = 0; zo ja, dan is de code 00.
Is de code 00, dan wordt eenmaal naar rechts geschoven. Als DF = 1 dan is de code 01. Is dat niet het geval, dan is de code 02 (of een ander oneven getal).
Door gebruik te maken van DF kan een heel blok verschoven worden. Afb. 7-5 laat b.v. zien hoe het 16 bits register RA één plaats naar links geschoven wordt. De dataflipflop is de lijm die de bytes aan elkaar lijmt.

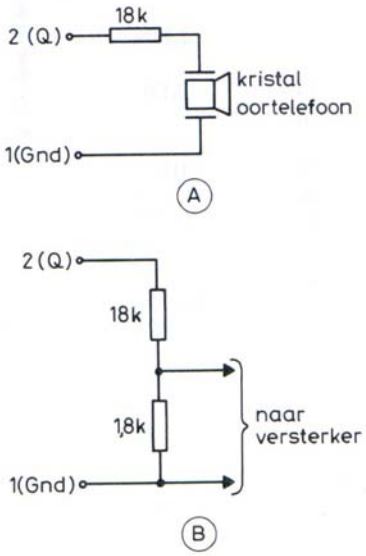


Afb. 7-5: Het schuiven hoeft niet beperkt te blijven tot bytes. Door gebruik te maken van DF kunnen blokken bytes worden verschoven. In dit voorbeeld wordt 16 bits register RA een plaats naar rechts geschoven.

Cosmicos maakt muziek

Een van de leukste toepassingen van Cosmicos is om deze als orgeltje te programmeren. Het programma dat dit bewerkstelligt is programma 7-5. In essentie is het een vertaalprogramma dat een reeks opvolgende bytes steeds omzet in een toonhoogte en een tijdsduur. Per frequentie zijn 2 bytes nodig; de eerste bepaalt de tijdsduur; de tweede de toon-

hoogte. MOPS neemt 37 bytes in beslag; het muziekprogramma 66; er zijn dus 256 – (37 + 66) = 153 bytes over voor de muziek zelf. Dit komt neer op max. 76 tonen.
In de tabel staan een aantal noten vermeld met de daarmee corresponderende byte. Een tweede byte geeft de tijdsduur aan. Deze kan gekozen worden in stappen van 1/8 seconden. Een toonbyte van 00 geeft een rust die eveneens in stappen van 1/8 seconden gekozen kan worden. D.m.v. 2 pointers wordt het begin en eind van de tabel aangegeven. RE staat op het begin en behoeft dus nooit te worden veranderd. RB staat op het einde van de tabel en deze moet dus ingesteld worden, afhankelijk van de lengte van het concert. Dit is de byte op M 002D.
Ter illustratie hebben we een 2-tal muziekfragmenten samengesteld, nl. Vader Jacob en O, kom er eens kijken. Helaas bleken onze muzikale aspiraties hiermee hun absolute top te hebben bereikt. In ieder geval bewijzen ze de deugdelijkheid van het muziekprogramma. Het muziek signaal kan van de punten 1 en 2 (Gnd en Q) op het connectorprintje worden afgenomen. Het eenvoudigste gaat dat met een **kristal** oortelefoontje (Amroh best. nr. ZB 850). In serie met de Q-aansluiting wordt een weerstand van 18 kΩ opgenomen (afb. 7-6a). De geluidssterkte is voldoende om in een geheel vertrek gehoord te worden. Wenst men echter te genieten van de volle, zuivere klank, dan is gebruik van een versterker noodzakelijk (afb. 7-6b).



Afb. 7-6: Door een kristal oortelefoontje tussen Gnd en Q aan te sluiten (serieweerstand 18 kΩ) kan de muziek hoorbaar worden gemaakt. Met een versterker is de klank beter en harder!

adres	byte	mnemonic	
0000	F8	LDI	1
1	00		
2	B2	PHI R2	
3	F8	LDI	
4	FF		
5	A2	PLO R2	
6	E2	SEX R2	
7	F8	LDI	2
8	00		
9	52	STR R2	3
A	67	OUT7	4
B	22	DEC R2	
C	37	B4	5
D	0C		
E	3F	BN4	6
F	0E		
10	02	LDN R2	7
1	FC	ADI	8
2	01		
3	30	BR	
4	09		

Programma 7-1

adres	byte	mnemonic	
0000	F8	LDI	
1	00		
2	B2	PHI R2	
3	F8	LDI	
4	FF		
5	A2	PLO R2	
6	E2	SEX R2	
7	F8	LDI	
8	00		
9	52	STR R2	
A	67	OUT7	
B	22	DEC R2	
C	37	B4	
D	0C		
E	F8	LDI	wacht even!
F	FF		
10	FF	SMI	
1	01		
2	E2	SEX R2	
3	E2	SEX R2	
4	3A	BNZ	
5	10		
6	3F	BN4	
7	16		
8	F8	LDI	wacht even!
9	FF		
A	FF	SMI	
B	01		
C	E2	SEX R2	
D	E2	SEX R2	
E	3A	BNZ	
F	1A		
20	02	LDN R2	
1	FC	ADI	
2	01		
3	30	BR	
4	09		

Programma 7-2

adres	byte	mnemonic	
0000	90	GHI R0	(=00)
1	B2	PHI R2	
2	B3	PHI R3	
3	F8	LDI	
4	FE		
5	A2	PLO R2	
6	E2	SEX R2	
7	6F	INP 7	
8	67	OUT 7	
9	3F	BN4	
A	09		
B	6F	INP 7	
C	A3	PLO R3	
D	67	OUT 7	
E	37	B4	
F	0E		
10	22	DEC R2	
1	22	DEC R2	
2	42	LDA R2	
3	3A	BNZ	
4	16		
5	D3	SEP R3	
6	F6	SHR	
7	CF	LSDF	
8	7B	SEQ	
9	C4	NOP	
A	3F	BN4	
B	1A		
C	37	B4	
D	1C		
E	E3	SEX R3	
F	39	BNQ	
20	22		
1	6F	INP 7	
2	67	OUT 7	
3	30	BR	
4	1A		

Programma 7-3

adres	byte	mnemonic	
0030	93	GHI R3	
1	BA	PHI RA	
2	AA	PLO RA	
3	9A	GHI RA	
4	52	STR R2	(=RX)
5	67	OUT 7	
6	22	DEC R2	
7	1A	INC RA	
8	F8	LDI	
9	FF		
A	FF	SMI	
B	01		
C	3A	BNZ	
D	3A		
E	8A	GLO RA	
F	3A	BNZ	
40	37		
1	30	BR	
2	33		

Programma 7-4

adres	byte	mnemonic		
0025	93	GHI R3		
6	BE	PHI RE		
7	BB	PHI RB		
8	BA	PHI RA		
9	F8	LDI		
A	67		←begin tabel	
B	AE	PLO RE		
C	F8	LDI		
D			←eind tabel	
E	AB	PLO RB		
F	F8	LDI		
30	02			loop 2
1	AC	PLO RC		
2	4E	LDA RE		
3	AA	PLO RA		
4	8A	GLO RA		
5	FE	SHL		SHL RA
6	AA	PLO RA		(tijd)
7	9A	GHI RA		
8	7E	SHLC		
9	BA	PHI RA		

```
A   2C   DEC RC
B   8C   GLO RC      Loop 2 = 0?
C   3A   BNZ
D   34
E   0E   LDN RE
F   32   BZ           rust?
40  5B
1   7B   SEQ
2   0E   LDN RE
3   FF   SMI
4   01
5   3A   BNZ          Wek
6   43                1 cyclus
7   39   BNQ          op
8   4E
9   7A   REQ
A   E2
B   E2
C   30   BR
D   42
E   2A   DEC RA
F   9A   GHI RA       tijd om?
50  3A   BNZ
1   41
2   8E   GLO RE
3   52   STR R2
4   1E   INC RE       einde
5   8B   GLO RB       tabel?
6   F3   XOR
7   3A   BNZ
8   2F
9   30   BR           ja!
A   25
B   F8   LDI
C   7F
D   FF   SMI          rust!
E   01
F   3A   BNZ
60  5D
1   2A   DEC RA
2   9A   GHI RA
3   3A   BNZ
4   5B
5   30   BR
6   52
```

7	tijd	←	begin tabel
8	toon	noot 1	
9	tijd		
A	toon	noot 2	
B	tijd		
C	toon	noot 3	
D	tijd		
E	toon	noot 4	
	enz.		

Programma 7-5

Toon			Tijd in s							
Noot	**Freq. Hz**	**HEX**	**1/8**	**2/8**	**3/8**	**4/8**	**5/8**	**6/8**	**7/8**	**8/8**
C	130,8	CA	44	47	4B	4F	52	56	5A	5D
C#	138,6	C0	44	48	4C	50	53	57	5B	5F
D	146,8	B4	44	49	4D	52	56	5A	5F	63
E	164,8	A0	45	4A	4E	53	58	5D	61	66
F	174,6	96	45	4A	4F	55	5A	5F	64	69
F#	185,0	8C	46	4B	51	56	5C	61	67	6C
G	196,0	86	46	4B	52	58	5D	63	69	6F
G#	207,6	7E	46	4C	52	5A	5F	65	6B	71
A	220,0	79	47	4D	54	5B	61	68	6E	75
A#	233,1	72	47	4E	55	5D	64	6B	72	79
B	247,0	6B	47	4F	56	5E	65	6C	75	7B
C	261,6	65	49	53	5C	66	6F	78	82	88
C#	277,1	60	4A	54	5D	67	71	7B	84	8E
D	293,1	5A	4A	54	5F	69	73	7E	88	92
D#	311,1	55	4B	55	60	6B	75	80	8A	95
E	329,6	50	4B	56	61	6C	76	83	8E	99
F	349,2	4B	4C	57	63	6F	79	86	91	9D
F#	372,1	46	4C	59	65	71	7D	8A	96	A2
G	392,0	43	4D	5A	66	73	80	8D	99	A6
G#	415,5	3F	4D	5B	68	75	82	90	6D	AA
A	440	3B	4E	5C	6A	78	86	94	A2	B0
A#	466,5	37	4F	5D	6C	7B	89	98	A6	B5
B	493,9	34	4F	5E	6D	7D	8C	9B	AA	B9
C	523,2	31	50	60	70	81	91	A1	B1	C1
C#	554,0	3E	51	62	72	83	94	A5	B5	C6
D	587,3	2B	52	64	75	87	99	AB	BC	CE
D#	622,0	29	53	65	78	8A	9D	AF	C2	D4
E	659,3	26	54	67	7B	8E	A2	B6	C9	DD
F	698,5	24	55	69	7E	92	A7	BB	D0	E5
F#	742,0	22	56	6B	81	97	AD	C1	D7	EE
G	784,0	20	57	6E	84	9B	B2	C9	DE	F6
G#	880,0	1E	57	6F	86	9D	B5	CC	E3	FF
rust	00	00	4C	5A	67	75	81	8D	99	A7

Tabel 7-1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0060								66	65	69	5A	6C	50	60	65	6C
0070	50	6F	4B	73	43	00	5A	43	5C	3B	5A	43	57	4B	6C	
0080	50	66	65	41	00	66	65	58	87	66	65	75	00			

Tabel 7-2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0060								86	4B	59	46	57	4B	59	46	90
0070	3F	A5	2E	5D	60	41	00	54	60	55	55	54	60	55	55	59
0080	4B	57	46	59	4B	80	55	FF	00							

Tabel 7-3

HOOFDSTUK 8

Introductie in het programmeren 3.

