D. M. DE BOER

De Microprocessor

Met de ontwikkeling van de microprocessor of kortweg μP is in de techniek een revolutionaire ontwikkeling op gang gekomen. Deze bouwsteen, in de vorm van een IC, bevat een zeer complexe digitale schakeling met een zeer groot aantal poorten. De opzet is echter zo algemeen dat de microprocessor een groot toepassingsgebied heeft. De eerste microprocessors werden rond 1971 ontwikkeld door Intel en waren toen nog erg duur. De laatste tijd echter zijn de prijzen aanzienlijk gedaald, en ze zullen nog verder dalen, zodat de microprocessor ook voor de amateur interessant gaat worden.

Wat is een microprocessor?
Voordat we kunnen uitleggen wat je met een microprocessor kunt doen, is het belangrijk om eerst te begrijpen wat een microprocessor precies is.

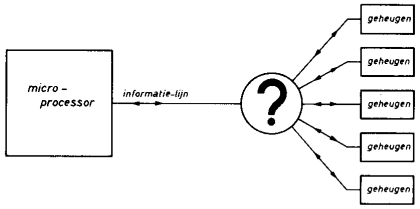
Zoals gezegd bevat de microprocessor een zeer complexe digitale schakeling. Eenvoudig voorgesteld kan men een microprocessor zien als een soort centrale verwerkingseenheid, die digitale informatie naar binnen haalt, de informatie bewerkt, en vervolgens het resultaat weer aan de uitgang afgeeft. Natuurlijk moet die informatie ergens vandaan komen, en na bewerking moet het weer ergens heen kunnen. Het zal dan ook wel duidelijk zijn, dat de microprocessor niet alléén kan functioneren, er zijn nog andere componenten nodig, waar informatie aan kan worden ontrokken en waar de uitgaande informatie (het resultaat) naar teruggestuurd kan worden. Meestal zullen die componenten bestaan uit geheugens, waarin de informatie voor langere of kortere tijd kan worden opgeslagen. Ook kan informatie worden uitgewisseld met ingangs- en uitgangsschakelingen, die op hun beurt weer contact met de buitenwereld hebben. Maar hierover later meer.

Tot nu toe hebben we dus gezien dat de microprocessor informatie naar binnen haalt van b.v. een bepaald geheugen en die informatie, al dan niet na bewerking, weer naar hetzelfde of naar een ander geheugen terugstuurt. Zie hiervoor afbeelding 1. Uit deze afbeelding blijkt dat het voor de microprocessor niet voldoende is om alleen informatie uit te wisselen, hij moet ook aangeven waar de informatie vandaan moet komen, of waar de informatie heen moet gaan. Om dit te bereiken hebben alle componenten waar mogelijk informatie mee kan worden uitgewisseld, (dus niet alleen de geheugens, maar ook de in- en uitgangen) een nummer gekregen. Dit wordt, net als bij huizen in een straat, een adres genoemd. Omdat de microprocessor uitsluitend met binaire getallen werkt, (dus getallen geschreven in het tweetallig stelsel) is het 't handigst als we ook de adressen meteen binair opschrijven. Als we bij ons voorbeeld blijven hebben we dus 5 adressen. (In werkelijkheid loopt het aantal adressen, afhankelijk van de grootte van het systeem tussen de honderden en tienduizenden!!) Deze adressen nummeren we met 000, 001, 010, 011 en 100. Uit de microprocessor komen nu ook drie adreslijnen, ruim voldoende om de 5 adressen te coderen. Het systeem zoals het er nu uitziet is getekend in afbeelding 2. De microprocessor geeft via zijn adreslijnen aan, met welk onderdeel informatie moet worden uitgewisseld. De elektronische schakelaar kiest het juiste adres, en de microprocessor kan met de informatieoverdracht beginnen.

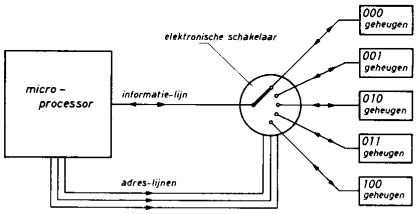
Een zwak punt in dit systeem is de elektronische schakelaar. Vooral als het aantal adressen toeneemt, neemt de omvang van deze schakeling schrikbarende vormen aan. Daarom wordt in dit soort systemen vaak gewerkt met een schakeling volgens afbeelding 3. In deze schakeling krijgen alle onderdelen (geheugens of in- en uitgangen) de informatie- èn de adreslijnen. Elk onderdeel 'weet' zijn eigen adres, en komt pas in actie zodra het zijn eigen adres ziet.

Op deze manier wordt de elektronische schakelaar vermeden. (In wezen wordt die schakelaar verdeeld over de diverse geheugenplaatsen.) Een ander groot voordeel is, dat men met dit systeem het aantal adressen gemakkelijk kan uitbreiden. Een extra geheugen, of een extra in- of uitgang behoeft slechts met de informatielijn en met de adreslijnen verbonden te worden.
Natuurlijk moet een extra onderdeel meteen een nog niet gebruikt adres meekrijgen. Dit gebeurt door de adreslijnen al dan niet geïnverteerd op het onderdeel aan te sluiten. Een en ander zal nog nader worden besproken. In het voorbeeld van afbeelding 3 kunnen we nog slechts 3 nieuwe adressen maken, nl.: 101, 110 en 111. Als deze adressen eenmaal zijn bezet, is verdere uitbreiding niet meer mogelijk. We hebben hier dus 3 adreslijnen totaal $2^3 = 8$ adressen.
Bij veel gangbare typen microprocessoren is op het ogenblik 16 adreslijnen het meest gebruikelijk. Het totaal aantal adressen is hier dus $2^{16} = 65.536$!! Zeker een amateur zal nooit aan zoveel adressen toekomen en dus mag het aantal uitbreidingsmogelijkheden wel onbeperkt worden genoemd.

