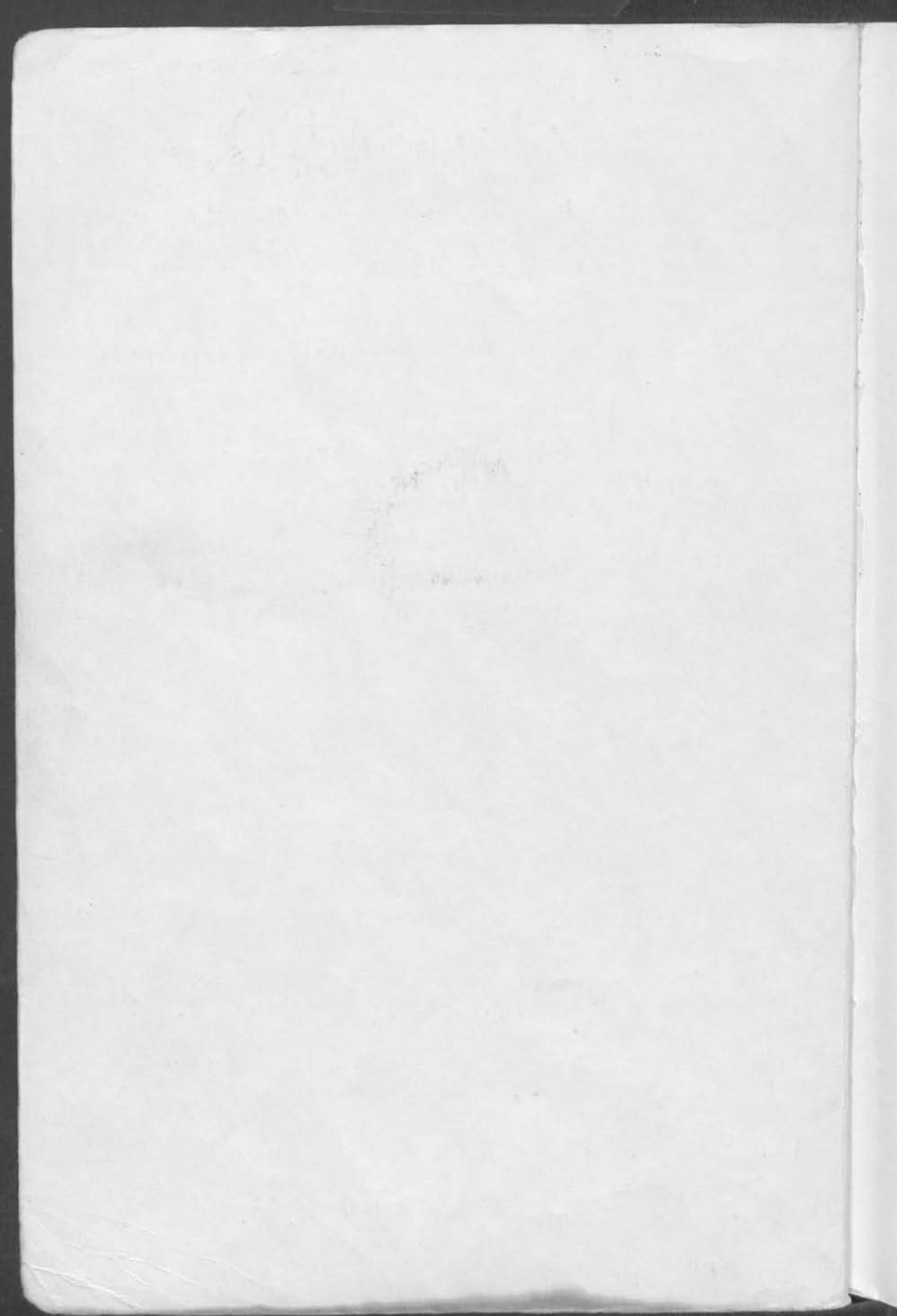


microprocessors

C.J. van Spronsen
F. Bruggeman

21081
Delftse Uitgevers Maatschappij — 1984



Voorwoord

In dit boek wordt de microprocessor behandeld. Het wordt te beginnen beschreven hoe de microprocessor wordt gebruikt en hoe hij kan worden gebruikt in een systeem. Daarna wordt de microprocessor behandeld in zijn toepassing in een systeem.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.

De microprocessor is een microcomputer die bestaat uit een microcontroller en een microprocessor. De microcontroller is een microcomputer die bestaat uit een microcontroller en een microprocessor.



VERVALLEN

C10087
10780

Microprocessors

C.J. van Spronsen en F. Bruggeman



2108 3056

BIBLIOTHEEK TU Delft
P 2108 3056



C

871078

Delftse Uitgevers Maatschappij



© C.J. van Spronsen en F. Bruggeman

Delftse Uitgevers Maatschappij b.v.

P.O. Box 2851, 2601 CW Delft, The Netherlands

Tel. 015-123725

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

ISBN 90 6562 034 6

Voorwoord

In dit boek wordt de microprocessor geïntroduceerd aan de hand van een algemeen model. Het model is geschikt om duidelijk te maken hoe de interne verwerking van instructies plaatsvindt en hoe de samenhang is met externe logica. Deze benadering geeft een beter inzicht in opbouw en werking van een microprocessor dan wanneer wordt uitgegaan van een commercieel leverbaar type, waarbij men vrijwel altijd met bepaalde eigenaardigheden te maken heeft.

De diverse methoden van adressering, zoals die bij microprocessors voorkomen, worden behandeld. De programmatuur wordt vanuit een aantal gezichtspunten gezien: de ontwikkeling van (applicatie-) programmatuur op verschillende niveaus en de daarvoor bestaande ondersteuning in hard- en software komen aan de orde.

Uitgebreide aandacht krijgen de in- en uitvoer, alsmede de speciaal daarvoor verkrijgbare circuits. De standards die er bestaan op hardware-gebied komen aan de orde, zowel die voor koppeling van apparatuur aan een computersysteem als die voor de koppeling van computermodules onderling.

Als afsluiting worden twee benaderingen behandeld die ieder leiden tot een oplossing in die gevallen waar een enkele microprocessor niet aan de gestelde systeemeisen kan voldoen. Ten eerste parallel verwerking, een aantal processors gekoppeld tot één systeem. Ten tweede het gebruik van zogenaamde 'bit-slices', processorbouwstenen, waarmee men zelf een processor kan opbouwen, aangepast aan de specificaties.

Dit boek is ontstaan uit het dictaat *Microprocessors* bij het gelijknamige college aan de TH in Delft. In eerste opzet was het daarbij nodig om wat meer inleidende tekst op te nemen, onder andere in verband met de lopende wijzigingen in studieprogramma's. Gevolg daarvan is dat dit boek nu toegankelijk is voor lezers met een beperkte kennis van digitale technieken.

Delft, augustus 1984

De auteurs

Inhoud

1. Opbouw van een microcomputer	7
1.1. Inleiding	7
1.2. Opbouw van een CPU	8
1.3. Opbouw van een instructie	11
1.4. Instructie-afhandeling	12
1.5. Geheugencircuits	16
2. Halfgeleider geheugens	17
2.1. Inleiding	17
2.2. Geheugenvoorstelling	18
2.3. Overzicht halfgeleider geheugens	19
2.4. Halfgeleider geheugenelementen	19
2.4.1. Random Access Memory (RAM)	19
2.4.2. Read Only Memory (ROM)	23
2.4.3. Seriële geheugens	27
2.5. Opbouw van een geheugensysteem	28
2.5.1. Inleiding	28
2.5.2. Geheugenmodule	29
3. Software begrippen bij microcomputers	32
3.1. Instructies: formaat en soorten	32
3.2. Conditie flip-flops	33
3.3. Adresseringsmethoden	34
3.3.1. Plaats van de operand	34
3.3.2. Redenen voor adresmodificatie	35
3.3.3. Onmiddellijke adressering (immediate addressing)	36
3.3.4. Directe adressering (direct addressing)	37
3.3.5. Indirecte adressering (indirect addressing)	37
3.3.6. Relatieve adressering (relative addressing)	38
3.3.7. Geïndexeerde adressering (indexed addressing)	40
3.3.8. Auto-increment en -decrement	40
3.3.9. Combinaties van adresseringen	41
4. Tijdaspecten in een microcomputer	43
4.1. Inleiding	43
4.2. Tijdbepalende elementen bij geheugens	43
4.2.1. ROM IC's	43
4.2.2. RAM IC's	44
4.3. Communicatie tussen CPU en geheugen	46
4.3.1. Inleiding	46
4.3.2. Validering en synchronisatie	46
4.4. Communicatie tussen CPU en 'langzame componenten'	48
4.4.1. Communicatie met behulp van het READY-mechanisme	48
4.4.2. Synchronisatie met behulp van het INTERRUPT	51
4.5. Direct Memory Access (DMA)	53
4.5.1. Inleiding	53
4.5.2. BURST DMA	54