De stack

''Stack is het Engelse woord voor stapel. In het verloop van een programma is het regelmatig noodzakelijk tussentijdse gegevens even te bewaren tot ze voor verdere verwerking weer nodig zijn. Dit bewaren kan uiteraard alleen gebeuren in een gedeelte van het geheugen waar kan worden geschreven en gelezen, dus RAM. Meestal worden de programma's in het onderste gedeelte van het geheugen ondergebracht dat, naarmate meer programma's worden ingebracht, in opwaartse richting wordt gevuld. Tijdens bedrijf worden deze programma's afgetast door de programmateller.

De programma's zijn zo ontworpen dat ze uitsluitend worden gelezen; dit om mogelijk te maken ze later in ROM of EPROM onder te brengen. De opslag van tussentijdse variabelen gebeurt op de stack en om nu conflicten tussen het programmageheugen en het stackgeheugen zo lang mogelijk uit te stellen, kiest men de stack zo hoog mogelijk. Dit zo hoog mogelijk is natuurlijk betrekkelijk. Cosmicos heeft nu nog maar een geheugen van 256 bytes. De hoogst mogelijke geheugenplaats is dus FF_{16}.

Een stack werkt volgens het principe: het laatste erin; het eerste eruit (last in; first out - LIFO). Er zijn vele goede redenen om een stack te gebruiken. De belangrijkste is wel dat er aldus een efficiënt gebruik wordt gemaakt van de beschikbare geheugenruimte. Immers, het aantal vereiste stackplaatsen komt overeen met het maximum aantal bytes dat op een zeker ogenblik moet worden opgeslagen en niet met het totale aantal over de gehele duur van het programma. Over het algemeen is de ''diepte'' van de stack klein.

Om de bytes op de stack te zetten en er vanaf te halen wordt een apart register gebruikt. Dit is, uitzonderingen daargelaten, de enige taak van dit register, dat de stackpointer wordt genoemd.

De vraag rijst: welk van de 16 universele registers van de Cosmac moet stackpointer zijn of maakt het niets uit? Wel, in dit stadium maakt het nog niets uit, maar later als we met interrupts gaan werken is het noodzakelijk register 2 te kiezen. Als de interrupt-aanvraag nl. wordt gehonoreerd, wordt register 2 tot X-register gemaakt en R1 tot programmateller. Ook de control-instruction ''MARK'' heeft betrekking op register 2. Veel data-manipulaties gebeuren op de stack en om die reden maakt men register 2 tevens X-register. Dit sluit de mogelijkheid niet uit om, als dat in de loop van een programma zo uitkomt, een ander register tot X-register te bombarderen. Dat kan altijd, als we er maar rekening mee houden.

Een stack wordt van boven naar beneden gevuld en de stackpointer wijst altijd naar de eerstvolgende vrije geheugenplaats. De reden hiervoor zal

duidelijk worden bij de bespreking van interrupts.
Aangezien normaal gesproken register 2 en register X één en dezelfde zijn, kunnen we de volgende instructies gebruiken om bytes op stack te zetten.

1e)	Store via N	STR	52
2e)	Store via X en decrement	STXD	73

In het eerste geval wordt de byte in het data register gecopieerd op de geheugenplaats waar R2 naar wijst. In het tweede geval wordt de byte daar eveneens gecopieerd maar **vervolgens** wordt RX = R2 met één verlaagd. Deze instructie is dus zeer geschikt om meerdere bytes op de stack te zetten. Instructies om bytes van de stack te halen zijn:

1e)	Load via N	LDN	02
2e)	Load advance	LDA	42
3e)	Load via X	LDX	F0
4e)	Load via X and advance	LDXA	72

Bij de ''advance'' instructie wordt **na** het copiëren van de byte in het Data-register de pointer met één verhoogd. Dit zijn dus geschikte instructies om meerdere bytes van de stack te halen.
Zolang X en N betrekking hebben op hetzelfde register kunnen de desbetreffende instructies door elkaar worden gebruikt.
De volgende sequentie zet universeel register 8 op de stack.

```
GHI   R8    98
STDX        73
GLO   R8    88
STDX        73
```

De inhoud van R8 is nu ''gered'' en R8 is beschikbaar voor een andere taak. Na het vervullen van deze taak wordt R8 weer hersteld.

```
INC   R2    12
LDA   R2    42
PLO   R8    A8
LDN   R2    02
PHI   R8    B8
```

We hadden voor hetzelfde geld gebruik kunnen maken van:
IRX, LDXA en LDX.
Het onderscheid gaat een rol spelen bij zeer geraffineerde toepassingen.

Het is niet noodzakelijk de stackpointer apart te initialiseren. Dit heeft MOPS al gedaan.
R2 is RX en wijst naar 00FF zodra we via MOPS een programma laten draaien. MOPS zorgt ook dat R3 de programmateller is.

Adressering; immediate en register indirect

Laat u zich niet afschrikken door deze termen. Als men het eenmaal weet is het eenvoudig. De Cosmac-instructieset kent een groot aantal logische-en rekenkundige instructies. Alle instructies die betrekking hebben op twee grootheden kunnen ''immediate'' (Onmiddellijk) of ''register-indirect'' zijn. In beide gevallen bevindt één grootheid (operand) zich in het D-register en de ander in het geheugen. Bij de immediate instructies staat de tweede operand onmiddellijk achter de instructie en wordt aangewezen (geadresseerd) door de programmateller; b.v.

	EXclusive OR immediate	XRI	FB
	operand ⟶		XX
of			
	ADD immediate	ADI	FC
	operand ⟶		XX

De immediate instructies zijn gemakkelijk te herkennen omdat de mnemonic op ''I'' eindigd.
Bij de register indirect instructies bevindt de tweede operand zich op de geheugenplaats waar register X naar wijst. Dit zal bijna altijd op de stack zijn. De operand wordt dus middels een register aangewezen (vandaar register indirect) en niet onmiddellijk door de programmateller. Het grote voordeel van register-indirecte adressering is dat met variabelen gewerkt kan worden. De stack is immers RAM.
Bij immediate instructies bevindt de operand zich in de programmastroom. Het programma moet ook in een ROM of EPROM kunnen worden gezet. Deze operand is dus een constante.

Logische instructies

Behalve de schuifinstructies kent de Cosmac drie logische constructies die zowel immediate als register indirect kunnen zijn, nl. OR, AND en EXclusive OR.
Wat kunnen we met deze instructies doen?

a) Met de OR-instructies kunnen afzonderlijke bits ''1'' worden gemaakt.

M(R)X of M(R)P	D	OR (resultaat in D)
0	0	0
0	1	1
1	0	1
1	1	1

b) Met de AND-instructies kunnen afzonderlijke bits ''0'' worden gemaakt.

M(R)X of M(R)P	D	AND (resultaat in D)
0	0	0
0	1	0
1	0	0
1	1	1

c) Met de EXOR-instructie kunnen de bits worden getest op gelijkheid. Gelijke waarden resulteren in een ''0''. Evenzo resulteren 2 bytes die gelijk zijn in 8 nullen in het D-register.

M(R)X of M(R)P	D	XOR (resultaat in D)
0	0	0
0	1	1
1	0	1
1	1	0

Door een byte te ''XORren'' met FF_H worden de oorspronkelijke enen nullen en de nullen enen. M.a.w., men heeft het complement van de byte gekregen, ofwel de byte is geïnverteerd.
Met bovenstaande drie basisinstructies zijn alle logische functies realiseerbaar. (NOF b.v. door eerst te ''OF-fen'' en vervolgens te ''XORren'' met FF_H).
Stel nu dat we van een byte, die b.v. afkomstig is van een ASCII keyboard, alleen belang hechten aan de laagste 7 bits
(dus b_0 t/m b_6). Alvorens tot verder verwerking over te gaan moet bit 7 ''0'' worden gemaakt. Bits 0 t/m 6 mogen niet veranderen

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
AND	0	1	1	1	1	1	1	1	($7F_H$)
	0	b_6	b_5	b_4	b_3	b_2	b_1	b_0	

U ziet, dit gaat zeer eenvoudig door de byte te ''AND-en'' met $7F_H$.

Willen we van een byte een bitje ''1'' maken zonder de andere bits te veranderen, dan maken we gebruik van de OR-functie.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
OR	1	0	0	0	0	0	0	0	(80_H)
	1	b_6	b_5	b_4	b_3	b_2	b_1	b_0	

Willen we bitjes omkeren dan gebruiken we de EXOR-functie.

	0	1	b_5	b_4	b_3	b_2	b_1	b_0	
EXOR	1	1	0	0	0	0	0	0	($C0_H$)
	1	0	b_5	b_4	b_3	b_2	b_1	b_0	

De bytes 7F, 80 en C0 worden ''maskers'' genoemd omdat ze zorgen dat de logische operatie uitsluitend op de geselecteerde bitjes betrekking heeft. De overige bitjes worden ''gemaskeerd''.
Verder is het nog van belang te weten dat de Data-flag (DF) door logische instructies niet wordt veranderd.

Rekenkundige instructies
De rekenkundige instructies kunnen in 3 groepen worden verdeeld. Deze groepen zijn:

1e) optellen;	ADD	[M(R)P of M(R)X] + D
2e) aftrekken;	SUBTRACT D	[M(R)P of M(R)X] – D
3e) aftrekken;	SUBTRACT memory	D – [M(R)P of M(R)X]

Al deze instructies kennen weer immediate of register indirect zijn. De Data-flag speelt bij de rekenkundige instructies een belangrijke rol en wel als volgt. Bij de optel-instructie wordt ze gebruikt om een overdracht aan te geven (Carry). Is de som van de 2 operands kleiner dan FF_H dan zal DF ''0'' zijn. Is de som groter dan FF_H dan is DF ''1''. DF kan dus ''0'' worden gemaakt zonder de inhoud van het D-register te veranderen door daar 00 bij op te tellen.
De ADD with Carry instructies worden gebruikt als men getallen bestaande uit meerdere bytes wil optellen. We komen hier nog op terug! De aftrek-instructies vormen een apart verhaal. Er wordt nl. niet afgetrokken maar van de operand in het Data-register wordt het complement genomen dat vervolgens bij de operand in het geheugen wordt opgeteld. Deze methode van aftrekken kenmerkt trouwens niet alleen de Cosmac; nagenoeg alle microprocessors trekken af volgens de z.g. 2-complement methode. Aan de hand van een getallen voorbeeld is eenvoudig aan te tonen dat het werkt. Terwille van de eenvoud nemen we

twee decimale getallen tussen 0 en 9. We moeten dan het 10-complement nemen i.p.v. het 2-complement. Als de uitkomst groter is dan 9 krijgen we een overdracht.

$$7—4 = 7 + \underbrace{(10—4)}_{\text{10-complement}} = 13$$

De uitkomst is 13 d.w.z. 3 met een overdracht. Als dus na een aftrek-instructie door de processor het verschil positief of nul is dan ontstaat er een carry (DF = 1).

$$5—7 = 5 + \underbrace{(10—7)}_{\text{10-complement}} = 8$$

Bij dit voorbeeld is het verschil negatief. Er is geen carry (DF = 0) en het antwoord is ook niet goed. Als we echter het 10-complement nemen krijgen we 2 en dat is wel het goede antwoord.

