µP

werken met de microprocessor

Zelf programma's maken

D. M. de Boer

In ons vorige nummer maakte u reeds kennis met de KIM I microcomputer. Wellicht heeft u de Melodiant reeds nagebouwd, maar het leukste werk is natuurlijk zelf programma's maken, dus de KIM I precies te laten doen wat u wilt. Nu is het zelf programmeren voor iemand die dat nooit heeft gedaan, niet zo eenvoudig. Zeker in het begin werkt de aspirant programmeur zo onoverzichtelijk dat hij op het laatst niet meer weet welke instructie op welk adres staat. Het is daarom belangrijk om een programma overzichtelijk op te zetten, en alle veranderingen te noteren. In dit nummer zullen we aan de hand van een voorbeeld laten zien hoe dit wordt gerealiseerd. De exacte uitwerking laten we echter aan de lezer over, want ook hier geldt zeker: de praktijk is de beste leermeester!!!

Wat willen we?

Wanneer we een programma gaan schrijven moeten we eerst duidelijk formuleren wat de KIM I precies moet gaan doen.

Als voorbeeld zullen we stellen dat een ingetypt cijfer op het meest linkse display moet verschijnen, en vervolgens steeds een plaats naar rechts opschuift, totdat het cijfer uit beeld verdwenen is. Een duidelijke formulering zou er als volgt uit kunnen zien:

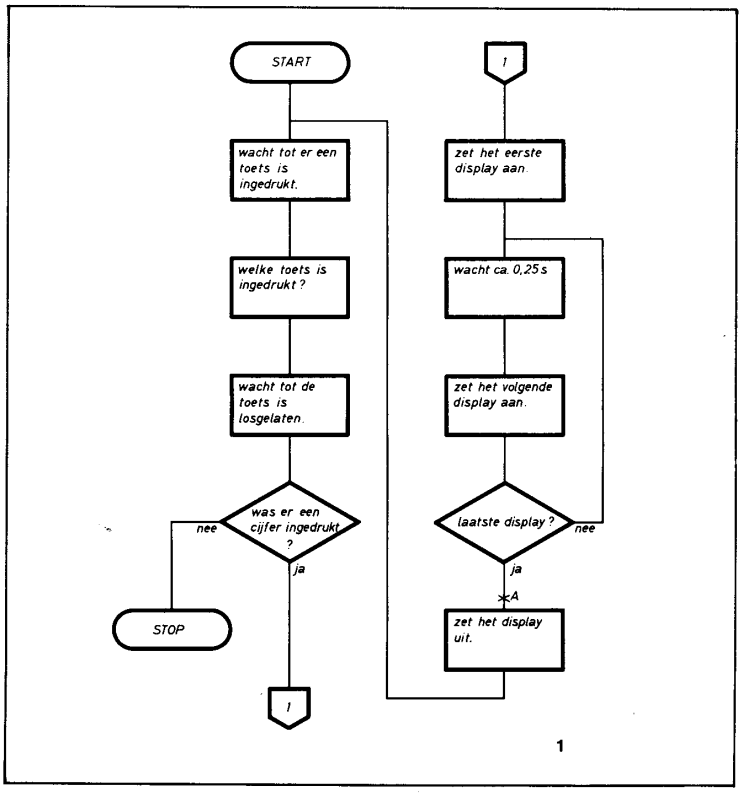
1 Het ingetypte cijfer mag pas bij het loslaten van de toets op het meest linkse display zichtbaar worden.
2 Vervolgens moet het cijfer met een vaste snelheid naar rechts schuiven.
3 Zolang het cijfer op één van de displays staat, mag het indrukken van een toets geen invloed hebben (behalve RS en ST), die altijd het programma onderbreken).
4 Wanneer de displays uit zijn, en er wordt een toets ingedrukt waar geen cijfer op staat, moet het programma na het loslaten van de toets worden onderbroken.