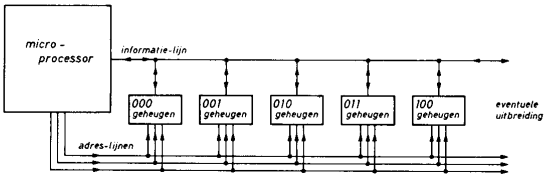
Maar wat kun je er nu mee doen?
Aan de hand van het besproken (vereenvoudigde) microprocessorsysteem zullen we ook een heel eenvoudige toepassing bespreken. Stel, we hebben voor een bepaalde toepassing een schakeling nodig zoals getekend in afbeelding 4. We kunnen nu, met de soldeerbout in de hand, en met de juiste IC's deze schakeling in elkaar solderen. Een andere mogelijkheid is, om een microprocessor-systeem gewoon te 'vertellen' wat er moet gebeuren. In afbeelding 5 is de configuratie getekend waarmee een dergelijke schakeling kan worden gerealiseerd. Om te beginnen hebben we een geheugen nodig om het programma in op te slaan. (Door middel van het programma wordt de microprocessor verteld wat er gedaan moet worden.) Verder zijn er nog drie ingangen en één uitgang nodig. In afbeelding 5 zien we de drie ingangen op de adressen 0000, 0001 en 0010, en de uitgang op adres 0011. De eerste instructie (het begin van het programma) voor de microprocessor staat


micro - processor
informatie-lijn
?
geheugen
geheugen
geheugen
geheugen
geheugen

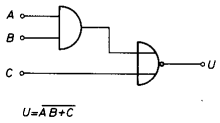
1


elektronische schakelaar
micro - processor
informatie-lijn
adres-lijnen
000 geheugen
001 geheugen
010 geheugen
011 geheugen
100 geheugen

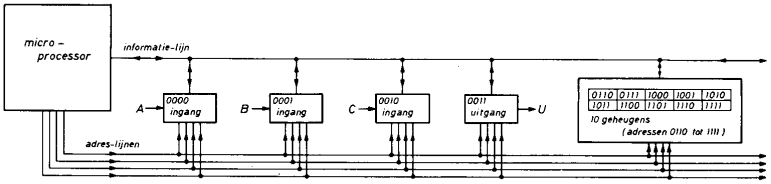
2


micro - processor
informatie-lijn
000 geheugen
001 geheugen
010 geheugen
011 geheugen
100 geheugen
eventuele uitbreiding
adres-lijnen

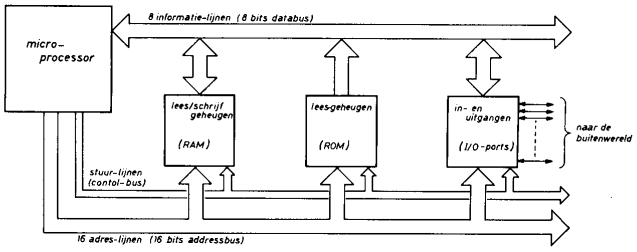
3


A
B
C
U
U=$\overline{A\,B+C}$

4


micro - processor
informatie-lijn
A
0000 ingang
B
0001 ingang
C
0010 ingang
0011 uitgang
U
0110 0111 1000 1001 1010
1011 1100 1101 1110 1111
10 geheugens (adressen 0110 tot 1111)
adres-lijnen

5


micro-processor
8 informatie-lijnen (8 bits databus)
lees/schrijf geheugen
(RAM)
lees-geheugen
(ROM)
in- en uitgangen
(I/O-ports)
naar de buitenwereld
stuur-lijnen (contol-bus)
16 adres-lijnen (16 bits addressbus)

6

in het geheugen op adres 0110. Het programma ziet er nu als volgt uit:

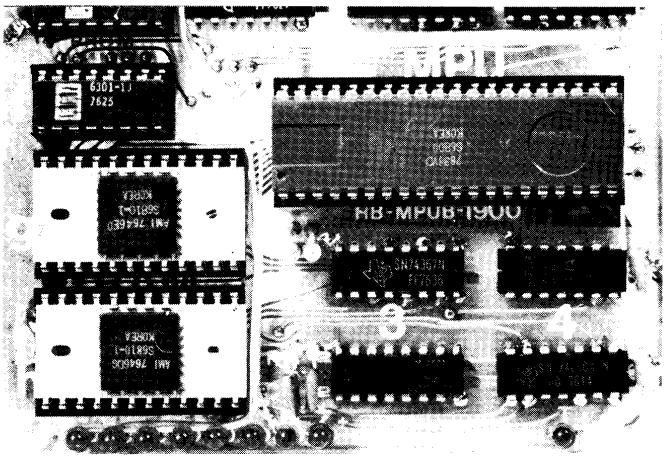
adres	instructie
0110	lees de inhoud van adres 0000
0111	zet de informatie in de buffer
1000	lees de inhoud van adres 0001
1001	voer de 'en' bewerking uit
1010	zet het resultaat in de buffer
1011	lees de inhoud van adres 0010
1100	voer de 'of' bewerking uit
1101	voer de 'niet' bewerking uit
1110	het resultaat naar adres 0011
1111	volgende instructie op adres 0110

1 **De microprocessor wisselt informatie uit met geheugens, maar met welk geheugen??**
2 **Door middel van de adreslijnen geeft de microprocessor aan met welk element informatie uitgewisseld moet worden.**
3 **Een meer praktische methode om informatie uit te wisselen.**
4 **Een confessionele poortschakeling, die we nu met de microprocessor gaan maken.**
5 **Een minimale configuratie, voor het maken van de poortschakeling uit afb. 4.**
6 **Een systeem zoals dat in de praktijk voorkomt. Tussen haakjes de Engelse benamingen, zoals ze normaal worden gebruikt.**