Rekenen en de dataflag (DF)

Als na een optel-instructie het antwoord groter is dan de 8 bits van het Data-register onstaat er een carry; DF = 1. Het antwoord is in alle gevallen juist. Als na een aftrek-instructie DF = 1; is het antwoord positief of nul en volkomen correct. Als echter na een aftrek-instructie DF = 0; dan is het antwoord negatief.Het gedrag van DF is met het volgende ezelsbruggetje gemakkelijk te onthouden. Ga er van uit dat vòòr het aftrekken DF = 1. Is DF na het aftrekken ''0'' dan is er geleend (borrow). Is DF echter ''1'' gebleven dan is er niet geleend.
De stand van DF kan worden gebruikt om te onderzoeken welke van 2 bytes de grootste of kleinste is.

HOOFDSTUK 9

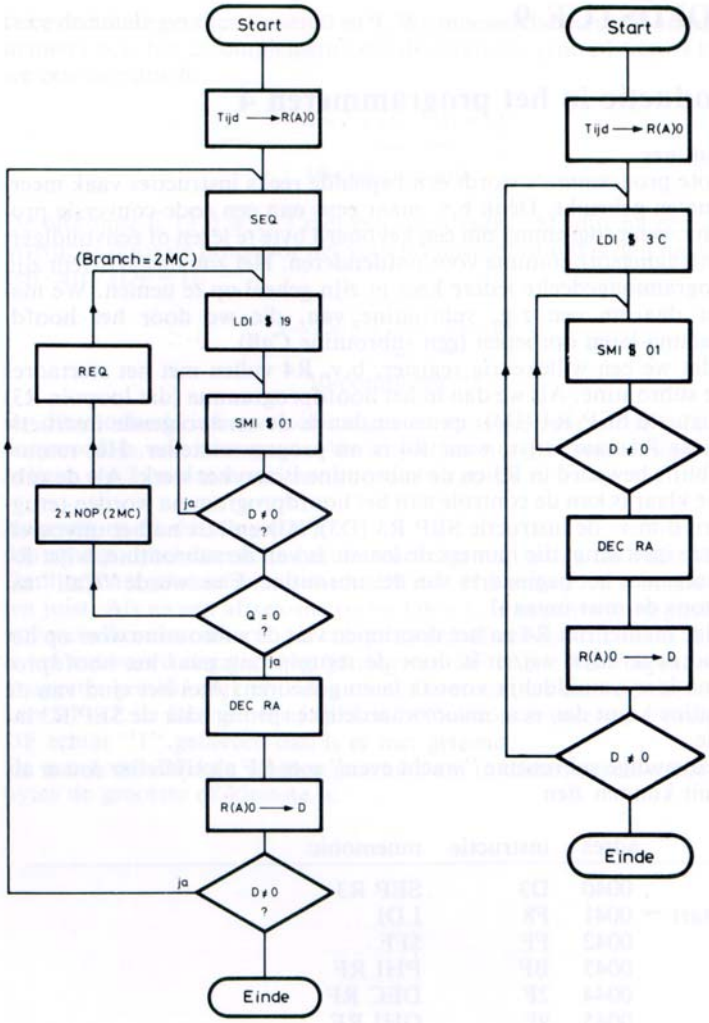
Introductie in het programmeren 4

Subroutines

In grote programma's wordt een bepaalde reeks instructies vaak meerdere malen gebruikt. Denk b.v. maar eens aan een code-conversie programma, een programma om een keyboard byte te lezen of eenvoudiger: een vertragingsprogramma voor ontdenderen. Het zou niet efficiënt zijn dit programmagedeelte iedere keer in zijn geheel op te nemen. We maken er daarom een z.g. subroutine van, die we door het hoofdprogramma laten oproepen (een subroutine Call).
Stel, dat we een willekeurig register, b.v. R4 vullen met het startadres van de subroutine. Als we dan in het hoofdprogramma (dat loopt in R3) de instructie SEP R4 (D4) opnemen dan is de eerstvolgende instructie die, waar R4 naar wijst, want R4 is nu programmateller. Het retouradres blijft bewaard in R3 en de subroutine is aan het werk. Als de subroutine klaar is kan de controle aan het hoofdprogramma worden teruggegeven d.m.v. de instructie SEP R3 (D3). Alleen . . . na het uitvoeren van deze instructie, die immers de laatste is van de subroutine, wijst R4 niet meer naar het beginadres van de subroutine. Een tweede ''Call'' zal grandioos de mist ingaan!
De enige manier om R4 na het doorlopen van de subroutine weer op het beginadres te laten wijzen is door de terugsprong naar het hoofdprogramma hier onmiddelijk voor te laten gebeuren. Aan het eind van de subroutine komt dan een onvoorwaardelijke sprong naar de SEP R3 instructie.
Een eenvoudige subroutine ''wacht even'' met RF als tijdteller zou er als volgt uit kunnen zien.

	adres	instructie	mnemonic
	0040	D3	SEP R3
start →	0041	F8	LDI
	0042	FF	$FF
	0043	BF	PHI RF
	0044	2F	DEC RF
	0045	9F	GHI RF
	0046	3A	BNZ
	0047	44	
	0048	30	BR
	0049	40	

Merk op dat het startadres van deze subroutine 0041_H is. Dit is dan ook het adres waarmee R4 gevuld moet worden. De op één na laatste instruc-



Afb. 9-1: (links) Het stroomdiagram van subroutine Dottie. Dottie wekt een toon op ter lengte van een punt.

Afb. 9-2: (rechts) Het stroomdiagram van Rustie. Rustie wekt een pauze op ter lengte van een punt. Merk op dat bij Dottie en Rustie de sprong terug naar de SEP-instructie nog ontbreekt. Pas dan zijn het subroutines geworden.

tie is een short branch naar 0040_H. Hier staat instructie SEP R3. Na het uitvoeren hiervan staat R4 weer op 0041_H.

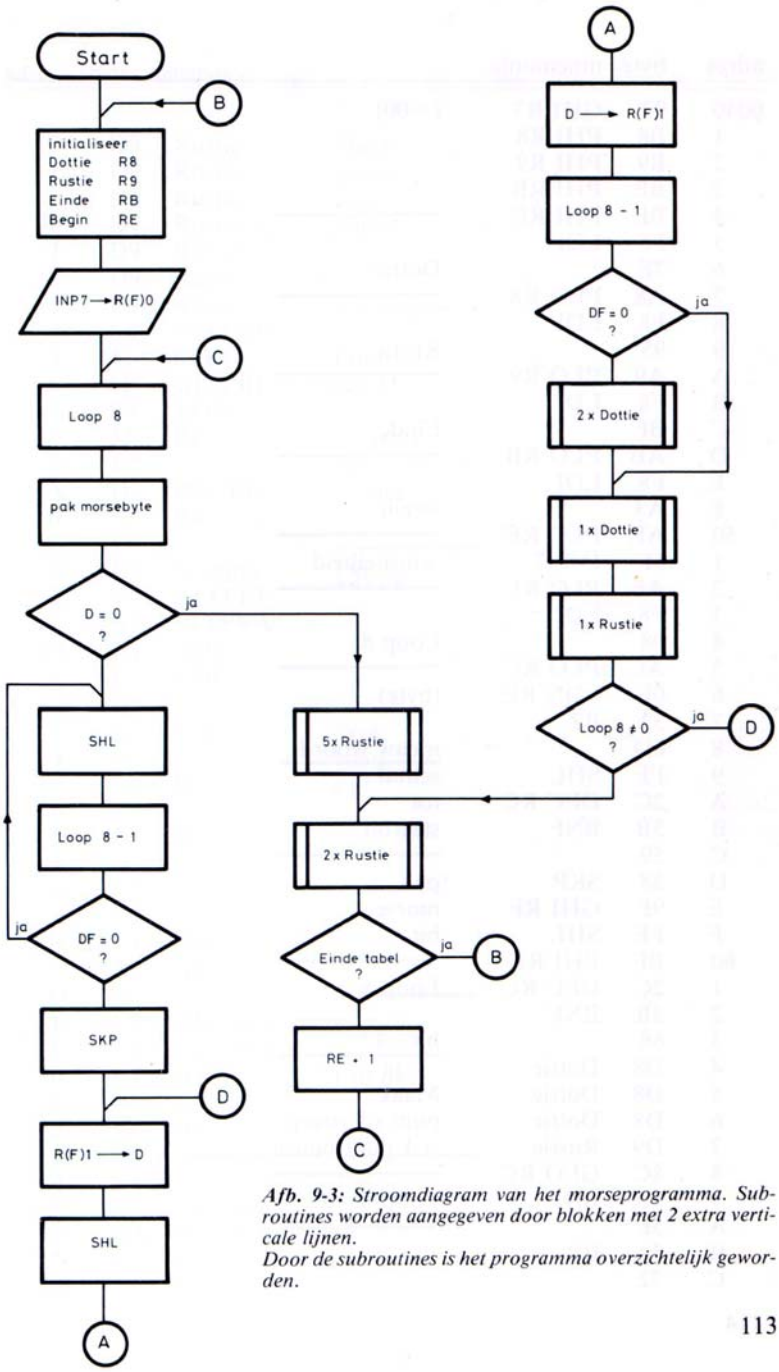
Cosmicos als morseseiner

Het morse-alfabet is door de opbouw uit punten en strepen a.h.w. geschapen voor verwerking door een computer. Hoewel heden ten dage het belang van dit communicatiesysteem niet zo groot meer is als vroeger, is kennis ervan voor radio zendamateurs noodzakelijk. Uitgezonderd voor de D-machtiging moet er nl. een examen in worden gedaan. Cosmicos kan door zijn ''onuitputtelijk geduld'' behulpzaam zijn bij het op doen van de noodzakelijke ervaring. Om der wille van de eenvoud wordt uitgegaan van een in het geheugen opgeslagen tabel met morse karakters. Koppeling met een ASCII keyboard behoort echter zonder meer tot de mogelijkheden. Door gebruik te maken van een input buffer is het dan tevens mogelijk een vloeiende aanpassing te krijgen op de seinsnelheid. Met wat extra geheugenruimte en software gestuurde display's is ontvangst van het morse alfabet ook mogelijk.
Het morse alfabet is opgebouwd uit punten en strepen. Een streep duurt 3 punten. Tussen de punten en/of strepen van een karakter is een pauze van 1 punt opgenomen. Tussen de karakters van een woord een pauze van 3 punten en tussen de woorden een pauze van 7 punten.
Met uitzondering van het ''error'' teken bestaan alle karakters uit minder dan 8 punten en/of strepen.
Het aantal is verschillend en ligt tussen 1-6. Een voor de hand liggend idee is om voor ieder morsteken een byte te nemen. Een bitje dat ''1'' is staat voor een streep en een bitje dat ''0'' is voor een punt. Door de byte naar links te schuiven en DF te testen kunnen de bitjes onderzocht worden. Het aantal benodigde bitjes is echter niet voor ieder karakter hetzelfde. Iedere byte moet dus een soort ''startbit'' bevatten ten teken dat de eerstvolgende bit de eerste punt of streep van het morseteken is.
Voorbeeld