Aan de hand van deze definitie zullen we nu laten zien hoe een programma tot stand komt. Nog even voor de duidelijkheid: normaal bepaalt u natuurlijk zelf wat de KIM I moet doen. De hierboven staande definitie leidt tot een zinloos programma, en dient dan ook alleen als voorbeeld.

Hoe gaan we het doen?

Bij de volgende fase gaan we ons afvragen hoe het programma opgezet moet worden. We letten daarbij nog niet op de beschikbare instructies en registers. Het is de bedoeling om het totale programma te splitsen in kleine, maar toch nog steeds begrijpelijke losse opdrachten. De eerste opdracht zou b.v. kunnen luiden: 'Wacht tot er een toets wordt ingedrukt'. De tweede opdracht wordt dus pas uitgevoerd zodra er ook inderdaad een knop is ingedrukt.

Deze tweede opdracht is dan: 'Kijk welke toets is ingedrukt'. Het komt er op neer dat we alle handelingen op-



1

1 Zo zou de flow chart van het voorbeeld er uit kunnen zien.

schrijven die we zelf achtereenvolgens zouden verrichten om het gewenste effect te verkrijgen. Op deze manier wordt het hele programma opgesteld. De gebruikte opdrachten schrijven we in een flow chart. De flow chart van het voorbeeld zou er uit kunnen zien als getekend in afb. 1. De opdrachten staan in rechthoeken, terwijl beslissingen (voorwaardelijke sprongen) in een ruit worden geschreven. Met deze flow chart zien we nu in één oogopslag in welke volgorde verschillende programmadelen worden doorlopen. Alvorens verder te gaan, moeten we nu eerst controleren of de flow chart voldoet aan de gestelde eisen. We zien dat de displays alleen kunnen oplichten als de knop is losgelaten, dus dat is in orde. Ook wordt er gecontroleerd of de ingedrukte toets wel een cijfer was. Als dit niet zo is, wordt het programma gestopt. Door de opdracht 'wacht ca. 0,25 sec.' wordt bereikt dat elk display gedurende een duidelijk zichtbare tijd brandt. Als we de flow chart echter kritisch bekijken zien we dat het laatste display vrijwel onmiddellijk weer uitgezet wordt, zodat het niet zichtbaar zal gaan branden.
Om dit te verhelpen kunnen we op plaats A in de flow chart een opdracht 'wacht ca. 0,25 sec.' inlassen, zodat ook het laatste display lang genoeg blijft branden. Een andere oplossing is dat we net doen of er 7 displays zijn i.p.v. 6. Het 6e display wordt nu 'normaal' behandeld, terwijl display 7 (welke niet bestaat) te kort brandt. Op deze manier zullen alle displays gedurende dezelfde tijdsduur branden.

Het uitwerken van de flow chart

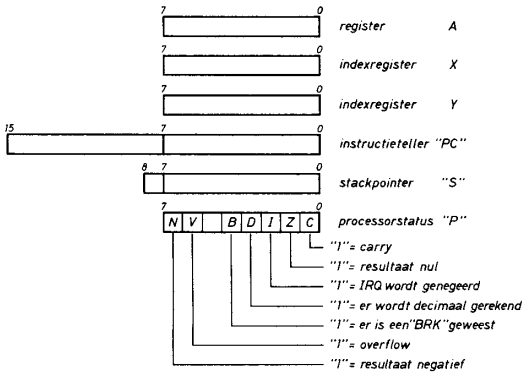
We hebben nu de flow chart in grote lijnen opgezet, in voor ons begrijpelijke losse opdrachten. De computer begrijpt er echter nog niets van. Daarom moeten we in deze fase de flow chart uitwerken in nog kleinere en voor de computer begrijpelijke opdrachten. We kunnen hierbij (waar het mogelijk is) gebruik maken van subroutines welke al aanwezig zijn in het ROM. Dit is meestal het geval voor opdrachten die ook door het monitorprogramma gebruikt worden. Als voorbeeld kunnen we de tweede opdracht nemen: 'Welke toets is ingedrukt'? Het zal duidelijk zijn dat het monitorprogramma (die zorgt dat ingetypte getallen op de juiste lokaties terecht komen) ook meerdere malen moet bepalen welke toets is ingedrukt. Na bestudering van het monitorprogramma (te vinden in het meegeleverde KIM I user manual) blijkt dan ook dat voor deze opdracht een kant en klare subroutine staat op adres 1F6A.