In de kolom 'adressen' staat steeds het nummer van het geheugen waar de instructie is te vinden. De microprocessor hoeft alleen te weten op welk adres de eerste instructie staat. Een teller welke zich in de microprocessor bevindt (de instructieteller) houdt dan verder bij op welk adres de volgende instructie staat.
Nadat de microprocessor gestart is (met de instructieteller op 0110) zal er het volgende gebeuren: Allereerst moet de microprocessor weten wat zijn eerste instructie is. Hij weet op welk adres deze instructie staat (stand van de instructieteller). De adreslijnen worden nu op 0110 gezet, en de eerste instructie stroomt (in digitale code) via de informatielijn de microprocessor in. De microprocessor onthoudt deze instructie in een speciaal instructieregister (een ander woord voor geheugen). Vervolgens zal de microprocessor de instructie ('lees de inhoud van adres 0000') uitvoeren. De adreslijnen worden op 0000 gezet, en de inhoud van adres 0000 zal weer via de informatielijn naar de microprocessor gaan. Deze informatie zal in een speciaal rekenregister, dat zich ook in de microprocessor bevindt, opgeslagen worden. Hiermee is de eerste instructie uitgevoerd, en de microprocessor zal nu gaan kijken wat de volgende opdracht is. De instructieteller is inmiddels een stapje verder gezet (dus van 0110 naar 0111). De microprocessor weet nu dus dat z'n volgende instructie in het geheugen op adres 0111 staat. Wel, de adreslijnen worden op 0111 gezet, en de tweede instructie gaat weer via de informatielijn van het geheugen naar het instructieregister in de microprocessor. Ook deze instructie ('zet de informatie in de buffer') wordt uitgevoerd. Met 'de informatie' wordt bedoeld de informatie die het laatst naar binnen is gehaald, en in het rekenregister stond. (Deze informatie kwam oorspronkelijk van ingang A op adres 0000.) De buffer is een tweede rekenregister. Met deze instructie wordt het eerste rekenregister vrij gemaakt voor het ontvangen van de informatie van ingang B (adres 0001). De instructieteller wordt weer verhoogd, de volgende instructie wordt naar de microprocessor gestuurd en uitgevoerd. Op deze wijze wordt het hele programma afgewerkt. Met de instructie op adres 1110 krijgt de microprocessor opdracht het eindresultaat naar uitgang U op adres 0011 te brengen. Met de laatste instructie ('volgende instructie op adres 0110') krijgt de microprocessor opdracht z'n instructieteller niet op te hogen, maar weer terug te zetten op 0110. Hierdoor zal het programma weer van voren af aan worden uitgevoerd, zodat een eventuele verandering van de ingangssignalen ook weer naar de uitgang wordt doorgegeven.
Natuurlijk zal in de praktijk een dergelijk eenvoudige schakeling nooit met een microprocessor-systeem worden uitgevoerd, deze schakeling was dan ook alleen bedoeld als eenvoudig voorbeeld. Uit dit voorbeeld zien we wel hoe enorm flexibel dit systeem is. Een verandering of uitbreiding van de schakeling kan eenvoudig gerealiseerd worden door een ander programma, en, wat wel zo belangrijk is, met de zelfde onderdelen!!
Juist dit laatste maakt de microprocessor een bijzonder interessant object voor de hobbyist.

Hoe ziet zo'n systeem er nu in de praktijk uit?

Zoals gezegd hebben we als voorbeeld een uiterst eenvoudig systeem genomen, met niet meer in- en uitgangen dan nodig waren. In werkelijkheid zijn er meer gecombineerde in/uitgangen aanwezig. Met behulp van het programma kan dan gedefinieerd worden of een ingang dan wel uitgang wordt bedoeld.
In afbeelding 6 zijn deze in/uitgangsschakelingen getekend in één blok.
Ook bestond de informatielijn slechts uit 'n enkel draadje. De nadenkende lezer zal zich dan ook wel afgevraagd hebben op welke manier een instructiecode over dat enkele draadje van het geheugen naar de microprocessor overgebracht zou moeten worden. Wel, in werkelijkheid is dit ook geen enkel draadje, maar varieert dit aantal (afhankelijk van het type microprocessor) van 2 tot 16 parallelle lijnen (2 tot 16 bits). Bij de meeste microprocessoren zijn dit 8 lijnen. In de schema's worden deze lijnen meestal niet apart getekend, maar als een bundel, met daarin aangegeven hoeveel bits worden bedoeld. Ook in afbeelding 6 zijn de informatielijnen evenals de adreslijnen op deze manier getekend.
Verder zien we 2 typen geheugens, nl. de RAM en de ROM.
De tot nu toe besproken geheugens waren van het RAM-type (Random Acces Memory). In deze geheugens kunnen we informatie opslaan die we later weer nodig hebben. We kunnen dus lezen én schrijven. Als de spanning echter één keer zou wegvallen is alle informatie verdwenen. Daarom zijn bepaalde basisprogramma's (waarover later meer) opgeslagen in ROM's.
Dit zijn geheugens waar de informatie van te voren vast ingezet is. In dit type geheugen kan ook geen informatie geschreven worden. De vast geprogrammeerde informatie kan uitsluitend gelezen worden (vandaar de naam: Read Only Memory). De capaciteit van beide geheugens wordt vaak gegeven in een aantal 'bytes', niet te verwarren met 'bit'. Een byte bestaat uit een aantal geheugen-elementen (een aantal bits) naast elkaar. Met een capaciteit van 1024 bytes worden dus 1024 geheugenadressen bedoeld (vaak wordt 1024 dan afgerond tot 1k.).
Ook nieuw in afbeelding 6 zijn de stuurlijnen. Deze lijnen zorgen dat de timing tussen de verschillende onderdelen goed verloopt. Zij geven b.v. de geheugens een seintje als er weer nieuwe adressen aangeboden worden. Ook geven zij aan, of informatie afgestaan moet worden, ofdat juist informatie moet worden opgeslagen. Voor het begrijpen van het principe van de microprocessor zijn deze stuursignalen echter van secundair belang, en daarom gaan wij er nu niet al te diep op in.
De bedoeling van dit artikel over de microprocessor was de lezer een glo-



7 Een stukje van een compleet systeem. Rechts boven de microprocessor (MPU). Links onder 2 RAM's (G en H).

bale indruk te geven van de werking en de toepassing van de microprocessor.

Het binnenste van de microprocessor

In het voorgaande hebben we dus gezien dat zowel de informatie als de verschillende instructies via de informatielijnen (databus) de microprocessor binnen komen.

We zullen nu wat dieper ingaan op de werkwijze van de microprocessor. Er is door de grote verscheidenheid van microprocessors niet precies aan te geven wat er in 'n microprocessor zit, en hoe het werkt. Dit omdat de inwendige structuur van type tot type sterk kan verschillen. Een algemene werkwijze is wel te geven. Wanneer we uitgaan van een minimale configuratie binnen de microprocessor hebben we nodig: (zie afb. 8).