teken	Morse	**byte**
A	• —	00000101
E	•	00000010
7	— — •••	00111000

De eerste 1 van links gerekend is de startbit. Indien na iedere punt of streep automatisch een pauze van een punt wordt opgewekt kunnen de 8 bits van een byte achter elkaar worden verwerkt. Tussen de bytes komt dan een extra pauze van 2 punten; totaal dus een pauze van een 3 punten tussen de letters. Tussen 2 woorden kan dan een byte worden opgenomen bestaande uit 8 nullen. Dit kan in het programma worden onderzocht met de: short branch if D = 0, instructie. Een extra pauze van 5 punten wordt dan ingelast. Voor het opwekken van de punten en stre-

pen gebruiken we subroutine ''Dottie''. Deze wekt een toon op met een tijdsduur van 1 punt. Voor een streep wordt Dottie 3× achter elkaar geroepen. De pauze wordt opgewekt met subroutine ''Rustie''. Deze wekt een pauze op met een tijdsduur van 1 punt. Zonodig wordt ook Rustie meerdere malen achter elkaar opgeroepen.
De tijdreferentie voor Dottie en Rustie is bepalend voor de seinsnelheid. Deze bevindt zich in R(F)0 en kan via het hoofdprogramma worden veranderd.
De stroomdiagrammen van Dottie en Rustie zijn afzonderlijk getekend. De aandacht kan nu worden gericht op het stroomdiagram van het hoofdprogramma (afb. 9-3).
Begonnen wordt met enige voorbereidende werkzaamheden; het laden van de programmatellers voor Dottie en Rustie en het instellen van de begin- en eindpointer. Vervolgens wordt de byte op input-binair naar R(F)0 gebracht; dit is de seinsnelheid.
Iedere byte moet in totaal 8× worden geschoven. Hierover gebruiken we een aparte teller; de ''Loop 8'' teller R(C)0.
De morsebyte wordt van de tabel gehaald; is de byte nul dan krijgen we 7× Rustie; een controle op einde tabel en de volgende byte wordt gepakt. Als de byte niet nul is dan wordt naar links geschoven tot de startbit. Na iedere keer schuiven wordt Loop 8 met 1 verminderd. Het nu volgende bitje is een punt of een streep en afhankelijk van DF wordt 3× Dottie of 1× Dottie geroepen. Het resultaat van tussentijds geschuif wordt bewaard in R(F)1. Na de laatste Dottie komt 1× Rustie en dan wordt gekeken of Loop 8 nul is; zo nee, opnieuw een SHL enz. Zo ja, 2× Rustie en controle einde tabel. Als de laatste byte van de tabel is geweest wordt opnieuw begonnen, waarbij tevens de seinsnelheid wordt ingesteld. We zeggen niet wat er in de morsetabel staat. Als u het morsealfabet machtig bent kunt u het zo horen, anders wordt het puzzelen. Het zal niet veel moeite opleveren zelf een willekeurige tekst in morse om te vormen. Belangrijk is echter wel dat de eindpointer goed staat.



Afb. 9-3: Stroomdiagram van het morseprogramma. Subroutines worden aangegeven door blokken met 2 extra verticale lijnen.
Door de subroutines is het programma overzichtelijk geworden.

adres	byte	mnemonic		
0040	93	GHI R3		(=00)
1	B8	PHI R8		
2	B9	PHI R9		
3	BB	PHI RB		
4	BE	PHI RE		
5	F8	LDI		
6	7E			Dottie
7	A8	PLO R8		
8	F8	LDI		
9	95			Rustie
A	A9	PLO R9		
B	F8	LDI		
C	BF			Einde
D	AB	PLO RB		
E	F8	LDI		
F	A3			Begin
50	AE	PLO RE		
1	6F	INP 7		seinsnelheid
2	AF	PLO RF		
3	F8	LDI		
4	08			Loop 8
5	AC	PLO RC		
6	0E	LDN RE		(byte)
7	32	BZ		
8	6D		→	nieuw woord
9	FE	SHL		schuif
A	2C	DEC RC		tot
B	3B	BNF		startbit
C	59			
D	38	SKP		pak
E	9F	GHI RF		morse-
F	FE	SHL		bit
60	BF	PHI RF		
1	2C	DEC RC		Loop 8-1
2	3B	BNF		
3	66			bit = 1?
4	D8	Dottie		ja
5	D8	Dottie		Maak
6	D8	Dottie		punt of streep
7	D9	Rustie		+1 punt pauze
8	8C	GLO RC		
9	3A	BNZ		Loop 8 = 0?
A	5E			
B	30	BR		
C	72			

adres	byte	mnemonic	
D	D9	Rustie	Maak
E	D9	Rustie	pauze
F	D9	Rustie	2 of 7
70	D9	Rustie	punt
1	D9	Rustie	
2	D9	Rustie	
3	D9	Rustie	———
4	8B	GLO RB	
5	52	STR	Einde
6	8E	GLO RE	tabel?
7	F3	XOR	
8	32	BZ	ja→
9	40		———
A	1E	INC RE	nee
B	30	BR	
C	53		———
D	D3	Retpgm	Dottie
→ E	8F	GLO RF	
F	AA	PLO RA	
80	7B	SEQ	
1	F8	LDI	
2	19		
3	FF	SMI	Wek
4	01		toon
5	3A	BNZ	op!
6	83		
7	39	BNQ	
8	8E		
9	E2		
A	E2		
B	7A	REQ	
C	30	BR	
D	81		———
E	2A	DEC RA	
F	8A	GLO RA	
90	3A	BNZ	tijd om?
1	80		
2	30	BR	
3	7D	———	———
4	D3	Retpgm	Rustie
→ 5	8F	GLO RF	
6	AA	PLO RA	
7	F8	LDI	

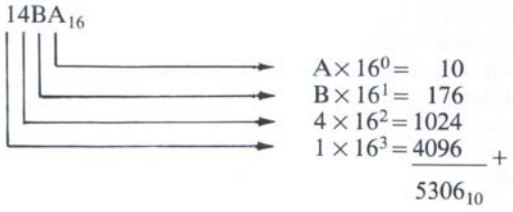
adres	byte	mnemonic	
8	3C		Wacht
9	FF	SMI	even!
A	01		
B	3A	BNZ	
C	99		
D	2A	DEC RA	
E	8A	GLO RA	
F	3A	BNZ	tijd om?
A0	97		
1	30	BR	
2	94		
3	0A		Tabel
4	05		
5	0C		
6	04		
7	0F		
8	00		
9	18		
A	09		
B	14		
C	14		
D	02		
E	03		
F	04		
B0	06		
1	00		
2	02		
3	06		
4	00		
5	1A		
6	0F		
7	08		
8	07		
9	04		
A	1A		
B	0F		
C	08		
D	00		
E	00		
F	00		

Programma 9-1: *Dit programma zet de tabel die begint op M00A3 om in morse volgens de in de tekst beschreven methode. U kunt zelf ook een morsetabel maken als u er maar aan denkt eindpointer RB op de laatste byte te zetten.*

Code conversie: hexadecimaal → decimaal

Het resultaat van een meting verschijnt als hexadecimaal getal op het display van de computer. Dit is, omdat we gewend zijn in het decimale stelsel te denken, voor ons moeilijk te interpreteren. Het is echter best mogelijk de computer de hexadecimale waarde om te laten zetten in een decimaal equivalent.

Volgens de rekenkundige manier zou b.v. het hexadecimale getal $14BA_{16}$ als volgt kunnen worden omgezet.



Helaas beschikken we nog niet over het hiervoor benodigde vermenigvuldigings programma.

Er bestaat echter een andere omzettings methode waarbij vermenigvuldigen vervangen wordt door schuiven. Om de zaak niet te ingewikkeld te maken gaan we uit van een hexadecimaal getal met twee cijfers: dus maximaal FF_{16}. Dit komt overeen met decimaal 255_{10}; drie cijfers. Zowel decimaal als hexadecimaal neemt ieder cijfer 4 bitjes; een nibble, in beslag. Voor het decimale getal hebben we 3 nibbles nodig. De eerste nibble geeft de eenheden aan; de afzonderlijke bits hebben de volgende waarden:

	$8\times1,\quad 4\times1,\quad 2\times1,\quad 1\times1.$
Nibble 2 geeft de 10-tallen weer:	$8\times10,\quad 4\times10,\quad 2\times10,\quad 1\times10.$
Nibble 3 geeft de 100-tallen:	$8\times100,\ 4\times100,\ 2\times100,\ 1\times100.$

We schuiven nu een hexadecimaal getal bit voor bit in het eerste nibble. Decimale cijfers lopen van 0 tot 9; een nibble mag dus maximaal het cijfer 9 bevatten. Zodra een cijfer hoger dan 9 verschijnt moeten we van dat nibble een waarde 10 aftrekken en bij het volgende nibble een waarde 1 optellen. Deze correctie kan echter ook voor het schuiven worden uitgevoerd. Zodra nl. in een nibble voor het schuiven een waarde staat groter dan 4 zal dit na schuiven een waarde groter dan 9 tot gevolg hebben.