Voor we de flow chart verder kunnen uitwerken moeten we weten welke instructies de microprocessor kent, en hoe de registers in de microprocessor zijn georganiseerd. De instructieset bestaat uit 56 instructies welke reeds werden besproken in het vorige nummer (RB aug. '77 pag. 288). De 6502 microprocessor heeft 3 vrij te gebruiken interne registers van elk 8 bits. Dit zijn register A (accumulator) en de indexregisters X en Y. Verder bevinden zich in de microprocessor natuurlijk nog de instructieteller (16 bits), de stackpointer (8 bits) en het statusregister (7 bits). Deze registers staan nog eens overzichtelijk weergegeven in afb. 2.
Zoals al gezegd is het de bedoeling elk blokje van de nu gevormde flow chart te splitsen in een aantal blokjes met zeer specifieke en voor de microprocessor begrijpelijke opdrachten. Op deze manier ontstaat dus een tweede flow chart. Het aantal blokjes waarin elk blokje van afb. 1 moet worden verdeeld, kan sterk variëren.
Als voorbeeld zullen we één blokje uitwerken. De andere blokjes kunt u dan zelf proberen uit te werken. We nemen hiervoor 'wacht ca. 0,25 sec.'. Hoe krijgen we dit nu voor elkaar? Het is niet mogelijk de microprocessor stil te zetten, hij blijft constant 200.000 . . . 500.000 instructies per sec. uitvoeren! De enige oplossing om de microprocessor schijnbaar te laten wachten is door hem een klein stukje programma een zeer groot aantal malen te laten doorlopen. Dit wordt dan een loop genoemd. Afb. 3 geeft de flow chart van zo'n loop.

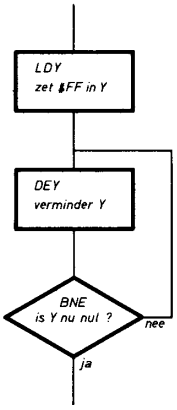
We beginnen indexregister Y te laden met een bepaalde waarde. Vervolgens verminderen we de inhoud van dit register steeds met 1 totdat het resultaat '0' is. Elke keer dat de loop doorlopen wordt is de microprocessor 5 µs zoet. (2 µs voor het verminderen en 3 µs voor het terugspringen). De maximum inhoud van register Y is $ FF, decimaal geschreven 256. De totaal haalbare vertraging is dus 256 x 5 µs = 1280 µs. Grotere tijden kunnen gemaakt worden door 2 of 3 loops in elkaar te zetten. Met 2 loops kunnen we tijden krijgen tot ca. 0,3 seconde, voor ons voorbeeld voldoende. De uitwerking van 'wacht ca. 0,25 sec.' wordt nu zoals in afb. 4 staat aangegeven. De flow chart is hier tot op instructieniveau teruggebracht, d.w.z. dat elk blokje van de flow chart één en niet meer dan één instructie voorstelt. Bij het gebruik van loops moeten we er aan denken dat de oorspronkelijke inhoud van de indexregisters verloren gaat. Bij de uitwerking (afb. 4) zijn we er van uitgegaan dat indexregister X reeds in gebruik was. De waarde van dit register wordt nu tijdelijk in het geheugen opgeslagen.
Pas als de hele flow chart (via één of meer tussenstappen) op het instructieniveau is, gaan we het programma schrijven.
Als voorbeeld zullen wij ons hier beperken tot het stukje flow chart van afb. 4. Om te beginnen schrijven we op alle punten waar flowlijnen bij elkaar komen, een naam (label). In afb. 4 zijn dit 'loop 1' en 'loop 2'. Dit wordt gedaan om later in het programma deze punten gemakkelijk terug te kunnen vinden, en om bij de sprongen aan te kunnen geven waarheen moet worden gesprongen. Ook wanneer gegevens tijdelijk ergens opgeslagen moeten worden geven we een geheugenplaats aan met een naam, dus niet met een adres! In de praktijk blijkt dit het handigst te werken, omdat nu achteraf altijd nog makkelijk geschoven kan worden met de geheugenplaatsen. We hoeven immers pas op het laatste moment een adres te definiëren.