1. De rekeneenheid (ALU, Arithmetic and Logic Unit). Deze eenheid kan rekenkundige (optellen en aftrekken) en logische ('en', 'of', 'excl. of') bewerkingen uitvoeren.
2. Twee rekenregisters, hierin komen twee getallen te staan waartussen logische of rekenkundige bewerkingen moeten plaatsvinden. Het resultaat wordt dan in één van de twee registers terug gezet, dit register wordt dan vaak 'buffer' of 'accumulator' genoemd.
3. Een 'conditie-code' of 'status' register. Aan de hand van de stand van de verschillende bits van dit register kunnen we b.v. zien of het getal in rekenregister A groter, kleiner dan wel gelijk is aan de inhoud van rekenregister B. Een en ander zal duidelijk worden als we met het programmeren beginnen.
4. De instructie-teller (PC, Program Counter). Deze teller houdt bij met welke regel van het programma we bezig zijn en, na ophoging, met welke regel we door moeten gaan.
5. Het instructieregister. Hierin wordt de uit te voeren instructie onthouden.
6. De stuur- en decodeerlogica. Dit stukje van de microprocessor decodeert de instructie, en regelt aan de hand van het resultaat hiervan het verkeer tussen de verschillende elementen.

Vaak bevinden zich binnen de microprocessor nog extra registers, die het mogelijk maken de programmering van de microprocessor te vereenvoudigen en te verkorten. Maar hierop komen we nog terug.

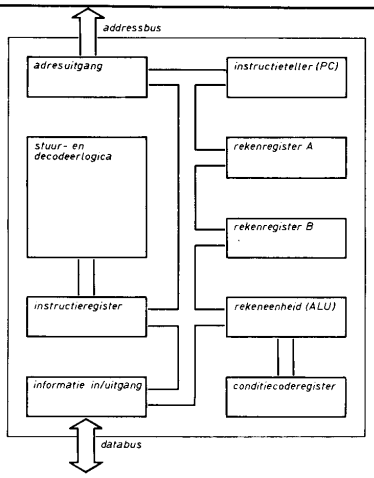
Bij elke programmastap zal de microprocessor als volgt handelen: Allereerst wordt de instructie uit het geheugen naar het instructieregister in de microprocessor gebracht. Deze instructie (eigenlijk maar een deel van de totale instructie) zal bij de meeste microprocessors bestaan uit een combinatie van 8 nullen of enen. Deze instructie wordt gedecodeerd, en aan de hand hiervan zal de zich in de microprocessor bevindende stuurlogica de instructie in een aantal stappen uitvoeren.

Een voorbeeld

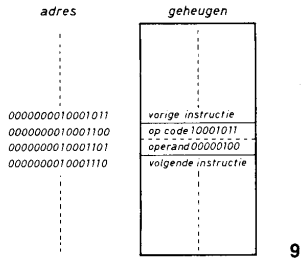
Als voorbeeld nemen we een instructie op adres 0000 0000 1000 1100. De instructie luidt: 'Tel een getal op bij de inhoud van rekenregister A'. Deze instructie wordt wel de OP-code genoemd. De microprocessor moet echter niet alleen weten wát er moet gebeuren, maar ook waarmee het moet gebeuren. Dus, in ons geval, welk getal moet er bij de inhoud van rekenregister A worden opgeteld? Deze informatie moet ook door het programma worden gegeven, maar het eerste deel van de instructie (de OP-code) neemt al 8 bits in beslag, zodat het op te tellen getal onmogelijk ook op hetzelfde adres kan worden geschreven. (Eén byte in het geheugen is maar 8 bits groot). Daarom wordt de informatie die nodig is om de instructie uit te voeren (de Operand), in het voorbeeld dus het getal dat opgeteld moet worden, geschreven op de volgende geheugenplaats. In ons geval is het op te tellen getal dus te vinden op adres 0000 0000 1000 1101. Afb. 9 laat ons nog eens zien hoe de informatie in het geheugen staat. Op adres 0000 0000 1000 1100 staat de op code 1000 1011. Voor de microprocessor betekent dit dus: 'Tel het getal dat op het volgende adres staat, op bij de inhoud van rekenregister A'. Afb. 10 tot en met 13 laten nu zien wat er achtereenvolgens in de microprocessor gebeurt.

Eerst worden de adreslijnen op 0000 0000 1000 1100 gezet. Er wordt een leescommando gegeven, en de informatie die op dit adres staat vloeit via de informatielijnen (databus) in het instructieregister (afb. 10). De instructie (OP-code) wordt gedecodeerd en de instructieteller (PC) wordt opgehoogd. De stuurlogica zorgt er, aan de hand van de gedecodeerde instructie, voor dat nu de juiste stappen genomen worden. (Afb. 11). De stand van de instructieteller (PC) wordt op de adreslijnen gezet en er wordt een 'lees' commando gegeven. Hierdoor zal het op te tellen getal (Operand) op de informatielijnen (databus) komen te staan. De stuurlogica zorgt er voor dat deze informatie terecht komt in rekenregister B. (Afb. 12). De laatste stap: De inhoud van registers A en B worden bij elkaar opgeteld en het resultaat wordt teruggezet in register A. Register A is hier

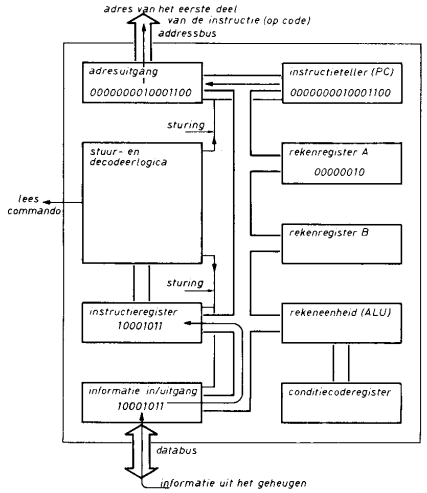
8. Zo kan een eenvoudige microprocessor er van binnen uitzien. Om het programmeren te vereenvoudigen en te verkorten worden vaak nog extra registers toegevoegd.
9. De geheugeninhoud zoals die er bij ons voorbeeld uitziet.
10. De eerste actie van de microprocessor: het ophalen van de instructie.
11. Tweede actie: De instructie wordt gedecodeerd.
12. Derde actie: De Operand wordt opgehaald.
13. Laatste actie: de instructie wordt uitgevoerd.


addressbus
adresuitgang
instructieteller (PC)
stuur- en decodeerlogica
rekenregister A
rekenregister B
instructieregister
rekeneenheid (ALU)
informatie in/uitgang
conditiecoderegister
databus