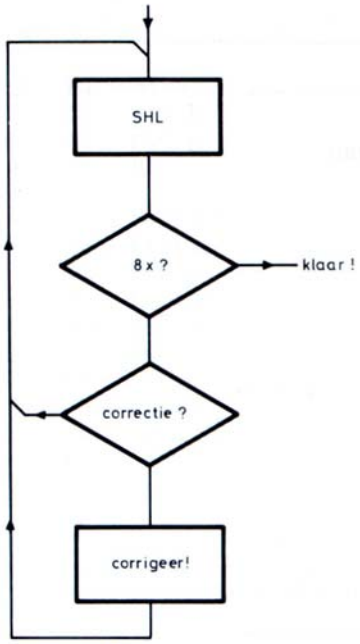
```
    0101    > 4
SHL: 1010   > 9
```

Als voor het schuiven 5 van het nibble wordt afgetrokken heeft dat hetzelfde effect als na het schuiven er 10 van af te trekken. Om bij het volgende nibble 1 op te tellen kan voor het schuiven het hoogste bitje 1 worden gemaakt. M.a.w. een waarde 8 bij dat nibble optellen. We moeten dus 5 aftrekken en 8 optellen. Het komt er dus op neer dat we 3 moeten optellen voor het schuiven bij die nibbles die een waarde groter dan 4 bevatten. Een andere correctie is noodzakelijk bij de overgang van een bitje van een nibble naar de volgende. Normaal zou de waarde van dat bitje verdubbeld worden; van waarde 8 naar waarde 16. Nu echter wordt de waarde bij zo'n overgang van 8 naar 10. We kunnen dit corrigeren door er 6 bij te tellen na het schuiven of 3 voor het schuiven. Als er echter van zo'n overgang van een nibble naar het volgende sprake is dan betekent dat dat het hoogste bit is van het voorgaande nibble een 1 was, en de waarde van dat nibble was dus zeker groter dan 4 waardoor de correctie al heeft plaatsgevonden tijdens de vorige correctieprocedure.
Conclusie: alle correcties worden automatisch uitgevoerd als we voor het schuiven 3 optellen bij die nibbles die een waarde groter dan 4 vertegenwoordigen.

Het schrijven van het programma.
We moeten nu een geschikte programma ontwikkelen zodat de processor de omzetting kan uitvoeren. Hiertoe moeten we het probleem eerst nauwkeurig beschrijven en dan een stroomdiagram maken.
De processor moet dus: schuiven, controleren of er al 8 keer geschoven is en controleren of correcties noodzakelijk zijn. Een mogelijk stroomdiagram toont ons afb. 9-4. Alvorens het programma te schrijven moeten een aantal afspraken worden gemaakt zoals: waar staat het hexadecimale getal dat omgezet moet worden en waar zetten we het resultaat. Laten we afspreken dat het hexadecimale getal is R(9)0 staat. Na omzetting staan de eenheden en 10-tallen in R(9)0 en de 100-tallen (max. 2) in R(9)1. De stack gebruiken we om de gegevens tijdelijk in op te slaan en wel als volgt.

Hexadecimaal getal		S
10-tallen	eenheden	S-1
	100-tallen	S-2

Verder spreken we af dat het programma een subroutine is die na afloop terugkeert naar het hoofdprogramma in R3. We kunnen dus R(3)0 gebruiken als "loop 8" teller mits we de oorspronkelijke inhoud bewaren op de stack. Alvorens naar het hoofdprogramma terug te keren herstellen we R(3)0.



Afb. 9-4: Principieel stroomdiagram van een Hex→BCD omzetting door schuiven en corrigeren.

Programma 9-2 toont ons de computer listing voor de Hex→BCD conversie.
Begonnen wordt met het vrijmaken van R(3)0 en het op de stack zetten van het Hex getal. Vervolgens wordt de byte voor de eenheden en 10-tallen nul gemaakt. De initialisering wordt besloten door de stackpointer op het hexgetal te zetten en de loop 8 teller te laden.
Vervolgens wordt het blok van 3 bytes een plaats naar rechts geschoven waarbij tevens R2 op de eenheden en 10-tallen byte wordt gezet. Nu wordt gecontroleerd of er al 8 maal geschoven is. Zoja; gereedmaken voor terugkeer. Hierna wordt gecontroleerd of de waarde van de 10-tallen nibble groter dan 4 is door van de gehele byte 50 af te trekken. De data-flag (DF) geeft uitsluitsel en zonodig wordt de nibble gecorrigeerd door bij de byte 30 op te tellen. Alvorens de waarde van de eenheden nibble te controleren is het noodzakelijk de inhoud van de 10-tallen nibble nul te maken door de byte te ''AND-en'' met $0F_{16}$. Zonodig volgt ook een correctie. De cyclus wordt $8\times$ herhaald waarna het omgezette getal van de stack in R9 wordt gezet. Hierna wordt R(3)0 hersteld en d.m.v. een SEP R3 instructie wordt terug gesprongen in het hoofdprogramma.

Zoals u ongetwijfeld zult hebben opgemerkt worden d.m.v. pointer R2 meerdere stackplaatsen geadresseerd. Bij het schrijven van programma's waarin dit gebeurt is het zeer belangrijk de stand van zo'n pointer in de loop van het programma goed bij te houden. Er worden gemakkelijk fouten gemaakt!

adres	byte	mnemonic	
00B5	D3	SEP R3	Return
start→ 6	83	GLO R3	
7	73	STXD	
8	89	GLO R9	(Hex getal)
9	73	STXD	
A	F8	LDI	
B	00		
C	52	STR R2	
D	12	INC R2	
E	F8	LDI	
F	08		Loop 8
C0	A3	PLO R3	
1	02	LDN R2	
2	FE	SHL	
3	73	STXD	
4	02	LDN R2	
5	7E	SHLC	SHL
6	73	STXD	
7	02	LDN R2	
8	7E	SHLC	
9	52	STR R2	
A	12	INC R2	
B	23	DEC R3	Loop 8 – 1
C	83	GLO R3	Loop = 0?
D	32	BZ	
E	E6		ja→
F	F0	LDX	nee
D0	FF	SMI	
1	50		high nibble > 4?
2	3B	BNF	
3	D8		nee→
4	F8	LDI	ja
5	30		
6	F4	ADD	corrigeert
7	52	STR R2	
8	F0	LDX	
9	FA	ANI	
A	0F		Low nibble > 4?
B	FF	SMI	
C	05		
D	3B	BNF	
E	E3		nee→
F	F8	LDI	ja
E0	03		corrigeert!

```
E1  F4  ADD
2   52  STR R2     ----------
3   12  INC R2
4   30  BR
5   C1             ----------
6   22  DEC R2
7   42  LDA R2
8   B9  PHI R9
9   42  LDA R2     prepare
A   A9  PLO R9     to
B   12  INC R2     leave!
C   F0  LDX
D   A3  PLO R3
E   30  BR
F   B5
```

Programma 9-2: *Dit is een subroutine die de byte in R(9)0 omzet in BCD. Het antwoord staat in R(9)0 met evt. 100-tallen in R(9)1. De subroutine start op M00B6.*

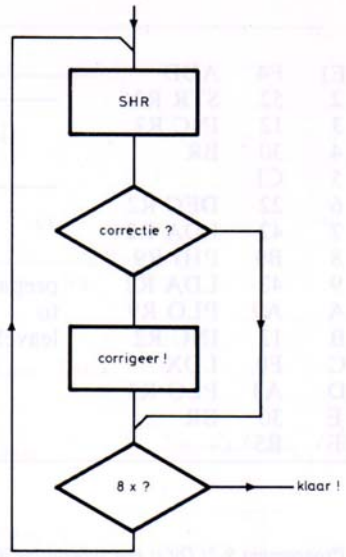
Decimaal naar Hexa-decimaal omzetting

Ook deze omzetting is d.m.v. schuiven en corrigeren te verwezenlijken. De principes zijn hetzelfde als bij de HEX → BCD omzetting. De verschillen zijn dat we nu niet niet naar links maar naar rechts schuiven. Een bitje dat op die manier van een hogere nibble naar een lagere verschuift verandert dus van waarde 10 naar waarde 8 i.p.v. 10 naar 5. Bij deze overgang moet dus een correctie worden uitgevoerd en wel door **na** schuiven 3 van de lager gelegen nibble af te trekken. Dit geldt ook voor de verschuiving van de 100-tallen naar de 10-tallen. Is er van zo'n overgang sprake dan moet dus na de verschuiving het hoogste bit van de laagste nibble 1 zijn; de waarde van die nibble is groter of gelijk aan 8. In dat geval moeten we corrigeren door er 3 van af te trekken. In afb. 9-5 is te zien hoe een stroomdiagram voor een BCD→Hex omzetting zou kunnen verlopen.

Een belangrijk verschil met het Hex → BCD diagram is dat we nu na de correctie controleren of de cyclus al 8× doorlopen is zodat de laatste schuif eventueel nog gecorrigeerd kan worden. We corrigeren dus na het schuiven in tegenstelling met de Hex → BCD omzetting waar we voor het schuiven corrigeren.

Gebruik van de code-conversie programma's

De Hex → BCD subroutine neemt de geheugenplaatsen 00B5 t/m 00EF in



Afb. 9-5: Principieel stroomdiagram van een BCD→Hex omzetting door schuiven en corrigeren. Let op de verschillen met de omzetting volgens afb. 9-4.

beslag. Het geheugengedeelte van 00F0 t/m 00FF is dus nog beschikbaar voor de stack. Hiervan is 1 plaats nodig voor opslag van R(3)0 en 3 plaatsen zijn nodig als de werkruimte tijdens de conversie. Het hoofdprogramma kan dus nog beschikken over 12_{10} stackplaatsen. Voor eenvoudige toepassingen is dat voldoende. De BCD → HEX subroutine loopt van 007B t/m 00B3. Als we gebruik maken van MOPS is het geheugengedeelte van 0025 t/m 007B beschikbaar voor het hoofdprogramma.

Programma 9-4 illustreert het gebruik van de code conversie subroutines. Begonnen wordt met het laden van 2 registers met de startadressen van de subroutines; R4 met 006C voor de BCD → Hex routine en R5 met 00B6 voor de Hex → BCD routine. Vervolgens wordt gewacht op EF4 waarna de byte op input binair in R(9)0 wordt gezet.

Nu wordt de Hex → BDC subroutine opgeroepen. Deze zet de byte om in zijn decimaal equivalent.