Het schrijven van het programma

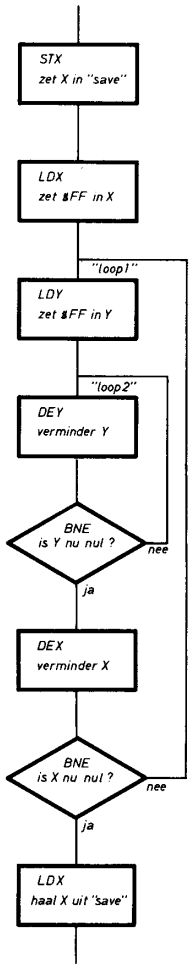
Wanneer we de hele flow chart op instructie-niveau gebracht hebben kunnen we het programma gaan schrijven. Het gemakkelijkst werken we wanneer we een vel papier in 8 kolommen verdelen, zoals aangegeven in afb. 5. De eerste 4 kolommen laten we voorlopig leeg. In de 5e kolom vullen we eventueel de naam van de betreffende geheugenplaats in. In de 6e kolom komt de OP-code te staan, in de 7e kolom de operand. De 8e kolom gebruiken we om in het kort uitleg over het programma te geven.
Deze programma-uitleg zal goede diensten bewijzen als u na een maand het programma nog eens wilt bekijken. In de kolom 'operand' staat steeds de naam van de geheugenplaats. Alleen bij de immediate addressing staat hier de operand zelf. Het # teken geeft aan dat de immediate addressing gebruikt is, terwijl $ gebruikt wordt om aan te duiden dat het getal hexa-decimaal geschreven is.
Wanneer we het hele programma op deze manier hebben geschreven gaan we de eerste 4 kolommen invullen. Het



2



3



4

adres	instructie	naam	OPcode	operand	verklaring
			STX	save	zet X zolang in het geheugen
			LDX	#$ FF	} inhoud bepaalt
		loop 1	LDY	#$ FF	} totale vertraging
		loop 2	DEY		
			BNE	loop 2	einde tweede loop
			DEX		
			BNE	loop 1	einde eerste loop
			LDX	save	haal X uit het geheugen

5

adres	instructie	naam	OPcode	operand	verklaring
0200	8E 23 01		STX	save	zet X zolang in het geheugen
0203	A2 FF		LDX	#$ FF	} inhoud bepaalt
0205	A0 FF	loop 1	LDY	#$ FF	} totale vertraging
0207	88	loop 2	DEY		
0208	D0 FD		BNE	loop 2	einde tweede loop
020A	CA		DEX		
020B	D0 F8		BNE	loop 1	einde eerste loop
020D	AE 23 01		LDX	save	haal X uit het geheugen
0210					

6

2 De interne registers van de 6502 microprocessor.
3 Een enkele loop, deze flow chart is tot het instructieniveau terug gebracht.
4 Een dubbele loop, hiermee zijn tijden tot ca. 0,3 sec. te realiseren. Ook deze flow chart is tot instructieniveau teruggebracht.
5 Het programma van het stukje flow chart uit afb. 4. De eerste 4 kolommen blijven voorlopig leeg.
6 Als het hele programma klaar is gaan we de machinecodes en de adressen bijschrijven. De instructie kan bestaan uit 1, 2 of 3 bytes.