4.5.3. CYCLE STEALING DMA	54
4.5.4. Transparent DMA	55
5. Communicatie in de microcomputer	56
5.1. Dataflow	56
5.2. Busstructuur	56
5.2.1. Algemene opzet	56
5.2.2. Bustiming	57
5.2.3. Multimaster bussen	58
5.3. Busstandaards	59
5.3.1. De IEEE 696 (S-100)	59
5.3.2. De IEEE 796 (Multibus)	62
5.3.3. Overige bussen	64
6. Interfacing	66
6.1. Inleiding	66
6.2. Wat is een interface	66
6.3. I/O onder programmabesturing	67
6.4. I/O onder programmabesturing en interrupt	68
6.5. I/O onder hardware besturing	69
6.6. Seriële communicatie	70
7.3.1. Circuits voor parallel I/O	81
7.3.2. Circuits voor seriële I/O	86
7.3.3. Interrupt controller	88
7.3.4. DMA-controller	89
7. Ondersteuningscircuits voor microcomputers	72
7.1. Inleiding	72
7.2. Hulpcircuits voor de interne besturing	72
7.2.1. Circuits rond de CPU	72
7.2.2. Busdrivers	75
7.2.3. Dynamic RAM controller	78
7.3. Hulpcircuits voor de input/output	80
7.3.1. Circuits voor parallel I/O	81
7.3.2. Circuits voor seriële I/O	86
7.3.3. Interrupt controller	88
7.3.4. DMA-controller	89
8. Enkele standaards voor interfacing	91
8.1. Inleiding	91
8.2. De IEEE 488 bus (HP-bus)	91
8.2.1. Ontstaan van de bus	91
8.2.2. Organisatie van de interface	92
8.2.3. Beschrijving van de bussignalen	92
8.3. Standaards voor seriële communicatie	95
8.3.1. Inleiding	95
8.3.2. De RS 232-C standaard	95
8.3.3. RS 422 en RS 423	96
8.3.4. De V24, V10 en V11 standaard	97
8.4. Karaktercodering	97

9. Programma-ontwikkeling	101
9.1. Inleiding	101
9.2. Assembler programmering	101
9.2.1. Eigenschappen van assemblers	101
9.2.2. Code generatie	104
9.3. Compilers	105
9.4. Interpretators	107
9.5. Schematische voorstelling vertaling/executie	108
9.6. De editor	110
9.7. Debuggers	110
9.8. Ontwikkelssystemen	111
9.9. Assembler programmering of HLL	113
10. Commercieel verkrijgbare microprocessors	115
10.1. Inleiding	115
10.2. 8-Bits microprocessor	115
10.2.1. Algemeen	115
10.2.2. De 8085	116
10.2.3. De 6502	119
10.2.4. De 2650	122
10.2.5. De 6809	124
10.3. 16-Bits microprocessors	126
10.3.1. Algemeen	126
10.3.2. De 8086	127
10.3.3. De 68000	129
10.4. Enkelchip computers	131
10.4.1. Inleiding	131
10.4.2. De TMS 1000	132
10.4.3. De 8048 reeks	133
11. Bit-slice processorbouwstenen	137
11.1. Inleiding	137
11.2. De centrale processor bouwsteen	137
11.3. Overige bouwstenen	139
11.4. Ontwerp van een processoreenheid	142
11.5. Hulpmiddelen bij het ontwerp	145
11.6. Specifieke toepassingsgebieden	145
12. Parallel processing	147
12.1. Inleiding	147
12.2. Redenen voor parallel processing	147
12.3. Structuren voor parallel processing	148
12.4. Software implicaties	152
12.5. Parallel processing met microprocessor-chips	153
12.6. Parallel processing met boards	155
12.7. Conclusie	155

1. Opbouw van een microcomputer

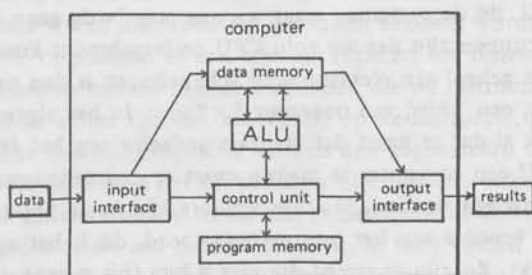
1.1. Inleiding

Voor de verwerking van informatie staan ons een hele serie van machines ter beschikking. Dit loopt van pocketcalculators tot grote computerinstallaties. Van deze reeks vormen de digitale computers een zeer belangrijke deelverzameling. Als meest kenmerkende eigenschappen van een digitale rekenmachine kunnen we noemen:

- Alle informatie wordt intern in de machine voorgesteld door getallen. Deze getallen zijn weergegeven in een binaire vorm.
- De verwerking die de informatie ondergaat wordt bepaald door een reeks opdrachten, 'instructies', die in principe geen deel uitmaken van de machine zelf. Deze reeks opdrachten, het *programma*, wordt wel tegelijk met de gegevens in de machine ingevoerd.

Het geheel aan onderdelen, vaste componenten, van een dergelijk systeem wordt dan aangeduid met de term *hardware* en het totaal aan opdrachten dat de machine uit te voeren heeft noemen we *software*.

Veruit het grootste aantal digitale computers heeft een opbouw als in figuur 1.1 is weergegeven en die in eerste aanleg de *von Neumann architectuur* wordt genoemd naar de Amerikaanse wiskundige die een grote invloed had op het ontwerp van de eerste computers.



Figuur 1.1. Von Neumann architectuur.

In de figuur is weergegeven hoe gegevens (of data) via een *input interface* aan de computer worden toegevoerd. Resultaten komen via een *output interface* ter beschikking aan de buitenwereld. In deze interfaces vindt ook de aanpassing plaats tussen de voorstellingswijze van de gegevens zoals die buiten de computer gebruikt wordt en de wijze waarop de computer de gegevens kan verwerken, respectievelijk verwerkt heeft. Die aanpassing kan bijvoorbeeld zijn: van serie naar parallel, van digitaal naar analoog, of andersom. Ook aanpassingen qua signaalniveau of impedantie moeten we ons hierin denken. Input en output interfaces en alles wat daartussen ligt vormen dan de eigenlijke computer.

De in de figuur aangegeven delen hebben de volgende functie:

- In het *data memory* wordt binnenkomende informatie opgeslagen tot het moment waarop deze informatie nodig is voor verwerking; resultaten kunnen er bewaard worden totdat ze naar de output interface gaan en tussenresultaten die bij berekening of verwerking ontstaan kunnen we er bewaren.
- De *ALU*, wat staat voor *arithmetic and logic unit*, is het orgaan van de computer dat de rekenkundige en logische bewerkingen, welke door instructies worden gespecificeerd, uitvoert. De operanden (gegevens waarop de bewerking moet plaatsvinden)

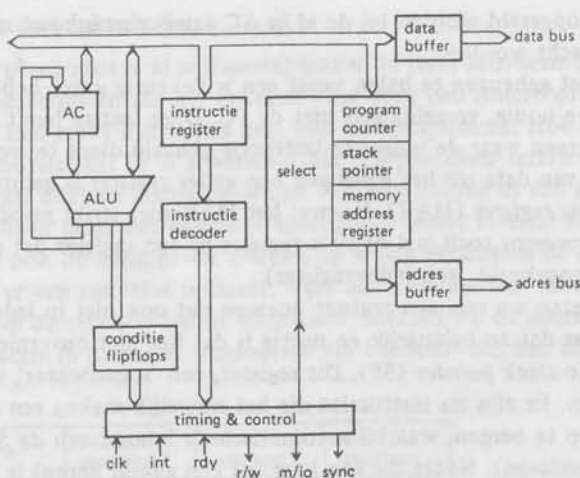
- komen uit het geheugen en de resultaten gaan daar ook weer terug naar toe.
- De *control unit* verzorgt de besturing van het geheel. Deze leest instructies uit het programmeergeheugen, decodeert ze en zendt de nodige besturingssignalen naar de diverse onderdelen van de computer om voor een juiste executie van de opdrachten zorg te dragen.
 - Het *program memory* bevat alle instructies die uitgevoerd moeten worden. Dit totaal, het *programma*, dient vooraf in het geheugen gezet te worden. De opbouw van het programmeergeheugen is vergelijkbaar met die van het datageheugen, reden waarom deze beiden vaak in één enkel blok geheugen worden ondergebracht.

1.2. Opbouw van een CPU

In een computer, en dus ook in een microcomputer, worden bewerkingen op gegevens uitgevoerd. In de inleiding is al vermeld dat de gegevens intern worden voorgesteld door tweetallige cijfers. Deze vorm van getalrepresentatie is de enige waarmee een digitale computer overweg kan. Afhankelijk van wat er met een binaire waarde (een waarde in het tweetallig stelsel) wordt voorgesteld, heeft dat getal een bepaalde betekenis. Zo kunnen ook karakters, bijvoorbeeld de letters A–Z, door een binaire waarde worden weergegeven. Zodra we echter manipulaties op en met gegevens gaan uitvoeren doet het er eigenlijk niet toe wat het voorstelt: de computer ziet niets anders dan een getal.

Het orgaan in de computer dat alle manipulaties uitvoert en regelt is de *central processing unit* of *CPU*. Bij de systemen waar we ons mee bezig gaan houden zou een eerste afbakening kunnen zijn dat we zo'n CPU ondergebracht kunnen denken in een *microprocessor*. Het geheel aan elektronische schakelingen is dan ondergebracht in één geïntegreerd circuit, een 'chip', van ongeveer 5×5 mm. In het algemene schema van een computer bleek al dat er naast dat centrale gedeelte nog het één en ander nodig is. Om van een CPU een computer te maken moet er nog geheugen en input/output voorzieningen toegevoegd worden. Microprocessors zijn in de praktijk onder andere in te delen naar de breedte van het interne datawoord, dit is het aantal bits dat parallel kan worden verwerkt. Zo zijn er typen die met 4 bits (bit = eenheid van tweetallig stelsel) werken, met 8 bits, 12 bits, 16 bits en 32 bits. Voorlopig zullen wij ons hier beperken tot het type dat veruit het meeste voorkomt, namelijk de 8-bits versie. Om een idee te krijgen wat een CPU eigenlijk inhoudt zullen we de interne werking van zo'n CPU gaan bekijken aan de hand van het blokschema van figuur 1.2.