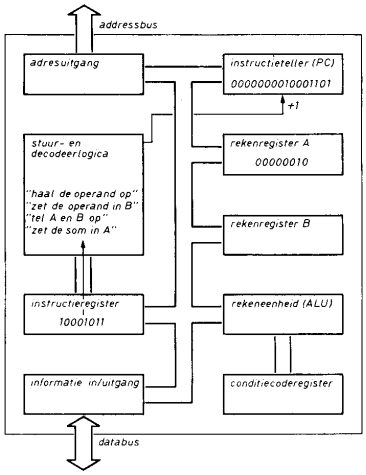
8


adres
geheugen
0000000010001011
0000000010001100
0000000010001101
0000000010001110
vorige instructie
op code 10001011
operand 00000100
volgende instructie

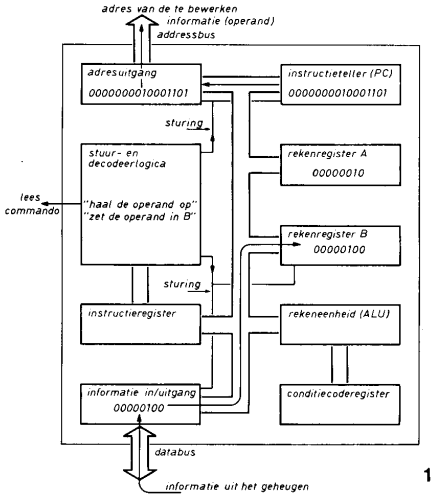
9


adres van het eerste deel van de instructie (op code)
addressbus
adresuitgang
0000000010001100
instructieteller (PC)
0000000010001100
sturing
stuur- en decodeerlogica
lees commando
rekenregister A
00000010
rekenregister B
sturing
instructieregister
10001011
rekeneenheid (ALU)
informatie in/uitgang
10001011
conditiecoderegister
databus
informatie uit het geheugen

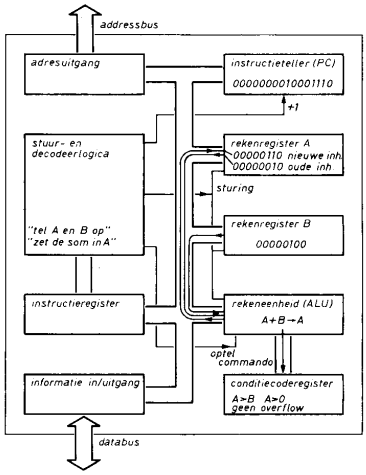
10


addressbus
adresuitgang
instructieteller (PC)
0000000010001101
+1
stuur- en decodeerlogica
"haal de operand op"
"zet de operand in B"
"tel A en B op"
"zet de som in A"
rekenregister A
00000010
rekenregister B
instructieregister
10001011
rekeneenheid (ALU)
informatie in/uitgang
conditiecoderegister
databus

11


adres van de te bewerken informatie (operand)
addressbus
adresuitgang
0000000010001101
instructieteller (PC)
0000000010001101
sturing
stuur- en decodeerlogica
lees commando
"haal de operand op"
"zet de operand in B"
rekenregister A
00000010
rekenregister B
00000100
sturing
instructieregister
rekeneenheid (ALU)
informatie in/uitgang
00000100
conditiecoderegister
databus
informatie uit het geheugen

12


addressbus
adresuitgang
instructieteller (PC)
0000000010001110
+1
stuur- en decodeerlogica
"tel A en B op"
"zet de som in A"
rekenregister A
00000110 nieuwe inh.
00000010 oude inh.
sturing
rekenregister B
00000100
instructieregister
rekeneenheid (ALU)
A+B→A
optel commando
informatie in/uitgang
conditiecoderegister
A>B A>0
geen overflow
databus

13

dus de accumulator. Ook het conditiecoderegister wordt geset, zodat later in het programma voorwaardelijke stappen genomen kunnen worden (afb. 13). Tevens wordt de instructieteller (PC) weer opgehoogd, zodat met de volgende instructie kan worden begonnen.

Een andere schrijfwijze voor de lange binaire getallen

In ons voorbeeld hebben we steeds, zowel de adressen als de inhoud van de adressen, binair geschreven. (Dus in het 2-tallige stelsel.) Het voordeel hiervan is, dat we snel kunnen zien welk bit '1' is, en welk bit '0' is. De microprocessor werkt immers in het binaire stelsel. Een nadeel is dat de lange getallen (b.v. de 16-bits adressen), erg onoverzichtelijk worden, met hierdoor een vergrootte kans op fouten. Bovendien is het bij het schrijven van een programma een tijdrovend werk om steeds de lange rijen enen en nullen op te schrijven. Ook het schrijven van de getallen in het ons bekende decimale (10-tallig) stelsel voldoet niet, omdat deze getallen niet vlot om te zetten zijn in binaire waarden. We zoeken dus een systeem dat enerzijds de 16-bits adressen met veel minder dan 16 cijfers naast elkaar kan weergeven, en dat anderzijds in een oogopslag om te zetten in een binaire waarde, zodat we nog steeds snel kunnen zien welk bit '1' en welk bit '0' is.

Het 8-tallig en 16-tallig stelsel

Juist omdat 8 en 16 machten van 2 zijn, zijn het 8-tallig en 16-tallig stelsel bijzonder geschikt om de lange binaire getallen verkort te schrijven. Zoals bekend hebben we in ons 10-tallig of decimale stelsel 10 verschillende symbolen, nl. 1 tot en met 9 en 0. Evenzo hebben we in het 8-tallig of octale stelsel 8 symbolen, nl. 1 tot en met 7 en 0. Nu kunnen er met een groepje van 3 nullen of enen ook precies 8 combinaties worden gemaakt. Wanneer dus een binair getal naar een octaal getal moet worden omgerekend kunnen we het binaire getal groeperen in groepjes van 3 symbolen. Elk groepje is nu om te zetten in een getal tussen 0 en 7. Indien dit moeilijkheden oplevert kan de nevenstaande tabel geraadpleegd worden.
Voorbeeld: Stel we hebben het binaire getal 11101111. Eerst gaan we groepjes van 3 maken: 011 101 111. Elk groepje wordt omgezet, en we krijgen het octale getal 357. Een identieke redenatie geldt voor het 16-tallig of hexadecimale stelsel. We hebben nu echter 16 symbolen nodig. Voor de eerste 10 symbolen nemen we gewoon de getallen 0 tot en met 9. Voor de resterende symbolen nemen we de letters A tot en met F. Wanneer we hexadecimaal gaan tellen krijgen we dus: 1, 2, 3, ... 8, 9, A, B, C, D, E, F, 10, 11, ... 19, 1A, 1B, 1C, 1D, 1E, 1F, 20 enz. Het omzetten gaat weer hetzelfde, we moeten nu echter groepjes van 4 maken. (Met 4 symbolen hebben we 16 mogelijkheden.) Wanneer we hetzelfde voorbeeld nemen krijgen we na het groeperen: 1110 1111. Hexadecimaal wordt dit: EF. Om in het begin het omzetten te leren kan de volgende tabel een hulp zijn:

	digitale code bij:	
Symbool	8-tallig	16-tallig
0	000	0000
1	001	0001
2	010	0010
3	011	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8		1000
9		1001
A		1010
B		1011
C		1100
D		1101
E		1110
F		1111