best kunnen we beginnen met de 2e... 4e kolom. In deze kolom schrijven we de machinecodes welke bij de instructies behoren. De eerste instructie (afb. 5) luidt: 'STX save'. De inhoud van indexregister X moet zolang op een geheugenlokatie met de naam 'save' gezet worden. We hebben de keus uit zero page addressing en absolute addressing. In het voorbeeld gaan we er van uit dat geheugenlokatie 'save' het adres $ 0123 gekregen heeft. We moeten dus absolute addressing toepassen. De instructie wordt als volgt gecodeerd: Eerst zoeken we de code voor STX op. Voor deze code vinden we: 8E. Dit is de OP-code, deze komt op het eerste byte te staan (afb. 6). Op het 2e en 3e byte zetten we het adres van de operand. Het kan voorkomen dat dit adres nog niet bekend is, in dat geval laten we deze plaatsen gewoon nog even open. Let er wel op dat de laatste 2 cijfers v.h. adres op het 2e byte komen te staan, en de eerste 2 cijfers op het 3e byte! Op deze manier vullen we alle codes in. Voor de instructies die maar 1 of 2 bytes lang zijn laten we gewoon de laatste kolommen leeg. (afb. 6). Het invullen van de adressen zal geen moeilijkheden geven, we kunnen in principe op elk adres beginnen, mits het programma binnen de beschikbare ruimte blijft. We kunnen alle adressen tussen 0000 en 03FF gebruiken, alleen de adressen 00EF... 00FF zijn gereserveerd voor het monitorprogramma, en de hoogste adressen op pagina 1, welke zijn gereserveerd voor de stack. (Een ruimte van 01E0... 01FF is meestal voldoende.)

Het berekenen van het relatieve adres bij de branch-instructies

Een laatste moeilijkheid bij het coderen van het programma is het bepalen van de relatieve adressen bij de branch-instructies. Op adres 020B (afb. 6) vinden we zo'n branchinstructie. In dit geval moeten we terugspringen naar 'loop 1'. De microprocessor bepaalt het nieuwe adres door het relatieve adres op te tellen bij de stand van de instructieteller. Als de microprocessor deze handeling uitvoert, is de instructieteller al opgehoogd. In ons geval staat de instructieteller dus op 020D. Na de sprong moeten we uitkomen bij 'loop 1' op adres 0205. We moeten dus een sprong maken van 0205 - 020D = -08 ($D=13, 5-13=-8). Wanneer de adressen nog niet bekend zijn kunnen we ook gewoon het aantal bytes tellen dat tussen beide instructies ligt. Het -teken duidt aan dat we terug moeten springen. Op het tweede byte van de branch-instructie moeten we dus -8 invullen. Hiervoor moeten we eerst het two's complement uitrekenen. Dit gaat als volgt:
We schrijven eerst het getal binair op, we inverteren het getal en tot slot tellen we '1' op.
Voorbeeld:
$08 → 00001000
inverteren 11110111
1 optellen 11111000 → $F8
Het hexadecimale getal -08 wordt dus in two's complement $F8 (het $-teken staat weer voor hexadecimaal). Voor kleine negatieve sprongen (kleiner dan $F) werkt men vaak sneller, door gewoon de verlangde sprong van 16 (decimaal) af te trekken. Voorbeeld: Een negatieve sprong van 8, 16-8=8. Het two's complement wordt $F8. Een sprong van 3, 16-3=13 of D hexadecimaal. Het two's complement wordt $FD.

Het sturen van de displays

Wanneer u wilt proberen het aangehaalde voorbeeld uit te werken, moet u nog weten hoe de displays moeten worden gestuurd. Afb. 7 geeft het aansluitschema van de displays en het toetsenbord. In de normale toestand tasten de ingangen PA 0... 6 het toetsenbord af. Hierbij worden de uitgangen 0... 2 van IC1 beurtelings laag gemaakt (dit gebeurt allemaal door het monitorprogramma). Wanneer we de displays willen sturen, moeten we van de ingangen PA 0... 6 uitgangen maken. Dit gebeurt zodra we in het richtingsregister PA DD2 (adres 1741) voor de bits 0... 6 een '1' schrijven ($ 7F). Verder moeten we m.b.v. PB 1... 4 en IC 1 één van de zes displays selecteren. Onderstaande tabel laat zien welk getal in PB D2 (adres 1742) geschreven moet worden om een bepaald display te kiezen.