Als we dit blokschema bekijken moet er met een aantal zaken rekening worden gehouden. Allereerst is het zo dat een microprocessor, zoals we die in de praktijk tegenkomen een andere opbouw zal hebben dan de hier geschetste. Dat komt omdat iedere fabrikant in zijn microprocessor een aantal (eigen)aardigheden inbouwt, die hij zelf belangrijk vindt. Meestal gaat het dan om de toevoeging van een aantal registers met al dan niet speciale functies of om een uitgebreide hoeveelheid logica rond de adresbepaling. Ten tweede is het voor het begrip van de interne werking niet belangrijk welke luxe accessoires zijn aangebracht, als de elementaire werking duidelijk is zal het functioneren van een CPU met eventuele uitbreidingen ook wel te begrijpen zijn. In iedere CPU, dus ook in deze, zijn er een aantal elementen die direct duidelijk zijn. Zo is in het gegeven blokschema de functie van de *adresbuffer* en de *databuffer*: het verzorgen van de aanpassing tussen interne niveau's/impedantie en de buitenwereld. Deze buffers fungeren zonder meer als versterker en hebben niet noodzakelijk ook een ingebouwde geheugenwerking. Dit laatste komt in sommige microprocessors wel voor, zie onder andere hoofdstuk 9.



Figuur 1.2. Blokschema CPU.

Instructies vinden we in het geheugen. Om nu bij te houden waar we in een programma bezig zijn, is er één of ander aanwijsmechanisme nodig. Hiertoe kent de CPU een programmateller of *program counter* (PC), waarin steeds het adres wordt bijgehouden van de plaats waar we de volgende uit te voeren instructie moeten halen. De instructie wordt dan opgehaald, de PC met één verhoogd en de opgehaalde instructie uitgevoerd. Om nu te weten met welke instructie we te maken hebben, wordt zo'n opgehaalde instructie in de CPU opgeslagen in een speciaal register, het *instructieregister* (IR). Dit register is verbonden met de *instructiedecoder*, die de instructie decodeert en bepaalt wat de passende acties zijn die, als reactie op de instructie, genomen moeten worden. De feitelijke besturingsignalen worden dan gegenereerd door het blok *timing & control*. Hoewel er in het blokschema hiervoor een enkel blok is aangegeven, moeten we ons dit gedeelte indenken als een gedistribueerde hoeveelheid logica. Alleen als functie is dit gedeelte als een blok te tekenen, in feite vinden we over het hele circuit verspreid gedeelten van de besturingslogica. Intern werkt het geheel op een klokpulssignaal en de besturing verzorgt een, met de klok synchrone, instructieverwerking.

Het basiselement dat voor de uitvoering zorgt van rekenkundige en logische instructies is de *arithmetic and logic unit* (ALU). Dit deel van de CPU voert de bewerkingen uit zoals die door een instructie zijn gespecificeerd. In zo'n stuk logica zal als minimum een opteller zijn opgenomen die twee binaire getallen kan optellen. Vaak is ook aftrekken, logische bewerkingen, schuiven, vergelijken, enzovoorts met een ALU mogelijk. Het resultaat van een bewerking met de ALU kan tevens tot gevolg hebben dat er één of meer van de *conditie flipflops* verandert. Veel voorkomende conditie flipflops zijn onder andere: carry, pariteit, teken en zero. Dit houdt in dat, als er na een bewerking aan een bepaalde conditie is voldaan, bijvoorbeeld: resultaat van de bewerking levert 'nul' op, dat dan de overeenkomstige flipflop op '1' wordt gezet. Via speciale, zogenaamde conditionele, instructies kan dan een voortzetting van het programma gekozen worden die afhankelijk is van het '0' of '1' zijn van de flipflop waarop wordt getest. Data die we in de CPU binnenhalen of die we vanuit de CPU naar buiten sturen komt terecht in of komt uit het *accumulator register* (AC). Deze accumulator kan dus als bron of als bestemming van data fungeren. Dit geldt voor zowel transporten naar buiten als voor interne transporten. Zo kan bijvoorbeeld data, afkomstig uit het geheugen,

door de ALU opgeteld worden bij de al in AC aanwezige inhoud en het resultaat weer in de AC gebracht worden.

Om data uit het geheugen te halen vanaf een willekeurig adres hebben we een aanwijsmechanisme nodig, vergelijkbaar met de PC. Deze laatste heeft al een functie, namelijk: aangeven waar de volgende instructie gehaald dient te worden, zodat voor de adressering van data uit het geheugen een ander register is geïntroduceerd, het *memory address register* (MAR). Hoewel het MAR niet strikt noodzakelijk is, heeft iedere microprocessor toch wel óf zo'n register óf een register dat als zodanig te gebruiken is (bijvoorbeeld: een indexregister).

Als laatste moeten we nog een register noemen dat ook niet in iedere microprocessor voorkomt, maar dat zo belangrijk en nuttig is dat het niet onvermeld mag blijven. Het gaat hier om de *stack pointer* (SP). Dit register, een 'stapelwijzer', wijst naar een adres in het geheugen. Er zijn nu instructies die het mogelijk maken om data op deze geheugenplaats op te bergen, waarbij automatisch de inhoud van de SP met één wordt verhoogd (of verlaagd). Nadat dit één keer (of een aantal keren) is gedaan, kan met een andere instructie de data weer uit het geheugen teruggehaald worden, waarbij dan de inhoud van SP met één wordt verlaagd (respectievelijk: verhoogd). Zo kan een hoeveelheid gegevens tijdelijk op een 'stapel' gezet worden om er later, in de omgekeerde volgorde, weer vanaf te worden gehaald. De stapel is dus een blok in het geheugen en kan net zo groot zijn als het geheugen aan ruimte biedt, met als groot voordeel dat er voor de referentie naar dat geheugen geen aparte adresvermelding door het programma nodig is.

Willen we communiceren met de buitenwereld dan zijn daar een aantal lijnen voor nodig. Allereerst de *adres-* en *databus*, die een aantal bits brede verbinding vormen waarover respectievelijk adressen en gegevens worden getransporteerd. Een enkele adres- en dataverbinding is echter niet genoeg. Er zijn daarnaast nog signalen nodig ter synchronisatie van de transporten. Daartoe dienen de signalen onderaan figuur 1.2. Ze hebben de volgende betekenis. Het signaal *sync* geeft binnen de afhandeling van de instructie een vast referentiepunt in de tijd. Buiten de CPU kan de logica zich instellen/synchroniseren voor een correcte signaalafhandeling. Het signaal *m/io* geeft aan of een adres dat op de adreslijnen komt te staan, een adres is voor het geheugen (m) of een in- of outputregister (io). Ook hierop moet extern gereageerd worden om de juiste verbindingen vrij te maken. Om aan te geven of de gegevens op de databus de processor in moeten of juist eruit, dient het signaal *r/w*. Dit signaal geldt dus zowel bij de communicatie met het geheugen als bij in- of outputoperaties. Het signaal *rdy* (ready) maakt een aanpassing mogelijk in reactietijd tussen CPU en de buitenwereld. Zolang een geheugen (of een in-/outputregister) niet gereed is om te voldoen aan de gespecificeerde bewerking, hetzij lezen, hetzij schrijven, kan het geheugen dit signaal in de 'niet ready' stand houden. De timing & control detecteert dit en zal de CPU zolang laten wachten totdat wel aan de 'ready' voorwaarde is voldaan. Zodoende kan een snelle CPU gekoppeld worden aan een langzamer geheugen (of randapparaat). Een faciliteit die een gebruiker in staat stelt om een lopend programma te onderbreken, heet *interrupt*. Zo'n interruptsignaal, aan te sluiten op *int*, dwingt onmiddellijke afhandeling af. Zonder een interrupt is het vrijwel onmogelijk om een toepassing te realiseren waarin toestanden voorkomen die zo'n directe afhandeling vereisen. Het kloksignaal tenslotte, aan te sluiten op *clk*, voorziet de CPU van een stuursignaal waarop alle interne acties zijn gesynchroniseerd. Ook alle signalen die naar buiten komen zijn gerelateerd aan de klok. De afhandelings tijd van een instructie kan worden uitgedrukt in een aantal malen de periodetijd van het kloksignaal.