Hoewel het omzetten van het hexadecimale of 16-tallig stelsel aanvankelijk wat moeilijker lijkt dan het omzetten van het octale of 8-tallige stelsel, verdient het hexadecimale stelsel toch de voorkeur. Dit, omdat bij de meeste microprocessoren de adressen en de informatielijnen uit een aantal bits bestaat dat een veelvoud van 4 is, zodat het verdelen in groepjes van 4 het meest efficiënt is. Wij zullen dan voortaan de hexadecimale schrijfwijze aanhouden.

De adresseermogelijkheden

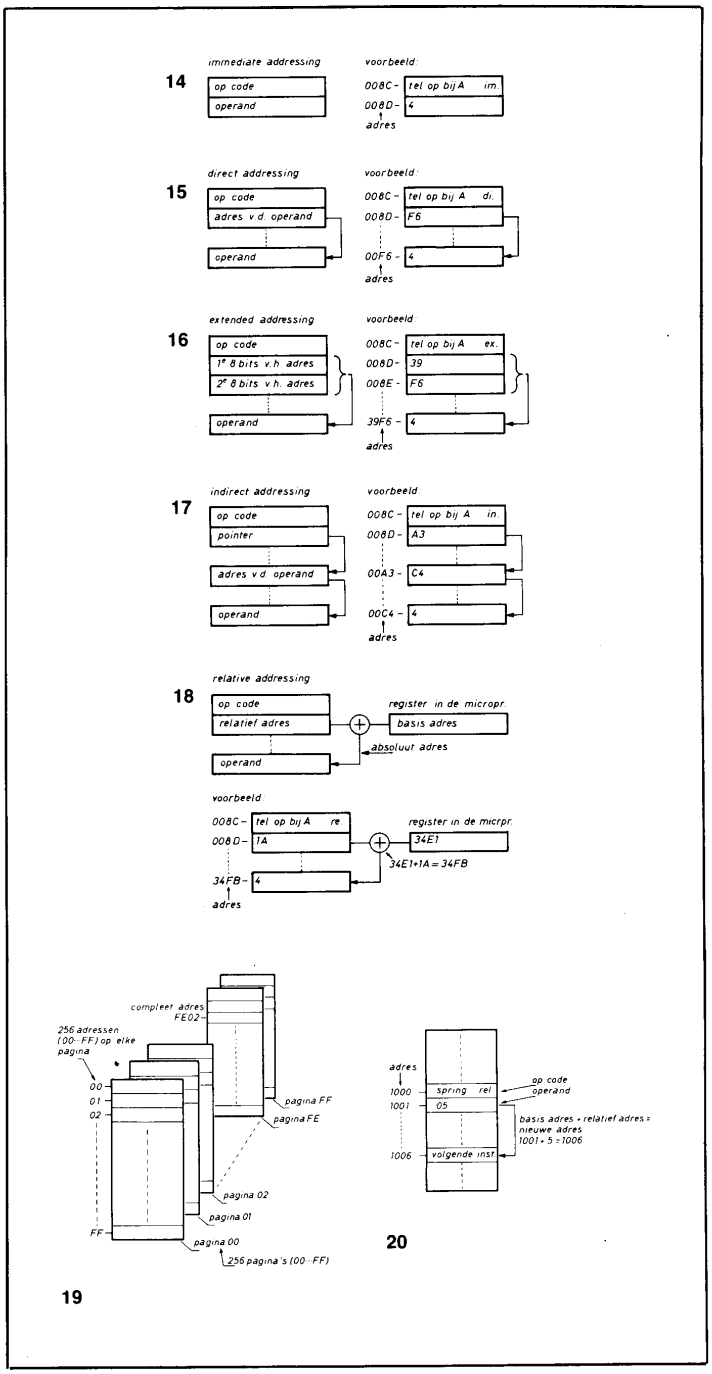
We hebben het nu gehad over de manier waarop een 'doorsnee' microprocessor informatie verwerkt. Een ander belangrijk aspect is het adresseren, dit is een techniek om de te verwerken informatie (de Operand) te lokaliseren. Ook hier bestaan weer talloze varianten bij de verschillende fabrikaten microprocessors. We zullen ons daarom beperken tot een algemene beschouwing van de fundamentele adresseermogelijkheden.
Zoals al eerder gezegd past een volledige instructie bijna nooit op een geheugenadres. Daarom wordt op het eerste adres de eigenlijke instructie (OP-code) gezet, en op het volgende adres de te bewerken informatie (Operand). In het voorgaande voorbeeld stond de instructie 'Tel een getal op bij de inhoud van rekenregister A' op adres 008C. (Hexadecimaal, dus in het 16-tallig stelsel geschreven). Op het volgende adres (008D) staat het getal dat opgeteld moet worden. De totale instructie neemt dus 2 bytes in beslag. In dit geval is dus al bij het programmeren vastgelegd welke waarde moet worden opgeteld. Deze methode van adresseren wordt in het engels 'immediate addressing' genoemd. (Afb. 14). Vaak is echter deze waarde bij het programmeren nog niet bekend, omdat dit het resultaat is van voorgaande bewerkingen en berekeningen, of omdat deze waarde van een ingang moet worden betrokken. In dat geval kunnen we dus niet aangeven **welk** getal opgeteld moet worden. Wel kunnen we aangeven **waar** het getal staat dat opgeteld moet worden. In dit geval moet de OP-code worden gevolgd door het **adres** van een geheugenplaats of ingang waar de te bewerken informatie staat. (Afb. 15). Omdat de adressen meestal uit 16 bits bestaan, en een byte maar 8 bits breed is, kan met deze methode maar een deel van de adressen worden bestreken. Meestal krijgen de eerste 8 bits van het adres een vaste waarde, en kan de waarde van de 2e 8 bits door het programma bepaald worden. Deze methode heet dan 'direct addressing'. Een variant hierop in de 'extended addressing'. Hierbij wordt een volledig 16-bits adres gespecificeerd. Dit wordt gedaan door de tweede 8-bits van het adres in een extra byte te zetten (afb. 16). De totale instructie neemt nu dus 3 bytes in beslag!
Verder bestaat nog de indirecte adressering (indirect addressing). In dit geval geeft het 2e deel van de instructie niet het adres van de te bewerken informatie (de Operand) maar het adres van een geheugenplaats waar het adres van de operand in staat. Een en ander wordt nog eens verduidelijkt in afb. 17. De laatste fundamentele adresseertechniek is de 'relative addressing'.
Hierbij wordt het getal, dat in de tweede helft van het geheugen staat, opgeteld bij de inhoud van b.v. de instructieteller (PC) of bij de inhoud van een extra register. Het resultaat van deze optelling wordt dan beschouwd als het adres van de te bewerken informatie (de Operand) (afb. 18). Bij