Display no.	getal in PB D2
1	08
2	0A
3	0C
4	0E
5	10
6	12

Wanneer het display gekozen is, en de aansluitingen PA Ø... 6 zijn als uitgang geprogrammeerd, kunnen we op PA D2 (adres 1740) de displaycode zetten. In afb. 8 zien we bij elk segmentje een getal. Wanneer we de getallen van de segmentjes die moeten branden optellen, krijgen we de displaycode.
De code om het getal 5 op het display te krijgen wordt dus 1+4+8+20+40=6D. Wanneer we klaar zijn met het displayen, moeten we niet vergeten om de nu als uitgang geprogrammeerde aansluitingen PA Ø... 1 weer als ingang te definiëren ($00 op PADD2 adres 1741).

Handige subroutines

Zoals al eerder gezegd maakt het monitorprogramma regelmatig gebruik van subroutines welke reeds in het ROM staan. Ook in zelfgeschreven programma's kunnen we van deze subroutines gebruik maken. Hieronder volgt een opsomming van interessante subroutines.

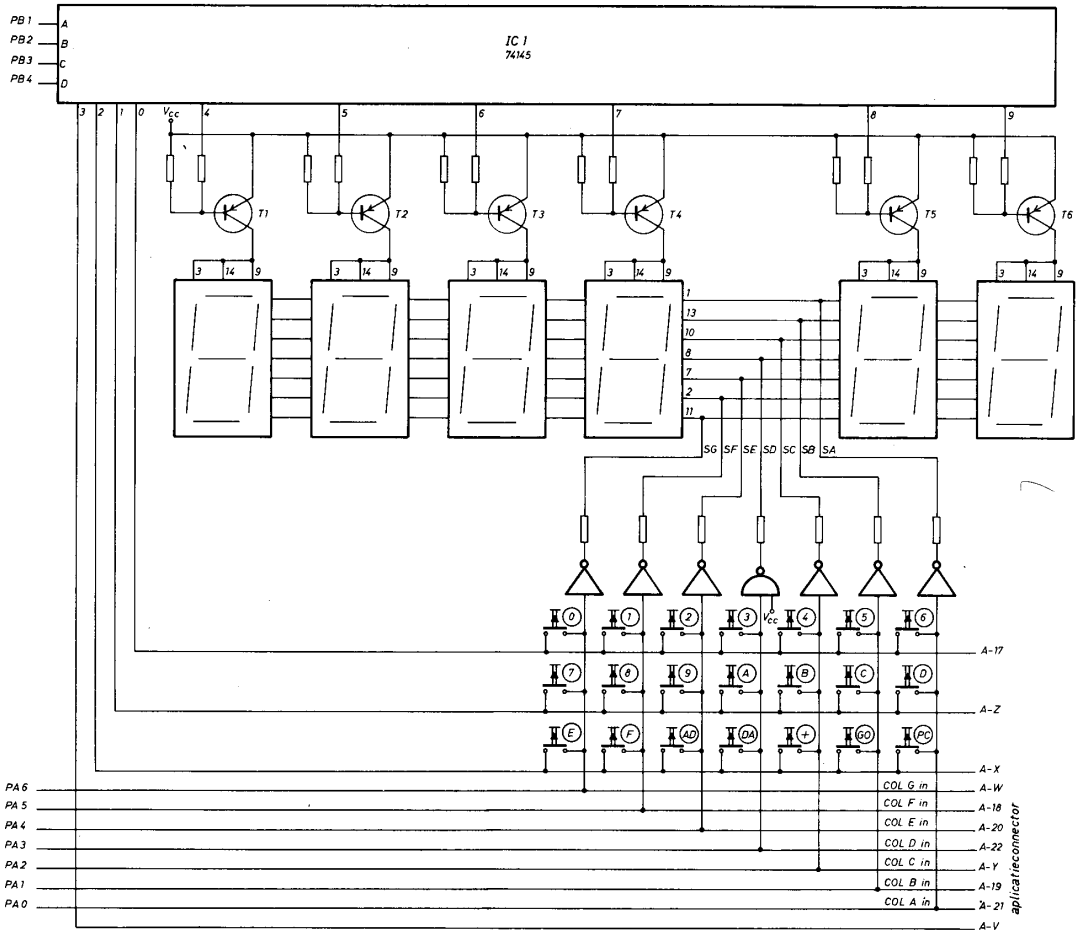
INIT 1 adres IE8C
Deze subroutine zet alle richtingsregisters in de juiste stand. De decimaal mode flag wordt op '0' gezet, de interrupt flag op '1'. De inhoud van indexregister X gaat verloren.