1.3. Opbouw van een instructie

In de voorafgaande paragrafen is al een aantal malen de term *instructie* genoemd. De instructie is te beschouwen als een opdracht, die door een (micro)processor kan worden uitgevoerd. Een reeks instructies heet dan een programma. Hoe moeten we ons een instructie voorstellen? Er is al vermeld dat een processor intern alleen met binaire getallen werkt, dus een instructie is ook een voorstelling in een getalvorm. Een instructie die alleen maar een opdracht specificeert levert meestal niet voldoende informatie, er moet ook bij aangegeven worden op welke variabelen de opdracht van toepassing is en als er een resultaat ontstaat, waar dat resultaat dan moet blijven. De variabelen waarop de operatie wordt uitgevoerd noemen we de *operanden* en willen we een instructie in de meest uitgebreide zin beschouwen, dan zou die er als volgt uitzien:

functie	adres operand 1	adres operand 2	adres resultaat
---------	-----------------	-----------------	-----------------

Een dergelijke opbouw van een instructie wordt wel aangeduid als een 3-adres instructie. Meestal zijn we bij het berekenen van een resultaat niet meer geïnteresseerd in de waarde van één van de operanden, wat leidt tot de volgende opzet:

functie	adres operand 1	adres operand 2
= adres resultaat		

Deze opzet leidt tot een opbouw waarvan het geen verbazing zal wekken dat deze met 2-adres instructie wordt aangeduid. Hoewel simpeler, blijkt toch nog een verdere vereenvoudiging mogelijk te zijn als we bedenken dat één van de operanden het resultaat van een vorige operatie is. Dan ligt het voor de hand om in het functiegedeelte ons een aparte plaats in te denken waarin dat resultaat wordt bewaard en waarvan we steeds aannemen dat zich daar één van de operanden bevindt, vóór het uitvoeren van de operatie. We hebben dan de *accumulator* ingevoerd, een algemeen rekenregister. Dat leidt tot het volgende instructieformaat:

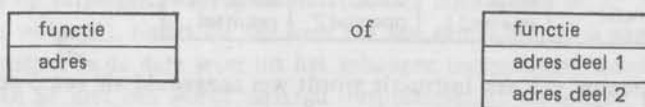
functie	adres operand 1
operand 2 & resultaat } in de accumulator	

Dit levert uiteindelijk de 1-adres instructie op. Dat er zoveel moeite is gedaan om de instructielengte te beperken komt doordat iedere instructie in het geheugen moet worden opgeslagen. Hoe korter de instructie, hoe minder geheugenruimte dit vraagt en hoe sneller de processor het programmagedeelte kan uitvoeren, immers iedere geheugentoeegang kost tijd.

Dit geldt met name in die toepassingen waar we microprocessors tegenkomen, vooral bijvoorbeeld in wat kleinere apparatuur of een besturing met beperkte mogelijkheden. Daar maakt de grootte van een geheugen wel degelijk verschil uit in de totale systeemgrootte.

De stap die gemaakt is om van een 2- naar een 1-adres instructie te komen, namelijk het invoeren van de *accumulator*, hadden we ook kunnen doen bij de stap van een 3- naar een 2-adres instructie. Dan had bijvoorbeeld het resultaat altijd in deze accumulator terecht moeten komen en in de instructie zelf worden de twee adressen van de operanden meegegeven.

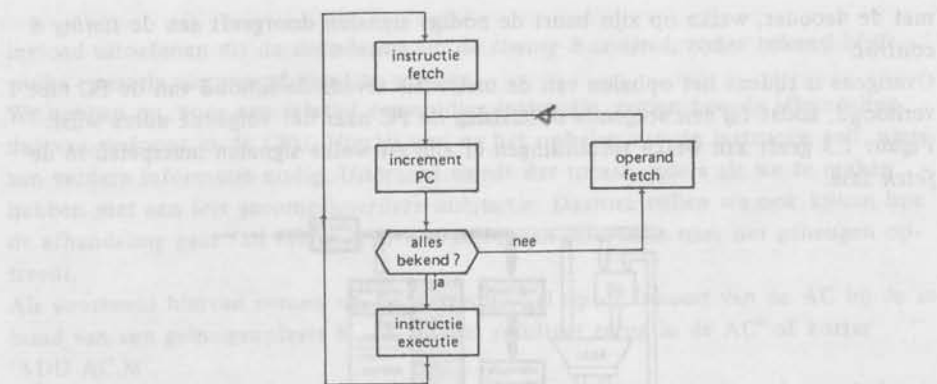
We keren nu terug naar een instructie zoals die hierboven is geschetst, dus met een enkele operand. Zo'n instructie kent dan een gedeelte waarmee de functie wordt gespecificeerd. We noemen dit ook wel de *operatie code* of *opcode*. De opcode geeft dus aan welke functie door de programmeur gewenst is. Als voorbeeld van een functie kunnen we denken aan 'tel op', 'schuif', 'verplaats', 'inverteer', enz. Het totaal aan verschillende functies loopt al gauw in de tientallen. Willen we met een bitcombinatie al deze verschillende mogelijkheden aangeven, dan zijn daarvoor al een 6 à 7 bits nodig. In een 8-bits microprocessor laat dit dan in een enkel woord (van 8 bits = 1 byte) heel weinig ruimte over voor het specificeren van het adres van operand 1. Bij microprocessors (en ook wel bij grotere typen) wordt dan gebruik gemaakt van één of twee woorden van ieder 8 bits, die direct na de instructie in het geheugen staan, en waarin het adres van de operand is weergegeven. Een dergelijke instructie heet dan een *meerdere woorden instructie* en in het geheugen kunnen we ons die als volgt voorstellen:



Het eerste voorbeeld geldt voor een instructie waarbij bijvoorbeeld het adres relatief is opgegeven ten opzichte van een afgesproken adres, meestal de programmateller. In het tweede voorbeeld wordt het adres van de operand volledig bepaald door de 16 adresbits die in twee opvolgende 8-bits woorden staan opgeslagen. In de bovengenoemde voorbeelden wordt gesproken van respectievelijk 2- en 3-byte instructies, naar de hoeveelheid bytes die nodig zijn voor een complete instructie.

1.4. Instructie-afhandeling

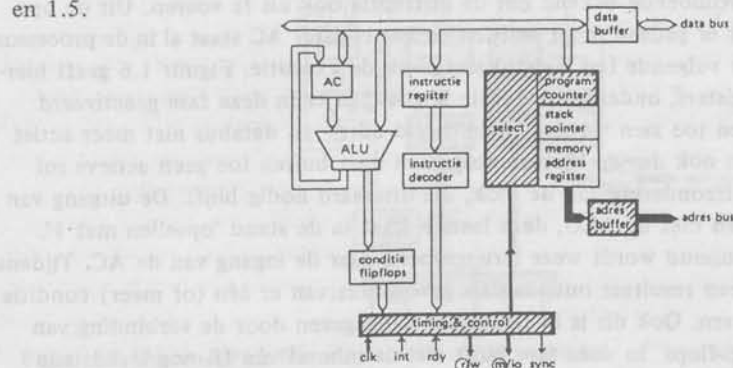
Om een complete instructie in een enkele slag af te ronden is een enorme hoeveelheid logica nodig. Zelfs zoveel, dat een dergelijke benadering niet verantwoord is. In de praktijk zien we dat een instructie dan ook in een aantal stappen wordt afgehandeld. Allereerst is het nodig om te weten om welke instructie het gaat, dus we beginnen met de instructie op te halen uit het geheugen. Hiertoe wordt de geheugenplaats, waarvan de PC het adres aangeeft, uitgelezen. Deze uitleesslag wordt aangeduid met *instructie fetch*. De instructie wordt in het instructieregister geplaatst en de instructie-decodering geeft nadere informatie door aan de *timing & control*. Er is dan bekend of, en zo ja, hoeveel informatie er verder nog nodig is om de instructie te kunnen uitvoeren. Blijkt nu dat er nog een volgend byte gehaald moet worden (of zelfs nog twee bytes), dan wordt dit door de besturing gedaan. In een tweede (en eventueel derde slag) wordt dan de verdere informatie opgehaald. Ook hierbij geeft de PC het adres van de geheugenplaats die wordt uitgelezen. Deze slag(en) duidt men aan als *operand fetch*. Pas daarna weet de processor genoeg om de instructie te kunnen uitvoeren. Er volgt dan nog een laatste processorslag, deze slag heet dan de *executie*. Na iedere *fetch*-slag wordt de PC steeds met één verhoogd voordat aan de volgende slag wordt begonnen om er steeds voor te zorgen dat de PC het volgende adres klaar heeft. Dit proces van diverse slagen en het ophogen van de PC kunnen we weergeven in een stroomdiagram, zie figuur 1.3. Deze figuur geeft een functionele indeling van de diverse acties die nodig zijn om een instructie op te halen, de operand(en) op te halen en de instructie uit te voeren.



Figuur 1.3. Schematische voorstelling van een instructieverwerking.

Naast de voorstelling van de instructie-afhandeling, zoals die in figuur 1.3 is gegeven, bestaat er nog een manier van weergeven. Dat is in het blokschema van de processor. In dat schema geven we dan in een aantal opeenvolgende stappen aan, welke registers actief zijn, hoe de transporten verlopen, welke bussen signalen vervoeren, waar naar toe en wat er intern nog gebeurt. Om dat duidelijk te maken zal een tweetal instructies bekeken worden en de totale verwerking ervan worden voorgesteld in het blokschema van figuur 1.2.

Allereerst is dat de instructie *increment AC*, verhoog de inhoud van de accumulator met 1. In de tijd gezien zal om te beginnen deze instructie opgehaald moeten worden uit het geheugen. Dat ophalen van de instructie, de *instruction fetch*, geven figuur 1.4 en 1.5.