Direct hierop volgt de tweede subroutine Call nl. BCD → Hex. Tenslotte wordt het eindresultaat op het display gezet en wordt teruggesprongen naar het begin. Als u wilt weten of de routines echt werken kunt u instructie D4 op M 0034 vervangen door C4 (NOP) of instructie D5 op M 0033 door C4. Dat geeft wat meer duidelijkheid nietwaar!

adres	byte	mnemonic	
007B	D3	SEP R3	Return
start→ C	83	GLO R3	
D	73	STXD	
E	99	GHI R9	(BDC 100-tallen
F	73	STXD	
80	89	GLO R9	(BDC 10-tallen, 1-heden)
1	52	STR R2	
2	60	IRX	
3	F8	LDI	
4	08		Loop 8
5	A3	PLO R3	
6	F0	LDX	
7	F6	SHR	
8	73	STXD	
9	F0	LDX	SHL
A	76	SHRC	
B	73	STXD	
C	F0	LDX	
D	76	SHRC	
E	52	STR R2	
F	60	IRX	
90	F0	LDX	
1	FF	SMI	high nibble≧8?
2	80		
3	3B	BNF	
4	99		nee
5	F0	LDX	ja
6	FF	SMI	
7	30		corrigeer!
8	52	STR R2	
9	F0	LDX	
A	FA	ANI	
B	0F		
C	FF	SMI	low nibble≧8?
D	08		
E	3B	BNF	
F	A4		nee
A0	F0	LDX	ja
1	FF	SMI	
2	03		corrigeer!
3	52	STR R2	
4	23	DEC R3	
5	83	GLO R3	loop 8 = 0?
6	32	BZ	

```
7    AB                    ja
8    60   IRX              nee
9    30   BR
A    86
B    22   DEC R2
C    42   LDA R2
D    A9   PLO R9      Prepare
E    60   IRX         to leave
F    60   IRX
B0   F0   LDX
1    A3   PLO R3
2    30   BR
3    7B
```

Programma 9-3: *Deze subroutine zet het decimale getal met eenheden en 10-tallen in R(9)0 en 100-tallen en R(9)1 om in binair (Hex). Het resultaat staat in R(9)0. Als er geen 100-tallen zijn vergeet dan niet R(9)1 nul te maken.*
De subroutine start op M007C.

adres	byte	mnemonic	
0025	F8	LDI	
6	00		
7	B4	PHI R4	Laadt
8	B5	PHI R5	de
9	F8	LDI	programma
A	7C		tellers
B	A4	PLO R4	
C	F8	LDI	
D	B6		
E	A5	PLO R5	————————
F	3F	BN4	
30	2F		input byte!
1	6F	INP 7	
2	A9	PLO R9	————————
3	D5	Go Sub	(Hex → BCD)
4	D4	Go Sub	(BCD → Hex)
5	89	GLO R9	————————
6	52	STR R2	output byte!
7	67	OUT 7	
8	22	DEC R2	
9	37	B4	
A	39		
B	30	BR	opnieuw!
C	2F		————————

***Programma 9-4:** Testprogramma voor de Hex→BCD en BCD→Hex subroutines.*

HOOFDSTUK 10

Introductie in het programmeren 5

Gaande weg is het grootste deel van het instructie-repertoire van de Cosmac microprocessor besproken. Belangrijke zaken die nog aan de orde moeten komen zijn het z.g. ''Subroutine nesten'' en de interrupt techniek. Door van deze technieken gebruik te maken kunnen de mogelijkheden van de Cosmac ten volle worden benut. Vermeld dient nog te worden dat met de besproken subroutine-technieken het laatste woord nog niet is gesproken. De z.g. standard call en return techniek opent nieuwe perspectieven. Deze methode komt in Hfdst. 11 aan de orde.

MARK subroutine techniek

De opmerkelijkste eigenschap van de Cosmac is dat elk van de 16 universele registers programmateller kan zijn. Hierdoor is het op eenvoudige wijze mogelijk om met subroutines te werken. Een willekeurig universeel register wordt met het startadres van de subroutine geladen. Het hoofdprogramma kan dan deze subroutine oproepen door het betreffende register programmateller te maken. Na afloop geeft de subroutine de controle weer terug aan het hoofdprogramma door de oorspronkelijke programmateller weer te initialiseren. Het terugkeeradres is daarin bewaard gebleven. Deze subroutinetechniek is besproken in Hfdst. 9. Een probleem onstaat echter wanneer een subroutine een andere subroutine oproept; het z.g. subroutine nesten. De laatste instructie die de subroutine uitvoert is nl. een SEP-instructie waardoor de hoofdprogrammateller weer geïnitialiseerd wordt. De subroutine moet echter niet naar het hoofdprogramma terugkeren maar naar de eerste subroutine; de ''Caller''. Een fundamentele oplossing die het instructierepertoire biedt is de z.g. MARK subroutine techniek. De instructie ''MARK'' (79) heeft tot gevolg dat X en P via register T als één byte op de geheugenplaats worden gezet waar R2 naar wijst. Vervolgens wordt X gelijk aan P gemaakt en R2 wordt met 1 verlaagd.
De MARK instructies geeft ons de mogelijkheid X en P op de stack te zetten.
De instructie ''RETURN'' (70) heeft tot gevolg dat de byte op de geheugenplaats waar RX naar wijst wordt gecopiëerd in X en P. De bovenste nibble gaat naar X; de onderste naar P. Vervolgens wordt RX met 1 verhoogd. Instructie ''DISABLE'' heeft hetzelfde effect. Het verschil tussen beide instructies is dat bij RETURN bovendien de Interrupt Enable flipflop (IE) 1 wordt gemaakt. Bij DISABLE wordt IE 0 gemaakt; interrupts zijn dan niet mogelijk. Door gebruik te maken van MARK-en RETURN-instructies (of DISABLE) kunnen subroutines naar het programmateller register van de ''Caller'' terugkeren. De caller zet daartoe eerst d.m.v. de MARK-instructie zijn eigen X en P op de stack. Vervol-

gens wordt met de SEP-instructie de subroutine opgeroepen. Na afloop geeft de subroutine het commando terug aan de caller met een RETURN-instructie. Hierdoor worden X en P van de caller van de stack gehaald en gecopiëerd in de 4-bits X- en P-registers.
Van belang is verder dat bij de MARK-instructie R2 met 1 wordt verlaagd. Dit moet in de subroutine gecompenseerd worden door een Increment R2 instructie. Ook wordt X gelijk aan P gemaakt, dus alvorens terug te keren moet R2 tot X-register worden gemaakt. Vaak zal dit eerder in de subroutine al noodzakelijk zijn; voor de terugkeer hoeft dat uiteraard niet opnieuw te gebeuren.
Door de RETURN-instructie (of DISABLE) wordt RX met 1 verhoogd. Dit moet door de caller worden gecompenseerd met een Decrement R2 instructie onmiddellijk voor MARK.
Een voorbeeld:
Stel dat het startadres van een subroutine $00E0_H$ is. De taak van de subroutine is RA een plaats naar links schuiven.

	adres	opcode	mnemonic
start:	00E0	8A	GLO RA
	1	FE	SHL
	2	AA	PLO RA
	3	9A	GHI RA
	4	7E	SHLC
	5	BA	PHI RA

Na afloop moet de subroutine programmateller weer op startadres $00E0_H$ staan. De RET-instructie komt hier dus direct voor. Ook moet R2 nog met 1 verhoogd en "gesext" worden. De complete subroutine "SHL RA" ziet er als volgt uit.

	adres	opcode	mnemonic
	DF	70	RET
start:	00E0	8A	GLO RA
	1	FE	SHL
	2	AA	PLO RA
	3	9A	GHI RA
	4	7E	SHLC
	5	BA	PHI RA
	6	12	INC R2
	7	E2	SEX R2
	8	30	BR
	9	DF	

Aannemende dat R8 geladen is met het startadres van de subroutine geeft de volgende sequentie het commando over aan de subroutine.

opcode	mnemonic
22	DEC
79	MARK
D8	SEP R8

Als de subroutine zijn taak heeft vervuld worden door de instructie RETURN de oorspronkelijke X en P hersteld en de eerstvolgende instructie is die na SEP R8.

Stellen van de IE-flipflop

De interruptaansluiting van de Cosmac kan te allen tijde geactiveerd worden door input- of output schakelingen met als doel een directe reactie van de microprocessor te verkrijgen. Het interrupt heeft tot gevolg dat de processor met de uitvoering van het lopende programma stopt en overspringt naar een speciaal programma; ontworpen als antwoord op de interrupt aanvraag. Na afloop van dit interrupt programma wordt de uitvoering van het geïnterrumpeerde programma hervat. Door de interrupt-enable flipflop (IE) te resetten kan een interrupt niet meer optreden.

Er zijn 3 instructies met betrekking tot het X-register die van bijzonder nut kunnen zijn als X = P. Het zijn: de OUTPUT instructies (61-67), de RETURN-instructie (70) en de DISABLE instructie (71). Omdat elk van deze instructies het X-register met 1 verhoogd zal, als X = P, het R(P) / R(X) register met 1 verhoogd worden tijdens de fetchcyclus wanneer het fungeert als programma-teller en vervolgens nogmaals met 1 wanneer het fungeert als X-register. Het resultaat is dat de byte onmiddellijk na de instructie de operand is.

De volgende sequentie output de byte AD vooropgesteld dat P = 3.

opcode	mnemonic	commentaar
E3	SEX R3	SET X = 3
67	OUT 7	Output byte
AD	$AD	Immediate byte
	—	Next instruction

De RETURN- en DISABLE-instructie copiëren de byte op M(R)X in X en P. High nibble naar X; low nibble naar P. Bovendien wordt bij RETURN IE ''geset'' en bij DISABLE gereset. Door X gelijk aan P te maken en de immediate byte zo te kiezen dat de oorspronkelijke condities hersteld worden, kan IE geset of gereset worden, zonder verdere consequenties.

Stel: P = 3 en X = 2. De volgende sequentie reset IE en ''verbiedt'' interrupts.

	opcode	mnemonic
	E3	SEX R3
	71	DISABLE (0 → IE)
	23	immediate byte
		(2 → X; 3 → P)

Als inplaats van DISABLE, RETURN was gebruikt zou IE ''1'' geworden zijn (of gebleven).
Als vanuit de reset conditie de processor wordt gestart, dan geldt: P = 0, X = O, IE = 1.
Tijdens de eerste instructiecyclus wordt de interruptlijn nog niet getest. De volgende sequentie ''verbiedt'' interrupts tot de noodzakelijke voorzieningen zijn getroffen, waarna IE ''1'' gemaakt kan worden.

adres	opcode	mnemonic
0000	71	DIS
0001	00	$00

Interrupt techniek
De interruptaansluiting is beschikbaar aan pen 14 van het connectorprintje. D.m.v. een weerstand van 22 kΩ op de Cosmicosprint wordt de aansluiting op logisch ''1'' niveau gehouden. Een interrupt-aanvraag wordt gedaan door de interruptaansluiting ''0'' te maken. Of de aanvraag gehonoreerd wordt hangt af van de stand van IE. Als deze ''0'' is gebeurt er niets. Is de stand echter ''1'' dan gebeurt het volgende:
1e) de processor voltooit de lopende instructie-cyclus.
2e) X en P worden gecopieerd in T.
3e) X wordt 2 en P wordt 1.
4e) IE wordt 0.