AK adres IEFE
Deze subroutine bepaalt of er een toets is ingedrukt. De inhoud van register A wordt 0 als er geen toets is ingedrukt, en ongelijk 0 als er wel een toets is ingedrukt. De inhoud van register A en de indexregisters X en Y gaan verloren.

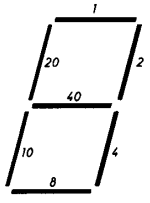
GETKEY adres 1F6A
Deze subroutine bepaalt welke toets is ingedrukt. In register A komt het ingetypte getal te staan. De inhoud van register A en de indexregister X en Y gaan verloren.

SCANDS adres IFIF
Deze subroutine zorgt ervoor dat de inhoud van de adressen 00FB, 00FA en 00F9 op de display verschijnen. Het is alleen mogelijk hexadecimale cijfers te laten zien, er branden altijd 6 displays tegelijk. Ook de richtingsregisters worden goed gezet. De inhoud van de indexregisters X en Y gaan, evenals de inhoud van register A verloren. Aan het eind van deze subroutine wordt naar AK gesprongen, zodat gelijk bepaald is of er een toets is ingedrukt.

CONVD adres IF48
Deze subroutine laat één van de 6 displays branden, en kan daardoor gebruikt worden in het aangehaalde voorbeeld. De richtingsregisters worden door deze subroutine **niet** goed gezet. Het getal in register A wordt omgezet in een displaycode. Het getal in index-



7



8

7 Zo zijn de displays en het toetsenbord op de uitgangspoorten PAD2 (1740) en PBD2 (1742) aangesloten.

8 De displaycode voor elke willekeurige combinatie van segmentjes wordt verkregen door de bijgetekende getallen op te tellen.

register X bepaalt welke display brandt. (Zie tabel bij 'het sturen van de displays'). De inhoud van indexregister X wordt automatisch met 2 opgehoogd, zodat dit register nu het volgende display aanwijst. De inhoud van register A gaat verloren.

Nakijken van het programma

U heeft nu voldoende informatie om zelf het aangehaalde voorbeeld uit te werken. Wanneer het hele programma geschreven is, kunt u het in de computer intypen.

Het is nu verstandig eerst het programma in 'single step' te doorlopen. Hiervoor moeten we op adres 17FA '00' zetten, en op 17FB 'IC'. Vervolgens zetten we het schuifschakelaartje op SST. Het is vooral belangrijk bij de branch-instructies te controleren of de sprongen goed zijn berekend. Tussen de stappen kunnen we de inhoud van de diverse registers controleren door de volgende adressen in te voeren:

00F1 statusregister
00F2 stack pointer
00F3 register A
00F4 indexregister Y
00F5 indexregister X

Eventueel kunnen we ook de inhoud van deze registers wijzigen, wat vooral bij loops handig kan zijn. Wanneer we weer verder willen gaan met het programma, moeten we eerst op PC drukken, zodat het programma weer vervolgd wordt vanaf het punt waar we gebleven waren.