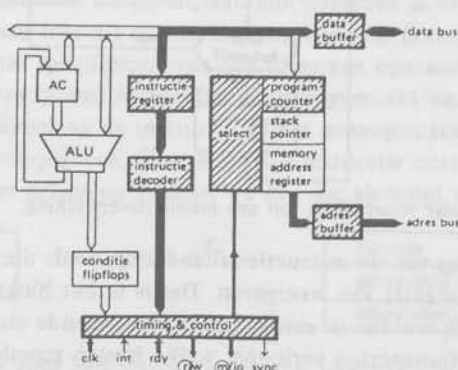
Figuur 1.4. Het ophalen van een instructie (*instruction fetch*).

De inhoud van de PC komt op de adresbus te staan omdat *timing & control* via *select* de PC selecteert en doorkoppelt aan de ingang van de adresbuffer. Tevens wordt het signaal *r/w* in de stand *r* gezet om aan te geven dat het om een leesopdracht gaat en het signaal *m/io* geeft aan *m* omdat de opdracht aan het geheugen wordt gegeven. Dit geheel onder besturing van een aangeboden *clk* klokpulssignaal. Het adres op de adresbus is de plaats waar de instructie in het geheugen staat. Het geheugen wordt geactiveerd, mede door *r/w* en *m/io* en als de inhoud van de geadresseerde plaats beschikbaar is meldt het geheugen dit door het signaal *rdy* te activeren. Dit is het sein om de databussignalen via de databuffer door te geven naar het instructieregister, waarin deze waarde wordt opgeslagen. Immers, de aangeboden informatie van de databus is de instructie die we ophaalden uit het geheugen. De uitgang van IR is doorverbonden

met de decoder, welke op zijn beurt de nodige signalen doorgeeft aan de *timing & control*.

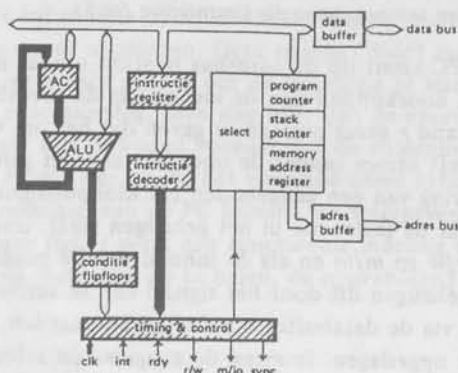
Overigens is tijdens het ophalen van de instructie tevens de inhoud van de PC met 1 verhoogd, zodat bij een volgende ophaalslag de PC naar het volgende adres wijst.

Figuur 1.5 geeft aan welke verbindingen er zijn en welke signalen meespelen in de *fetch* fase.



Figuur 1.5. Actieve verbindingen en signalen in de *instruction fetch* fase.

Na afloop van deze fase, de *instruction fetch*, kunnen we bekijken of er in de processor voldoende gegevens voorhanden zijn om de instructie ook daadwerkelijk te kunnen uitvoeren. Normaal gesproken betekent dit, dat we over moeten gaan tot het ophalen van de operand(en). In het geval met de instructie *increment AC* is er na de instructie *fetch* al voldoende bekend om de instructie ook uit te voeren. Uit de operatiecode blijkt wat er gedaan moet worden en het register AC staat al in de processor ter beschikking. De volgende (en laatste) slag is nu de executie. Figuur 1.6 geeft hiervan weer welke registers, onderdelen, verbindingswegen er in deze fase geactiveerd worden. Naar buiten toe zien we dat bijvoorbeeld adres- en databus niet meer actief gestuurd worden en ook dat de besturingssignalen naar buiten toe geen actieve rol meer spelen; met uitzondering van de klok, die uiteraard nodig blijft. De uitgang van AC wordt verbonden met de ALU, deze laatste staat in de stand 'optellen met 1'. De verhoogde AC-inhoud wordt weer teruggevoerd naar de ingang van de AC. Tijdens de bewerking kan een resultaat ontstaan als gevolg waarvan er één (of meer) conditie flip-flop(s) veranderen. Ook dit is in figuur 1.6 aangegeven door de verbinding van de ALU met de flip-flops. In deze fase blijft wel de inhoud van IR nog steeds zijn



Figuur 1.6. Executie van de *increment AC* instructie.

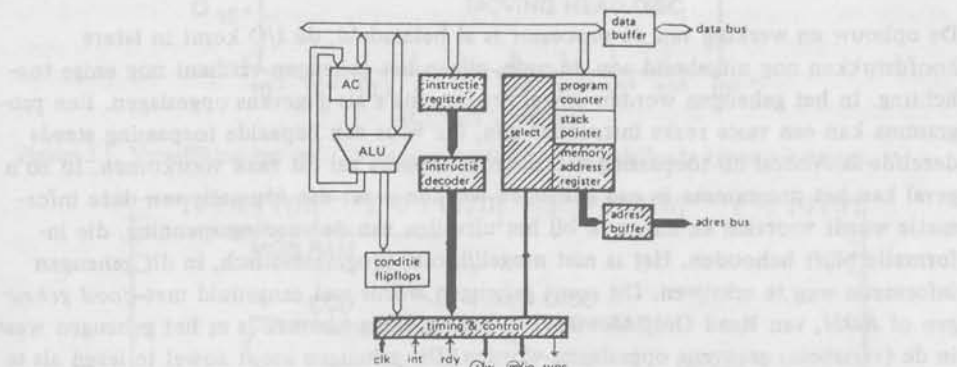
invloed uitoefenen via de decodering op de *timing & control*, zodat bekend blijft welke operatie uitgevoerd dient te worden.

We hebben nu, voor een relatief eenvoudige instructie, gezien hoe de afhandeling daarvan verloopt in de CPU. Hierbij was na het ophalen van de instructie zelf, niets aan verdere informatie nodig. Uiteraard wordt dat totaal anders als we te maken hebben met een iets gecompliceerdere instructie. Daarom zullen we ook kijken hoe de afhandeling gaat van een instructie waarbij een referentie naar het geheugen optreedt.

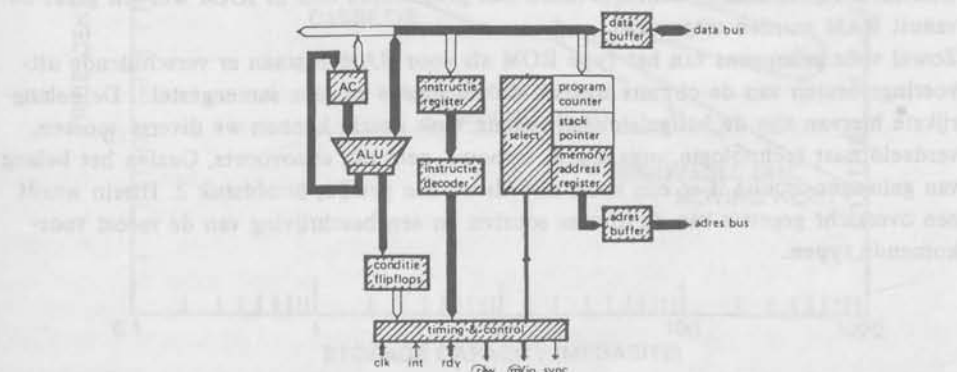
Als voorbeeld hiervan nemen we de instructie 'tel op de inhoud van de AC bij de inhoud van een geheugenplaats M, en zet het resultaat terug in de AC' of korter 'ADD AC,M'.

Het eerste gedeelte van de afhandeling, de *instruction fetch*, gaat precies zoals bij de vorige instructie, dus daarvoor kan worden verwezen naar de figuren 1.4 en 1.5 met bijbehorende tekst. Dan zijn we in de situatie terecht gekomen waarin het IR de instructie bevat en er blijkt dat, om de instructie uit te kunnen voeren, er nog een operand gehaald moet worden en wel de inhoud van geheugenplaats M. Het adres van deze geheugenplaats geeft register MAR. (Dit register kan in een voorgaand programma-gedeelte gevuld zijn met het adres van de gewenste plaats in het geheugen.) Een volgende toestand, zie figuur 1.7, laat zien dat nu het MAR via de adresbuffer wordt doorgeschakeld aan de adresbus met als actieve signalen *r/w* en *m/io*.

De processor wacht tot van het geheugen de *rdy* actief wordt als bevestiging van het klaar staan van de data. Gebeurt dit, dan kan de executieslag beginnen. In de CPU



Figuur 1.7. *Operand fetch* fase van de ADD AC,M instructie.



Figuur 1.8. Executie van de ADD AC,M instructie.

worden de doorkoppelingen gelegd die in figuur 1.8 zijn geschetst. De databus komt, via de databuffer aan de ene ingang van de ALU. Via de *timing & control* wordt de ALU in de stand 'optellen' gezet. De andere ingang van de ALU komt aan de AC en de uitgang van de ALU gaat naar de ingang van de AC.

Ook kunnen, afhankelijk van het resultaat van de bewerking, één of meer conditie flip-flops veranderen. De processor is dan klaar met deze instructie en kan aan de volgende beginnen.

Voor iedere instructie is op die manier aan te geven hoe de interne afhandeling plaatsvindt. Overigens blijkt wel dat de klokfrequentie van een bepaald type microprocessor niet alleen bepalend is voor de tijd waarin een instructie wordt uitgevoerd. Afhankelijk van de interne opbouw kan in het ene type een combinatie van acties parallel gebeuren, waar een andere die zelfde acties in een aantal slagen na elkaar moet doen.