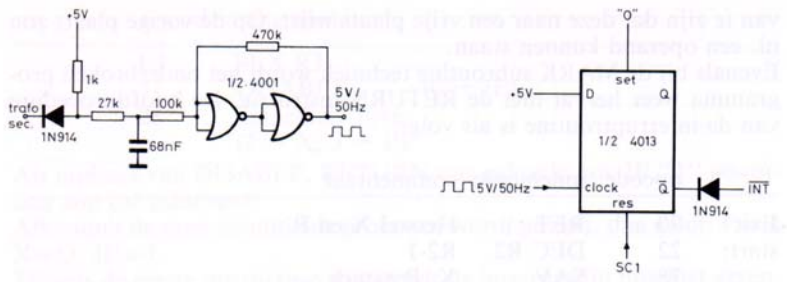
Bovenstaande handelingen geschieden in één machinecyclus; de z.g. S3 cyclus. Tijdens een S3 cyclus zijn de beide state code lijnen (SC0 en SC1) hoog. Na afloop van de S3 cyclus wordt de normale fetch/execute actie hervat. Nu echter met R1 als programmateller. Een interrupt kan op een willekeurig punt in het lopende programma optreden. Daarom is het noodzakelijk dat het interrupt programma alvorens met zijn eigenlijke taak te beginnen diè registers redt waarvan het zelf gebruik maakt. Na voltooiing van de taak worden deze registers hersteld, waarna de controle aan het oorspronkelijke programma wordt teruggegeven.
R1 moet altijd worden geïnitialiseerd met het startadres van de interruptroutine voordat aan een interruptaanvraag gehoor kan worden gegeven. Tijdens de initialisatie moet IE ''0'' zijn of de interruptlijn moet d.m.v. een schakelaar o.i.d. worden onderbroken.
De eerste byte die op de stack wordt gezet is T; de X en P van het onderbroken programma. Hiervoor dient de instructie SAVE (78). Alvorens dit te doen wordt eerst de stackpointer (R2) met 1 verlaagd om er zeker

van te zijn dat deze naar een vrije plaats wijst. Op de vorige plaats zou nl. een operand kunnen staan.
Evenals bij de MARK subroutine techniek wordt het onderbroken programma weer hervat met de RETURN instructie. De hoofdprocedure van de interruptroutine is als volgt:

	opcode	mnemonic	commentaar
Exit	70	RET	Herstel X en P
start:	22	DEC R2	R2-1
	78	SAV	X, P→stack
	22	DEC R2	R2 – 1
	73	STXD	D→stack
			Doe het werk!
	12	INC R2	R2 + 1
	42	LDA R2	herstel D
	30	BR Exit	

R1 dient geïnitialiseerd te zijn met adres "start".
Ter illustratie van de interrupt-techniek zullen we een programma behandelen, waardoor de Cosmicos gebruikt kan worden als klok. Vanzelfsprekend kan het programma naar eigen behoefte worden aangepast.

Cosmicos als klok
Voor het meten van tijd gaat men meestal uit van een bepaalde frequentie, die zodanig wordt gedeeld dat uiteindelijk een frequentie van 1 Hz overblijft. Deze frequentie kan worden opgewekt met b.v. een kristal waarmee een grote nauwkeurigheid kan worden bereikt. Een andere mogelijkheid is gebruik te maken van de 50 Hz lichtnetfrequentie. Deze wordt constant gecontroleerd en gecorrigeerd zodat het gemiddelde tijdsverloop over een periode van b.v. een maand nihil is. De netfrequentie is daardoor zeer goed bruikbaar voor tijdsbepalingen.
De secundaire spanning van de Cosmicos voedingstransformator bedraagt 7-12 V sinusvormig. Deze moet worden omgezet in een blokspanning van 50 Hz met een amplitude van 5 V. We maken hiertoe gebruik van de schakeling volgens afb. 10-1. Iedere puls moet door de processor worden geteld.
Om te voorkomen dat een puls wordt gemist maken we gebruik van de interrupt mogelijkheid. Bij iedere binnenkomende puls wordt het lopende programma onderbroken en wordt direct naar de interruptroutine gesprongen die de puls verder verwerkt. De tijdsduur van een puls is 10 ms; het interruptprogramma zal ruim binnen deze tijd doorlopen zijn.



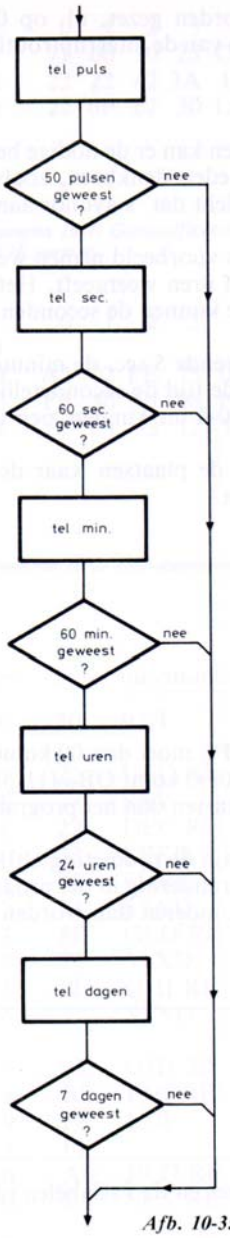
Afb. 10-1: Deze schakeling vormt de 50 Hz netfrequentie afkomstig van één van de twee secundaire trafo-aansluitingen om in een blokspanning met een amplitude van 5 V.

Afb. 10-2: D.m.v. de flipflop wordt de INT-lijn na iedere aanvraag gereset.

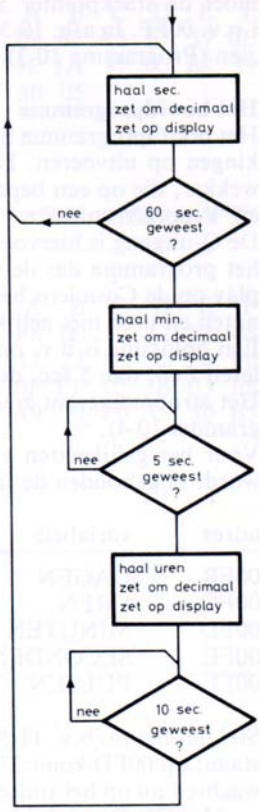
Om te voorkomen dat op hetzelfde interrupt meerdere malen wordt gereageerd, moet de interrupt ingang direct na erkenning gereset worden. Als resetsignaal gebruiken we SC1. Deze wordt immers hoog tijdens de S3 machinecyclus.
De 50 Hz netfrequentie wordt dus via de blokvormer van afb. 10-1 en de D-flipflop volgens afb. 10-2 op de interruptlijn aangesloten (Gnd, + 5V, SC1 en INT zijn beschikbaar op het connectorprintje).
Het totale klokprogramma bestaat uit 3 delen, t.w.: het interruptprogramma, het hoofdprogramma en de Hex → BCD subroutine. Daarnaast bevindt zich in het geheugen nog het mini-operating system: Mops. Voor de klokwerking is Mops niet noodzakelijk en deze ruimte kan eventueel voor andere taken worden gebruikt.
Zodra we de interruptschakeling hebben aangesloten reageert de processor op de interruptaanvraag. Dit wordt ondervangen door op de adressen 0000 en 0001 de codes 71,00 te zetten. Dit gaat natuurlijk niet zonder meer want Mops moet als geheel intact blijven. Op de adressen 0002 en 0003 volgt daarom een sprong over Mops heen, alwaar het nodige herstelwerk wordt uitgevoerd. Vervolgens wordt teruggesprongen naar M 0005 (Programma 10-1).
Procedure: Vul eerst m.b.v. Mops de geheugenplaatsen 0025 t/m 002B en vervolgens de plaatsen 0000 t/m 0003. Als tijdens de 2e faze een fout is gemaakt moet deze worden hersteld d.m.v. de ''load'' toestand.
De Hex → BCD subroutine is reeds besproken en is daarom als ''Hexdump'' opgenomen (Programma 10-2).

Interruptprogramma

Het interruptprogramma telt het aantal pulsen, het aantal seconden, minuten, uren en dagen. Al deze variabelen worden op vaste plaatsen in RAM opgeslagen, zodat ze door het hoofdprogramma kunnen worden bekeken en verwerkt.
We slaan de variabelen boven in de stack op; er zijn 5 variabelen dus



Afb. 10-3: Stroomdiagram van het eigenlijke interruptprogramma.



Afb. 10-4: Stroomdiagram van het hoofdprogramma dat de seconden, minuten en uren op het display zet.

moet de stackpointer 5 plaatsen omlaag worden gezet, nl. op 00FA i.p.v. 00FF. In afb. 10-3 is het stroomdiagram van de interruptroutine te zien (Programma 10-3).

Het hoofdprogramma

Het hoofdprogramma haalt de variabelen op en kan er de nodige bewerkingen op uitvoeren. Er zijn veel mogelijkheden denkbaar, zoals een wekker, die op een bepaalde tijd afloopt of licht dat 's avonds aangaat en 's morgens uit enz.

De Q-uitgang is hiervoor goed bruikbaar. Als voorbeeld nemen we hier het programma dat de seconden, minuten en uren weergeeft. Het display op de Cosmicos bestaat uit 2 cijfers. We kunnen de seconden, minuten en uren niet gelijktijdig laten zien.

Een oplossing is b.v. om om de 60 sec. gedurende 5 sec. de minuten te laten zien; dan 5 sec. de uren en de rest van de tijd de secondetelling.

Het stroomdiagram zou er dan als in afb. 10-4 uit kunnen zien (Programma 10-4).

Voor het gelijkzetten van de klok moet op de plaatsen waar de tijd wordt bijgehouden de juiste tijd worden gezet.

adres	variabele
00FB	DAGEN
00FC	UREN
00FD	MINUTEN
00FE	SECONDEN
00FF	PULSEN

Stel dat het nu b.v. 11.55 uur is. Op adres 00FE moet dan 00 komen te staan; op 00FD komt 37_H (55_D) en op adres 00FC komt OB_H (11_D). We wachten nu op het tijdsein van 11.55 uur en starten dan het programma op M 0075. De klok loopt nu precies gelijk.