Wanneer we ergens midden in het programma de inhoud van de diverse registers en/of geheugenplaatsen willen controleren is er een snellere methode. We kunnen nl. ook stops inlassen. Hiervoor moeten we eerst weer het 'stop adres' invoeren, d.i. het adres waarheen gesprongen moet worden wanneer er een stop staat. Meestal zal dit adres IC00 zijn (monitor programma). We voeren weer '00' in op adres 17FE en 'IC' op adres 17FF. Op de plaats waar we de registers en/of geheugens willen controleren vervangen we een bepaalde instructie zolang voor BRK (code 00). Het programma zal nu normaal gevolgd worden, tot het BRK-commando. Op dit punt springt de microprocessor naar het monitor-programma, zodat we de verschillende registers kunnen controleren.

Enkele veel voorkomende fouten

Tot slot zullen we nog een paar veel voorkomende fouten geven, zodat u op deze fouten bedacht kunt zijn.

- Foutief berekende branch-instructies. Door een verkeerde sprong kan een operand als OP-code worden gelezen. Hierdoor ontstaat een heel ander (en vaak erg vreemd) programma. Door een dergelijke fout kan het programma zich zelf overschrijven, zodat het hele programma weer gecontroleerd moet worden.
- Vaak worden registers dubbel gebruikt. Bij gebruik van subroutines gaat soms de inhoud van een van de registers verloren. Bij loops verandert de inhoud van de indexregisters. Denk er dus aan dat de inhoud van verschillende belangrijke registers in deze gevallen tijdelijk in het geheugen moeten worden gezet.
- Denk er aan dat u met zowel Push als Pull – instructies geeft. 'Push' heeft tot gevolg dat de stack pointer met 1 wordt verminderd, 'Pull' zorgt ervoor dat de stack pointer weer wordt opgehoogd. Wanneer één van deze instructies alleen in een loop voorkomt, wordt de stack pointer voortdurend opgehoogd (of verminderd) waardoor alle informatie op pagina 01 verloren kan gaan.

De KIM 1 tegen gereduceerde prijs voor abonnees van RB

De KIM 1 is normaal in de handel voor f 1110,– incl. BTW. Meegeleverd worden een stel van drie engelstalige handboeken en een printsteker. Niet inbegrepen is de voeding, waarvoor overigens ieder apparaat kan worden gebruikt dat 5V bij 1,2A èn 12V bij 0,1A kan leveren. Een schema van een eenvoudige voeding is te vinden in een van de drie handboeken.

Ten behoeve van de abonnees van RB heeft de uitgeverij De Muiderkring een partij KIM 1 microcomputers in voorraad genomen. Deze kunnen geleverd worden voor de gereduceerde stuks-prijs van f 998,– incl. BTW, excl. verzendkosten ad. f 8,15. Ook lezers die nog geen abonnee zijn kunnen van dit aanbod profiteren door zich nu als zodanig op te geven.

Bestelling van een KIM 1 en/of een abonnement op RB kan geschieden door inzending van onderstaande 'bon' of (als u het blad niet wilt beschadigen) van een briefje waarin de van toepassing zijnde gegevens zijn vermeld. Met de betaling kunt u wachten tot u een acceptgirokaart ontvangt. De bestellingen worden in volgorde van binnenkomst behandeld.

Via De Muiderkring bestelde microcomputers worden door de importeur volgens de gebruikelijke voorwaarden gegarandeerd.

BON

Uitknippen of overschrijven op ander papier en ingevuld opzenden in open, ongefrankeerde envelop aan: Redactie RB, machtiging 224, 1400VB Bussum.

☐ ik ben reeds RB-abonnee
☐ ik geef mij op als RB-abonnee
(nrs. okt., nov., dec. '77 en hele jaar '78 voor slechts f 40,–)
☐ ik bestel een KIM 1 microcomputer voor de gereduceerde abonnee-prijs van f 998,– plus f 8,15 verzendkosten

Het totaalbedrag ad. f ..... zal ik direct na ontvangst van uw acceptgirokaart overmaken.

naam: _____

adres: _____

woonplaats: _____

handtekening: _____