1.5. Geheugencircuits

Het geheugen vormt een belangrijk onderdeel van een computersysteem. In een veelgebruikt schema wordt een computer wel voorgesteld als in figuur 1.9. We onderscheiden daarin drie blokken: processor, geheugen en invoer/uitvoer (I/O).



Figuur 1.9. Blokschema computer.

De opbouw en werking van de processor is al behandeld, de I/O komt in latere hoofdstukken nog uitgebreid aan de orde, alleen het geheugen verdient nog enige toelichting. In het geheugen worden zowel programma's als gegevens opgeslagen. Een programma kan een vaste reeks instructies zijn, die voor een bepaalde toepassing steeds dezelfde is. Vooral bij toepassing van microprocessors zal dit vaak voorkomen. In zo'n geval kan het programma in een geheugen worden gezet dat éénmalig van deze informatie wordt voorzien en dat, ook bij het uitvallen van de voedingsspanning, die informatie blijft behouden. Het is niet mogelijk om, programmatisch, in dit geheugen informatie weg te schrijven. Dit soort geheugen wordt wel aangeduid met *dood geheugen* of *ROM*, van Read Only Memory. In tegenstelling hiermee is er het geheugen waarin de (variabele) gegevens opgeslagen worden. Dit geheugen moet zowel te lezen als te schrijven zijn en wordt daarom wel aangeduid met *RAM*, van Read Alterable Memory. Uiteraard kan in voorkomende gevallen een programma ook in RAM worden gezet en vanuit RAM worden uitgevoerd.

Zowel voor geheugens van het type ROM als voor RAM bestaan er verschillende uitvoeringsvormen van de circuits waaruit de geheugens worden samengesteld. De belangrijkste hiervan zijn de halfgeleidergeheugens. Ook daarin kennen we diverse soorten, verdeeld naar technologie, organisatie, opbouw, gebruik, enzovoorts. Gezien het belang van geheugencircuits is er een apart hoofdstuk aan gewijd, hoofdstuk 2. Hierin wordt een overzicht gegeven van de diverse soorten en een beschrijving van de meest voorkomende typen.

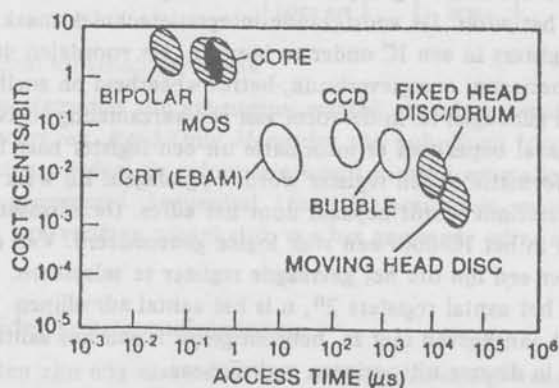
2. Halfgeleider geheugens

2.1. Inleiding

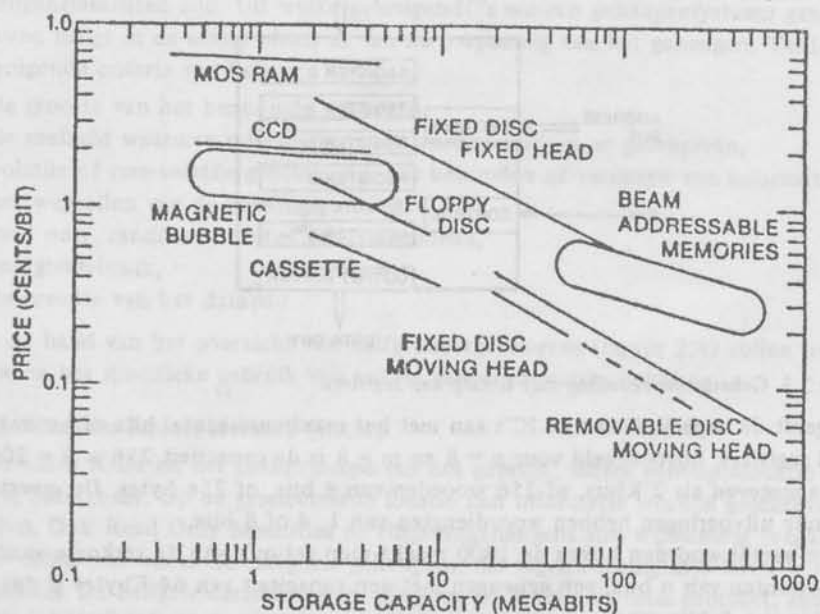
Geheugens worden gebruikt voor het bewaren van informatie. Globaal zijn er vier gebieden aan te geven voor het geheugengebruik, te weten:

- vast massageheugen voor het bewaren van macroprogrammatuur, data-bestanden enz.
- tijdelijk bulkgeheugen, waarin ruwe informatie voor verdere bewerking wordt opgeslagen,
- werkgeheugen, gevormd door een klein en snel geheugen voor het bewaren van tussentijdse resultaten, testfuncties en referenties,
- vast geheugen, waarin vaste informatie wordt bewaard, zoals systeemprogrammatuur, codetabellen en waardetabellen bijvoorbeeld voor mathematische functies.

In microcomputers worden voornamelijk halfgeleidergeheugens toegepast. Andere typen geheugens zoals magnetische bellen, ringkernegeheugens, draadgeheugens, Moving



Figuur 2.1. Vergelijking van prijs/bit en toegangstijd voor verschillende typen geheugens.



Figuur 2.2. Vergelijking prijs/bit en opslagcapaciteit voor diverse typen geheugens.

Head Disc, Fixed Head Disc/Drum, Charged Couple Devices dienen als achtergrond-geheugen of worden gebruikt in speciale toepassingen.

Halfgeleidergeheugens zijn snelle geheugens maar duur. Een vergelijking wat betreft snelheid en prijs ten opzichte van andere typen geheugens is in figuur 2.1 te vinden, aan de hand van het prijspeil '79, hetgeen voor de bipolaire en MOS geheugens circa 10 maal voordeliger is geworden.

Ontwikkelingen in de IC-fabricage en op de IC-markt zorgen er voor, dat de prijs van de halfgeleidergeheugens nog steeds dalende en de snelheid stijgende is.

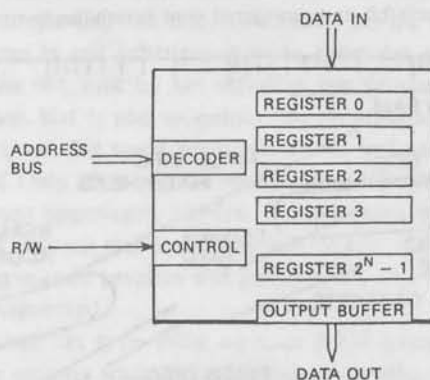
Vergelijken we de prijs per bit met de opslagcapaciteit (zie figuur 2.2) dan zien we de halfgeleidergeheugens geplaatst bij de dure geheugens en bij een geringe opslagcapaciteit.

2.2. Geheugenvoorstelling

Men kan een computergeheugen voorstellen als een groot aantal registers van gelijke lengte. Ieder register heeft zijn eigen plaats. Aan deze plaats wordt een numerieke waarde toegekend, het *adres*. De voortgaande integratietechniek maakt het mogelijk een groot aantal registers in een IC onder te brengen. De voordelen daarvan zijn terug te vinden in de afmetingen, energieverbruik, betrouwbaarheid en snelheid.

In figuur 2.3 is een geheugen-IC in de vorm van een verzameling registers getekend. Het Read/Write signaal bepaalt of er informatie uit een register naar buiten wordt gevoerd, of dat er informatie in een register wordt opgeslagen. Bij welk register een lees/schrijffactie plaatsvindt wordt bepaald door het adres. De adressen zijn binair gecodeerd en worden in het IC door een stuk logica gedecodeerd. Van de decoder gaat er naar ieder register een lijn om het gevraagde register te selecteren.

In dit voorbeeld is het aantal registers 2^n , n is het aantal adreslijnen. De lengte van ieder register wordt aangegeven met m , hetgeen gelijk is aan het aantal bitplaatsen. Geheugen-IC's zijn in diverse uitvoeringen verkrijgbaar.



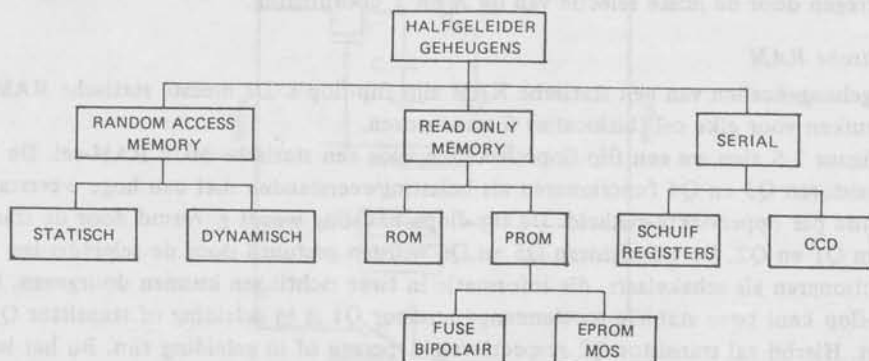
Figuur 2.3. Geheugenvoorstelling met behulp van registers.