Voor sommige toepassingen kan het handig zijn de tijdmeting altijd op 00.00 uur te laten beginnen. Met een kleine verandering in het programma is dit te realiseren. De volgende adressen moeten dan worden veranderd.

adres	oud	nieuw
007A	F8	73
B	FA	73
C	A2	73
D	C4	73
E	C4	73

Hierdoor wordt R2 automatisch op 00FA gezet en de variabelen plaatsen worden schoongemaakt.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	71	00	30	25	C4	A2	E2	6F	67	3F	09	6F	A3	67	37	0E
0010	22	22	42	3A	16	D3	F6	CF	7B	C4	3F	1A	37	1C	E3	39
0020	22	6F	67	30	1A	90	B2	B3	F8	FE	30	05				

Programma 10-1: *Gemodificeerde MOPS. Door de instructies 71,00 op M000 en M0001 wordt IE gereset.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00B0						D3	83	73	89	73	F8	00	52	12	F8	08
00C0	A3	02	FE	73	02	7E	73	02	7E	52	12	23	83	32	E6	F0
00D0	FF	50	3B	08	F8	30	F4	52	F0	FA	0F	FF	05	3B	E3	F8
00E0	03	F4	52	12	30	C1	22	42	B9	42	A9	12	F0	A3	30	B5

Programma 10-2: *Hex dump van de Hex→BCD subroutine.*

adres	byte	mnemonic	
002C	70	RET	stack → X, P
D	22	DEC R2	R2-1
E	78	SAV	X, P → stack
F	22	DEC R2	R2-1
30	73	STXD	D → stack
1	76	SHRC	
2	73	STXD	DF → stack
3	8E	GLO RE	
4	73	STXD	RE → stack
5	9E	GHI RE	
6	73	STXD	
7	92	GHI R2	
8	BE	PHI RE	00FF → RE
9	F8	LDI	
A	FF		RE = RX
B	AE	PLO RE	
C	EE	SEX RE	

```
D   F0   LDX
E   FC   ADI      tel
F   01            1 puls
40  5E   STR RE
                  ----------
1   FB   XRI      Vergelijk
2   32            met
3   3A   BNZ      50D
4   6A
                  ----------
5   73   STXD     00 → puls
                  ----------
6   F0   LDX
7   FC   ADI      tel
8   01            1 sec.
9   5E   STR RE
                  ----------
A   FB   XRI      Vergelijk
B   3C            met
C   3A   BNZ      60D
D   6A
                  ----------
E   73   STXD     00 → sec.
                  ----------
F   F0   LDX
50  FC   ADI      tel
1   01            1 min.
2   5E   STR RE
                  ----------
3   FB   XRI      Vergelijk
4   3C            met
5   3A   BNZ      60D
6   6A
                  ----------
7   73   STXD     00 → min.
                  ----------
8   F0   LDX
9   FC   ADI      tel
A   01            1 uur
B   5E   STR RE
                  ----------
```

C	FB	XRI	Vergelijk
D	18		met
E	3A	BNZ	24_D
F	6A		
60	73	STXD	00 → uren
1	F0	LDX	
2	FC	ADI	tel
3	01		1 dag
4	5E	STR RE	
5	FB	XRI	Vergelijk
6	07		met
7	3A	BNZ	7
8	6A		
9	5E	STR RE	00 → dagen
A	E2	SEX R2	
B	12	INC R2	
C	42	LDA R2	
D	BE	PHI RE	stack → RE
E	42	LDA R2	
F	AE	PLO RE	
70	42	LDA R2	
1	FE	SHL	stack → DF
2	42	LDA R2	stack → D
3	30	BR	
4	2C		

Programma 10-3: *De interrupt routine die de tijd bijhoudt.*

adres	opcode	mnemonic	
0075	93	GHI R3	
6	B1	PHI R1	
7	B9	PHI R9	
8	BA	PHI RA	
9	BD	PHI RD	
A	F8	LDI	
B	FA		00FA → R2
C	A2	PLO R2	
D	C4	NOP	
E	C4	NOP	
F	F8	LDI	
80	2D		002D → R1
1	A1	PLO R1	
2	F8	LDI	
3	FE		00FE → RA
4	AA	PLO RA	
5	F8	LDI	
6	B6		00B6 → RD
7	AD	PLO RD	
8	E3	SEX R3	
9	70	RET	enable interrupt
A	23	$23	(einde initialisering
B	0A	LDN RA	haal sec!
C	A9	PLO R9	
D	DD	SEP RD	Hex → BCD
E	89	GLO R9	
F	52	STR R2	
90	67	OUT 7	display!
1	22	DEC R2	
2	0A	LDN RA	Laat sec. zien
3	3A	BNZ	tot 1 min.
4	8B		voorbij is

```
 5    2A    DEC RA    haal min!
 6    0A    LDN RA
                      ----------
 7    A9    PLO R9
 8    DD    SEP RD    Hex → BCD
 9    89    GLO R9
                      ----------
 A    52    STR R2
 B    67    OUT 7     display!
 C    22    DEC R2
                      ----------
 D    1A    INC RA
 E    0A    LDN RA    wacht tot
 F    FB    XRI       sec. = 5!
A0    05
 1    3A    BNZ
 2    9E
                      ----------
 3    2A    DEC RA
 4    2A    DEC RA    haal uren!
 5    0A    LDN RA
                      ----------
 6    A9    PLO R9
 7    DD    SEP RD    Hex → BCD
 8    89    GLO R9
                      ----------
 9    52    STR R2
 A    67    OUT 7     display
 B    22    DEC R2
                      ----------
 C    1A    INC RA
 D    1A    INC RA    wacht tot
 E    0A    LDN RA    sec. = 10D!
 F    FB    XRI
B0    0A
 1    3A    BNZ
 2    AE
                      ----------
 3    30    BR
 4    8B              begin opnieuw!
```

Programma 10-4: *Het hoofdprogramma.*

HOOFDSTUK 11

Introductie in het programmeren 6

De Standaard Call en Returntechniek
De Standaard Call en Returntechniek (SCRT) is een flexibele, universele methode om met subroutines te werken.
Voordelen t.o.v. de SEP register techniek zijn:

1e) Er kan ongelimiteerd worden genest.
2e) Er is geen verwarring over programmatellers.
3e) Het doorgeven van parameters naar subroutines is duidelijk vastgelegd.

Nadelen zijn dat het oproepen en terugkeren van een subroutine meer tijd kost en dat er 3 registers voor moeten worden gereserveerd, afgezien van de stackpointer en de hoofdprogrammateller.
Register 2 is de stackpointer die naar een vrije plaats wijst. Voor iedere subroutine nesting zijn 2 stackplaatsen nodig.
Register 3 is de programmateller voor het hoofdprogramma en alle subroutines.
De SCRT maakt gebruik van 2 subroutines om de koppelingen tot stand te brengen. Eén subroutine: de Standaard Call routine loopt in R4, de ander: de Standaard Return routine loopt in R5. De Call- en Returnsubroutines worden opgeroepen met SEP R4, resp. SEP R5. R4 en R5 moeten aan het begin van het programma op het startadres van hun respectievelijke programma's worden gezet.
In tabel 11-1 staat de registertoewijzing voor de SCRT.

Het oproepen van een subroutine
Door instructie SEP R4 (D4) wordt de controle overgegeven aan de Standaard Call routine.
Deze gaat de volgende handelingen uitvoeren (bedenk dat R4 nu PC is).

1e) R6 wordt op de stack gezet.
2e) R3 wordt gecopieerd in R6.
3e) De 2 bytes volgend op SEP R4 worden in R3 geladen (tweemaal LDA R6).
4e) Het commando wordt teruggegeven aan R3 d.m.v. SEP R3.

Een subroutine wordt dus opgeroepen door D4 XXYY, waarbij XXYY het startadres van de subroutine is. Het resultaat van bovenstaande handelingen is dat:

a) De subroutine loopt in R3.

b) Register 6 wijst op de byte direct volgend op YY. Dit zal meestal het terugkeeradres zijn, maar het kan ook data voor de subroutine zijn.
c) Omdat R6 is gered is de stack met 2 bytes gegroeid.

Als de subroutine "in Line" data verwacht, moet de subroutine evenveel LDA R6 instructies uitvoeren als er databytes zijn. De databytes worden daardoor in het Data-register geladen voor gebruik door de subroutine en R6 wordt geïncrementeerd tot deze op het juiste terugkeeradres staat. De subroutine op zijn beurt kan ook weer een subroutine oproepen met D4 XXYY. Er kan ongelimiteerd worden genest.

Terugkeer van een subroutine
Als de subroutine zijn taak heeft vervuld, wordt het commando overgegeven aan de standaard Return subroutine d.m.v. SEP R5. Deze routine verricht de volgende handelingen.

1e) R6 wordt gecopieerd in R3.
2e) De "geredde" R6 wordt van de stack gehaald en weer in R6 geladen.
3e) Door een SEP R3 wordt naar de "Caller" teruggekeerd. Dit kan het hoofdprogramma zijn of een subroutine.

Programma 11-1 is de Listing van de standaard Call subroutine en Programma 11-2 van de standaard Return subroutine.

Overwegingen
De Standaard Call en Return techniek is een methode om met subroutines te werken, waarbij de koppelingen m.b.v. software tot stand komen. Dit betekent dat door gebruikmaking van aangepaste Call en Return subroutines bepaalde gewenste eigenschappen kunnen worden verkregen.
Een disassembler programma is hierdoor b.v. page relocatible gemaakt. M.a.w. het kan op iedere gewenste pagina beginnen zonder dat er één byte veranderd hoeft te worden. Voor een disassembler is dat een nuttige eigenschap. Dit soort foefjes is met een 1802 mogelijk.

Variabel programmeren op de stack
Een van de meest geavanceerde technieken is het variabel programmeren op de stack. Deze techniek maakt het mogelijk met een veranderlijke instructie te werken, terwijl het programma toch in ROM of EPROM staat. Stelt u zich een veel gebruikte subroutine voor, die afhankelijk van bepaalde condities een byte naar links of rechts moet schuiven. In R(B)1 zetten we b.v. F6 (SHR) of FE (SHL). We gaan nu op de stack een programma opbouwen.

DEC R2
LDI # D3 (is instructie SEP R3)
STXD
GHI RB (is variabele instructie)
STR R2

Na SEP R2 wordt de variabele instructie keurig uitgevoerd. Merk op dat er geen conflicten ontstaan met evt. interrupties.
Deze techniek is b.v. zeer geschikt om registers te redden en kan een opmerkelijk korter programma opleveren.

```
                ;STANDARD CALL
                ;
                ORG #C054
                ;
EXIT:   SEP R3
START:  SEX R2
        GHI R6
        STXD
        GLO R6
        STXD
        GHI R3
        PHI R6
        GLO R3
        PLO R6
        LDA R6
        PHI R3
        LDA R6
        PLO R3
        BR EXIT
```

Programma 11-1: *De standaard Call subroutine in 1802 assembly. Instructie ORG (Originate) geeft het eerste adres van het programma. Dit behoeft echter niet het startadres te zijn.*

```
                ;STANDARD RETURN
                ;
                ORG #C064
                ;
EXIT:           SEP R3
START:          GHI R6
                PHI R3
                GLO R6
                PLO R3
                SEX R2
                INC R2
                LDXA
                PLO R6
                LDX
                PHI R6
                BR EXIT
```

Programma 11-2: *De standaard Return subroutine.*

Register	Functie
R2	Stackpointer
R3	Programmateller
R4	Programmateller voor Call subroutine
R5	Programmateller voor Return subroutine
R6	Pointer naar terugkeeradres en ''in Line'' parameters.

Tabel 11-1: *Registeraanwijzing bij de standaard Call en Return techniek (SCRT).*