veel microprocessors worden ook wel combinaties van bovenstaande adresseertechnieken gebruikt.

De extra registers in de microprocessor

Al eerder hebben we vermeld dat zich in de microprocessor vaak extra registers bevinden om het programmeren te vereenvoudigen en te verkorten. Maar hoe gaat dat nu in z'n werk? We hebben gezien dat we b.v. met 'direct addressing' slechts 8 adresbits kunnen specificeren, en dus maar 2^8 = 256 adressen kunnen bestrijken. Zodra we meer dan die 256 adressen nodig hebben, zouden we aangewezen zijn op 'extended addressing'. Hiermee kunnen we door gebruik van een extra geheugenbyte alle 16 adresbits specificeren. Een instructie neemt dan echter 3 bytes i.p.v. 2 bytes in beslag. Met de toch al beperkte geheugenruimte die meestal bij de kleine microprocessorsystemen aanwezig is, moet deze wijze van adresseren zoveel mogelijk worden vermeden. Een oplossing voor dit probleem is gevonden in de reeds aangehaalde relative addressing. Hierbij wordt het Operand-adres bepaald door het relatieve adres bij een basisadres op te tellen. Voor dit basisadres is er in de microprocessor vaak een appart register aanwezig, het zgn. indexregister. In dit geval heeft men het ook wel over 'indexed addressing'. Wanneer dit indexregister uit 16 bits bestaat is het altijd mogelijk de volledige geheugenruimte te bestrijken.

Een andere vorm van relative addressing is de zgn. 'page-addressing'. Het geheugen wordt hierbij verdeeld in een aantal pagina's. (Afb. 19). Een speciaal 'page-register' dat zich in de microprocessor bevindt, wijst een bepaalde pagina aan. Welke pagina dit is kan worden bepaald door het programma. Op elke pagina bevinden zich 256 adressen, welke met een 8-bits adres geadresseerd kunnen worden. Voor het adresseren is dus maar 1 byte nodig, terwijl toch de volledige geheugenruimte bestreken kan worden, zij het in 256 blokken met elk 256 adressen. Vooral bij sprong opdrachten (dus b.v. wanneer een bepaald programmadeel moet worden overgeslagen) is het handig om bij de ‚relative addressing' de stand van de instructieteller zelf als basisadres te nemen. De instructieteller bevat het adres van de instructie, en het relatieve adres geeft nu in wezen aan hoeveel stappen moeten worden overgeslagen (afb. 20).



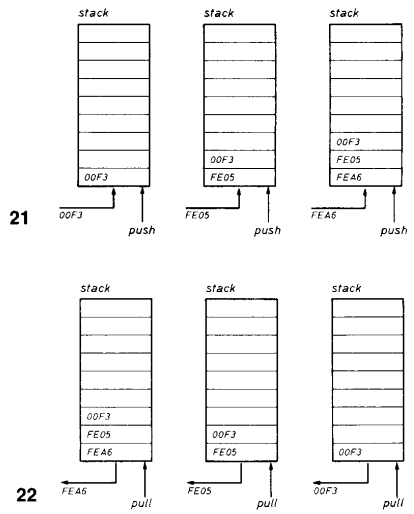
11 - 18 De belangrijkste adresseermogelijkheden. Bij de voorbeelden is steeds dezelfde instructie gekozen: 'Tel 4 op bij de inhoud van register A'.

19. Het geheugen wordt verdeeld in een aantal pagina's. De adressen zijn hexa-decimaal geschreven.

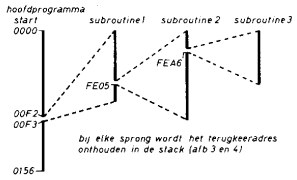
20. Relative addressing met de inhoud van de instructieteller als basisadres.