Men geeft de capaciteit van de IC's aan met het maximum aantal bits of het maximum aantal registers, bijvoorbeeld voor $n = 8$ en $m = 8$ is de capaciteit $256 \times 8 = 2048$ bits, ook aangegeven als 2 Kbits, of 256 woorden van 8 bits, of 256 bytes. De meest voorkomende uitvoeringen hebben woordlengten van 1, 4 of 8 bits.

Bij een aantal woorden boven de 1000 maakt men gebruik van de verkorte aanduiding in K woorden van n bits, een geheugen met een capaciteit van 64 Kbytes is dus 65.536 woorden van 8 bits groot.

2.3. Overzicht van halfgeleider geheugens

De geheugens, die in microcomputers veelvuldig toegepast worden, zijn in figuur 2.4 aangegeven. Een eerste onderscheid is de wijze waarop we toegang krijgen tot de geheugencellen.



Figuur 2.4. Overzicht halfgeleidergeheugens.

Random Access Memories zijn geheugens, waarbij elk adres direct toegankelijk is om te lezen of te schrijven. *Read Only Memories* zijn ook direct toegankelijk, maar kunnen alleen gelezen worden, de informatie wordt meestal eenmalig door fabrikant of gebruiker geprogrammeerd. *Sequential Access Memories* zijn seriële geheugens, waarbij de toegangstijd kan variëren afhankelijk van het gevraagde adres en de toestand van het geheugen.

2.4. Halfgeleider geheugenelementen

De IC-fabrikanten zijn nog niet zover, dat ze het door ons gewenste geheugensysteem in één IC kunnen leveren. Het geheugen zal daarom een verzameling van een aantal geheugenelementen zijn. Uit welke geheugen-IC's we een geheugensysteem gaan opbouwen hangt in de eerste plaats af van de toepassing van het geheugen. Verder zijn de volgende criteria van belang:

- de grootte van het benodigde geheugen,
- de snelheid waarmee informatie moet worden gelezen en geschreven,
- volatile of non-volatile gebruik, d.i. het behouden of verliezen van informatie bij het wegvallen van de voedingspanning,
- read only, random access of serial memories,
- energieverbruik,
- de breedte van het datapad.

Aan de hand van het overzicht van halfgeleidergeheugens (figuur 2.4) zullen we de opbouw en het specifieke gebruik van enkele geheugencircuits bekijken.

2.4.1. Random Access Memory (RAM)

In de naam RAM zit het basisprincipe van het gebruik, iedere geheugenlocatie is willekeurig bereikbaar. Op de geselecteerde locatie kan informatie worden geschreven of gelezen. Ook Read Only Memories en ringkerngeheugens zijn willekeurig toegankelijk, maar vallen niet onder de categorie geheugens, die tegenwoordig met RAM worden aangeduid. De halfgeleidertechnieken, die voor de RAM's worden gebruikt, zijn n-channel MOS (Metal Oxide Silicon), p-channel MOS en bipolair. Bipolaire geheugens

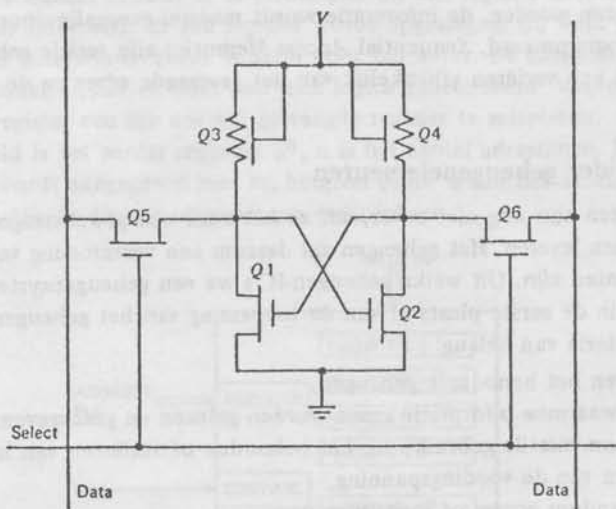
zijn circa 5 maal zo snel als MOS geheugens. MOS RAM's kunnen we verder nog onderverdelen in statische en dynamische geheugens.

Een RAM geheugen bestaat uit een matrix van geheugencellen, waarin digitale informatie kan worden opgeslagen of uitgelezen. Toegang tot een gegeven locatie wordt verkregen door de juiste selectie van de X en Y coördinaten.

Statische RAM

De geheugencellen van een statische RAM zijn flip-flop's. De meeste statische RAM's gebruiken voor elke cel (bitlocatie) 6 transistoren.

In figuur 2.5 zien we een flip-flopschakeling voor een statische MOS RAM-cel. De MOS transistoren Q3 en Q4 functioneren als belastingweerstand met een hoge weerstandswaarde per oppervlakte-eenheid. De flip-flopschakeling wordt gevormd door de transistoren Q1 en Q2. De transistoren Q5 en Q6 worden gestuurd door de selectdraden en functioneren als schakelaars, die informatie in twee richtingen kunnen doorgeven. De flip-flop kent twee stabiele toestanden, transistor Q1 is in geleiding of transistor Q1 spert. Hierbij zal transistor Q2 respectievelijk sperren of in geleiding zijn. Bij het lezen van de cel zal de geleidende tak van de flip-flop de bijbehorende datalijn naar het aardpotentiaal trekken. Schrijven van de cel gebeurt door de datalijnen naar het gewenste niveau te forceren tijdens selectie van de cel.

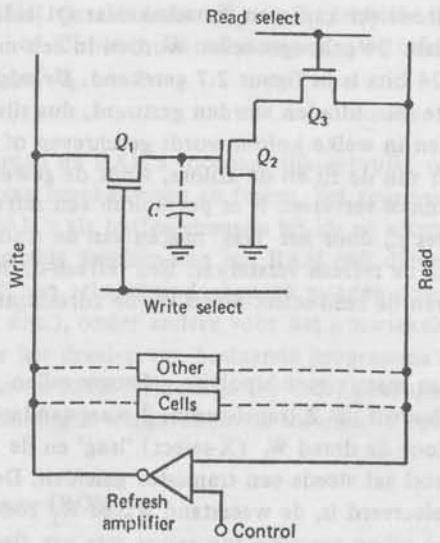


Figuur 2.5. Statische MOS RAM-cel.

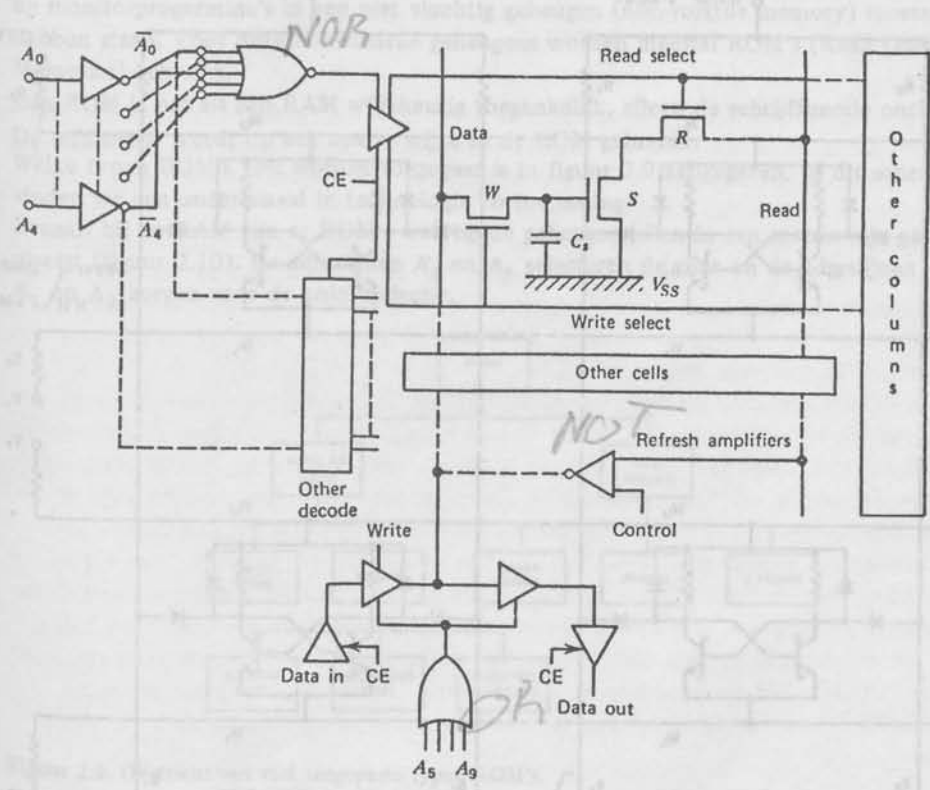
Dynamische RAM

Dynamische RAM's maken voor het opslaan van informatie gebruik van de aan- of afwezigheid van een condensatorlading. Per cel zijn 1 à 3 transistoren nodig. De condensator zal zijn lading door lekstromen verliezen, waardoor ook de informatie verdwijnt. De opgeslagen informatie moet daarom regelmatig worden ververs, gemiddeld eens in de 2 ms. In figuur 2.6 is een voorbeeld van een dynamische RAM geheugencel getekend.

De cel bestaat uit drie transistoren. De capaciteit, waarin de lading (informatie) wordt opgeslagen, bevindt zich aan de gate van de MOS transistor Q2. Is de lading voldoende groot, dan vertegenwoordigt deze de logische '1'. Q3 zal door sturing van de read-select het geheugenelement Q2 met de DATA OUT (read)-lijn verbinden.