De subroutines

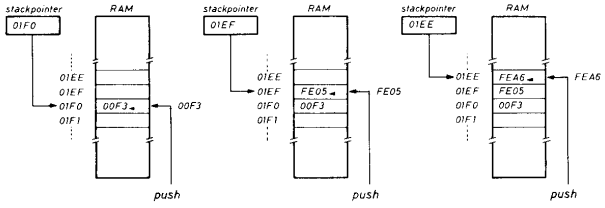
Wanneer een bepaalde reeks opdrachten in een programma veel voorkomt is het niet nodig dit programmadeel steeds opnieuw te schrijven. Vaak wordt zo'n programmadeel als subroutine uitgevoerd. Dit wil zeggen dat dit stukje programma ergens apart in het geheugen komt te staan, en dat vanaf elke plaats in het hoofdprogramma naar dit programmadeel toe gesprongen kan worden. Wanneer dit stukje programma (de subroutine) is afgewerkt wordt weer terug-gesprongen naar de juiste plaats in het hoofdprogramma. Wel moet natuurlijk bekend zijn naar welk adres in het hoofdprogramma moet worden teruggesprongen. Dit adres wordt, voordat naar de subroutine wordt gesprongen, opgeslagen in een speciaal stapelregister. Dit stapelregister wordt ook wel stack genoemd. Hoe de stack werkt zullen we duidelijk maken aan de hand van afb. 21 en 22. We zien dat de stack bestaat uit een aantal registers boven elkaar. De informatie kan echter alleen aan het onderste register worden toegevoerd. Steeds als er nieuwe informatie wordt aangeboden, zal de inhoud van alle registers een plaats naar boven opschuiven (afb. 21). Omdat het net lijkt alsof alle informatie in de stack wordt geduwd, wordt de laadinstructie ook wel 'push' genoemd. Wanneer de informatie uit de stack wordt gehaald ('pull') zal alle informatie weer een plaats zakken (afb. 22). Het voordeel van de stack is, dat hierin informatie kan worden opgeslagen, zonder dat een adres nodig is. Dit houdt in dat de complete instructie ('push' of 'pull') slechts 1 byte in beslag neemt, hetgeen weer een belangrijke besparing van geheugenruimte oplevert. Wel moet men er aan denken dat niet alle informatie willekeurig beschikbaar is. De informatie die het laatst is toegevoerd moet er weer het eerst uit. Dit type register wordt daarom ook wel 'LIFO' genoemd, dit is een afkorting van 'Last In First Out'. Juist bij gebruik van subroutines is de stack bij uitstek geschikt om de terugkeeradressen op te slaan. We kunnen zelfs in de subroutines weer subroutines aanroepen. De terugkeeradressen worden immers steeds in de juiste volgorde in de stack geduwd, en komen er ook weer in de juiste volgorde uit. In afb. 23 zien we dit schematisch weergegeven. Het hoofdprogramma begint op adres 0000 (hexadecimaal). Er wordt driemaal naar een subroutine gesprongen en achtereenvolgens worden de terugkeeradressen OOF3 FEO5 en FEA6 in de stack 'geduwd' (afb. 21). Aan het eind van elke subroutine wordt het terugkeeradres weer uit de stack 'getrokken' (afb. 22). Nadat dit 3 keer gebeurd is zijn we weer terug bij het hoofdprogramma. Hoewel het voor de overzichtelijkheid niet is aangegeven in afb. 23 zal het duidelijk zijn dat de drie subroutines ook vanuit andere plaatsen in het hoofdprogramma kunnen worden aangeroepen.



21 22



23



24

21. **De informatie wordt in de stack 'geduwd'.**
22. **De informatie wordt uit de stack 'getrokken'.**
23. **Een voorbeeld waarin subroutines worden aangeroepen. De getrokken lijn stelt het programmaverloop voor, de stippellijn de sprongen naar de subroutine.**
24. **De stack, maar nu 'nagemaakt' m.b.v. het systeemgeheugen.**

De vorm van de stack

De stack bestaat dus uit een aantal registers boven elkaar. We kunnen de informatie er aan de onderkant 'induwen' of er 'uittrekken', waarbij de reeds aanwezige informatie een plaats opschuift (afb. 21 en 22). Dit stapelregister kan zich in de microprocessor bevinden. Ook is het mogelijk dat een aparte schakeling de stackfunctie vervult. Beide mogelijkheden hebben echter een beperking voor wat betreft de capaciteit. Wanneer er teveel informatie in de stack geduwd wordt, zal de eerst uitgelezen informatie verloren gaan. (De stack loopt a.h.w. over.). Daarom wordt de stack ook wel nagemaakt met behulp van het systeemgeheugen (de RAM). In dit geheugen is doorgaans veel meer ruimte beschikbaar. De uitvoering verschilt echter van die volgens afb. 21 en 22. Dit komt, omdat het systeemgeheugen er niet op is gebouwd om de informatie door te schuiven. Dit schuiven is wel te realiseren, met een speciaal hiervoor geschreven programmaatje. Een dergelijk programma zou echter wat omslachtig zijn, en relatief veel tijd kosten. Een eenvoudiger oplossing is aangegeven in afb. 24. Nieuwe informatie wordt er nu niet vanaf de onderkant 'ingeduwd', maar eenvoudig aan de bovenkant in de eerste nog lege geheugenplaats gezet. Hierdoor is het niet nodig de reeds aanwezige informatie op te schuiven. Wel moet worden onthouden waar de eerste lege geheugenplaats zich bevindt. Daarom wordt het adres van deze geheugenplaats in een speciaal register binnen de microprocessor gezet. De inhoud van dit speciale register wijst dus eigenlijk de eerste lege geheugenplaats in de stack aan, en wordt daarom ook wel de 'stackpointer' genoemd. In afb. 24 zien we, dat steeds wanneer er nieuwe informatie in de stack wordt gezet, de stackpointer automatisch een plaats opschuift. Wanneer er weer informatie uit de stack wordt gehaald, zal de stackpointer weer terug schuiven. Hoewel de stack nu dus anders is opgebouwd, zal hij zich aan de 'buitenkant' hetzelfde gedragen als de stack volgens afb. 21 en 22.

Programma-onderbreking

Soms kan de behoefte ontstaan om van buitenaf het lopende programma te onderbreken, en de microprocessor eerst een ander programma te laten verrichten. Dit is b.v. het geval als de microprocessor moet reageren op een extern signaal, dat aangeeft dat er nieuwe informatie van buitenaf in het geheugen moet worden gezet. Zolang deze informatie nog niet beschikbaar is, zal de microprocessor aan andere programma's kunnen werken. Wanneer de externe informatie op de ingang(en) van het systeem komt te staan, én er wordt een 'interrupt request' aangeboden, (dus een signaal dat aan de microprocessor 'vraagt' of het programma onderbroken kan worden) zal de microprocessor de instructie waarmee hij bezig is, afmaken. Vervolgens zal hij de inhoud van de diverse interne registers in een vaste volgorde in de stack duwen zodat het programma ná de onderbreking weer kan worden vervolgd. Maar eerst zal de microprocessor naar een tweede programma springen, welke de onderbreking behandelt. In ons voorbeeld zal dit programma er voor zorgen dat de aangeboden informatie op de juiste geheugenplaatsen wordt opgeslagen. Wanneer dit tweede programma is afgewerkt, zullen alle registers in de microprocessor weer worden gevuld met hun oude informatie (deze wordt weer uit de stack getrokken) en het onderbroken programma zal worden vervolgd alsof er niets is gebeurd. Er is dus een duidelijk verschil tussen een subroutine en een programmaonderbreking. Bij de subroutine wordt verder gerekend met de reeds aanwezige informatie. De subroutine is dus een onderdeel van het programma.

De stack wordt hier alleen gebruikt om de terugkeeradressen naar het hoofdprogramma te onthouden (afb. 23). Bij de programma-onderbreking wordt een geheel nieuw programma gestart, terwijl alle op dat moment in de microprocessor aanwezige informatie in de stack wordt geduwd.