Figuur 2.6. Dynamische MOS RAM geheugencel.

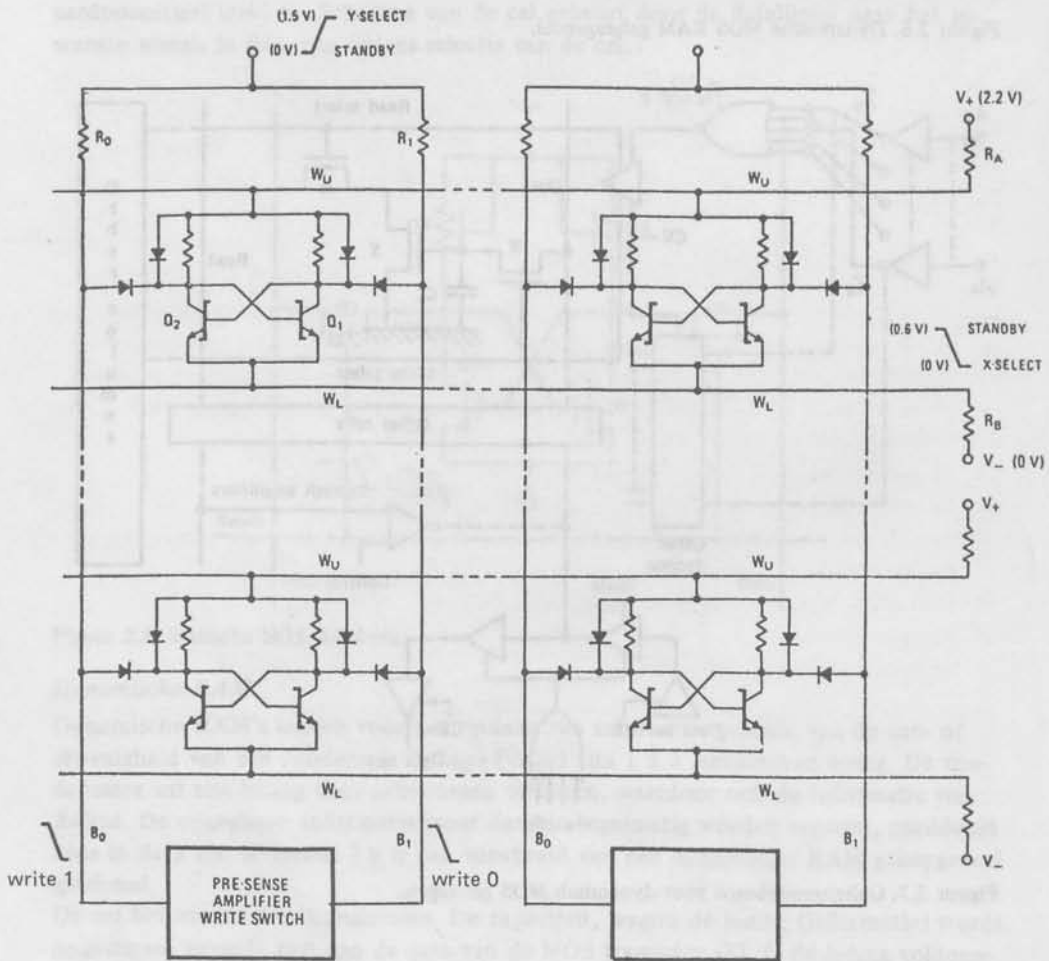


Figuur 2.7. Geheugenopbouw voor dynamisch MOS geheugen.

Door sturing van de write-select kan men via schakelaar Q1 lading op de condensator C aanbrengen of weghalen. De geheugencellen worden in een matrix geplaatst. Een geheugenmatrix van 1024 bits is in figuur 2.7 getekend. De adreslijnen A_0 t/m A_4 bepalen welke read/write selectdraden worden gestuurd, dus rijselectie, en de adreslijnen A_5 t/m A_9 bepalen in welke kolom wordt geschreven of gelezen. In de geheugencel, op het kruispunt van de rij en de kolom, vindt de gewenste actie plaats. Om het geheugen te kunnen verversen is er per kolom een refresh-versterker nodig. Het verversen gebeurt per rij door het 'laag' maken van de read-select en het sturen van de besturingslijn van de refresh versterker. Een refresh-cyclus bestaat verder uit het sequentieel sturen van de read-select-lijnen via de adresingangen A_0 t/m A_4 .

Bipolaire RAM's

In figuur 2.8 zien we een matrix met bipolaire geheugencellen. De cellen bestaan uit flip-flops, die zijn opgebouwd uit 2 transistoren, 2 weerstanden en 4 dioden. Een cel kunnen we selecteren door de draad W_L (X-select) 'laag' en de Y-select 'hoog' te maken. Van elke geheugencel zal steeds een transistor geleiden. De geleidende transistor zal, wanneer de cel geselecteerd is, de weerstand R_0 of R_1 zodanig belasten, dat lijn



Figuur 2.8. Matrix van bipolaire cellen voor een RAM.

B_0 of B_1 'laag' wordt. Het verschil tussen B_0 en B_1 geeft (in de read mode) aan of de cel de informatie '1' of '0' bevat. De cel is van nieuwe informatie te voorzien door sturing van de lijnen B_0 of B_1 .

RAM gebruik

In microcomputers worden de RAM's voornamelijk gebruikt voor het bewaren van tussentijdse resultaten van berekeningen en testen, het zogenaamde kladgeheugen. Verder gebruiken we de RAM's als buffergeheugen bij in- en uitvoer van informatie naar de buitenwereld. In bepaalde gevallen kan een RAM ook dienen voor het opslaan van programma's, die vanuit een achtergrondgeheugen worden opgehaald (ponsband, cassette recorder, diskette, enz.), onder andere voor het ontwikkelen en testen van nieuwe programmatuur of voor het draaien van bestaande programma's analoog aan het gebruik bij minicomputers. Daar de RAM's vluchtige (volatile) geheugens zijn, zal na iedere keer dat de voedingsspanning is weggefallen, de informatie opnieuw in de RAM moeten worden ingelezen.

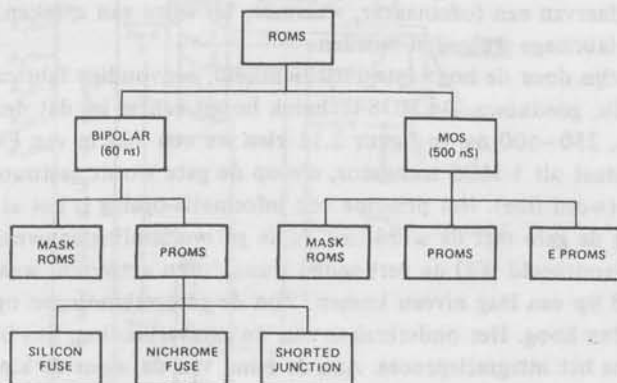
2.4.2. Read Only Memory (ROM)

Een microcomputer heeft een vast systeemprogramma nodig om na het inkomen van de voeding te kunnen starten. Iedere microcomputer zal daarom een aantal systeem- en monitorprogramma's in een niet vluchtig geheugen (non-volatile memory) moeten hebben staan. Voor deze non-volatile geheugens worden meestal ROM's (Read Only Memories) gebruikt.

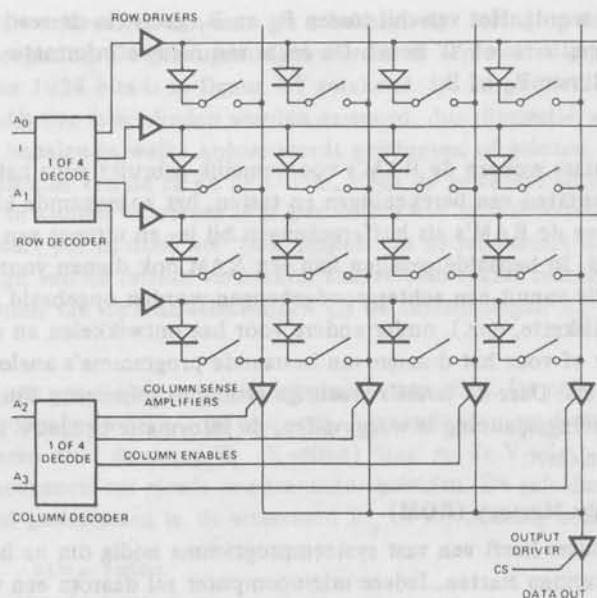
Een ROM is net als een RAM willekeurig toegankelijk, alleen de schrijffunctie ontbreekt. De informatie wordt op een aparte wijze in de ROM gebracht.

Welke typen ROM's veel worden toegepast is in figuur 2.9 aangegeven. In dit schema vinden we een onderscheid in technologie en toepassing.

Evenals bij de RAM zijn er ROM's waarbij de geheugencellen in een matrix zijn geplaatst (figuur 2.10). De adreslijnen A_0 en A_1 selecteren de rijen en de adreslijnen A_2 en A_3 zorgen voor de kolomselectie.



Figuur 2.9. Overzicht van veel toegepaste typen ROM's.



Figuur 2.10. Principe-schema van een ROM.

ROM's

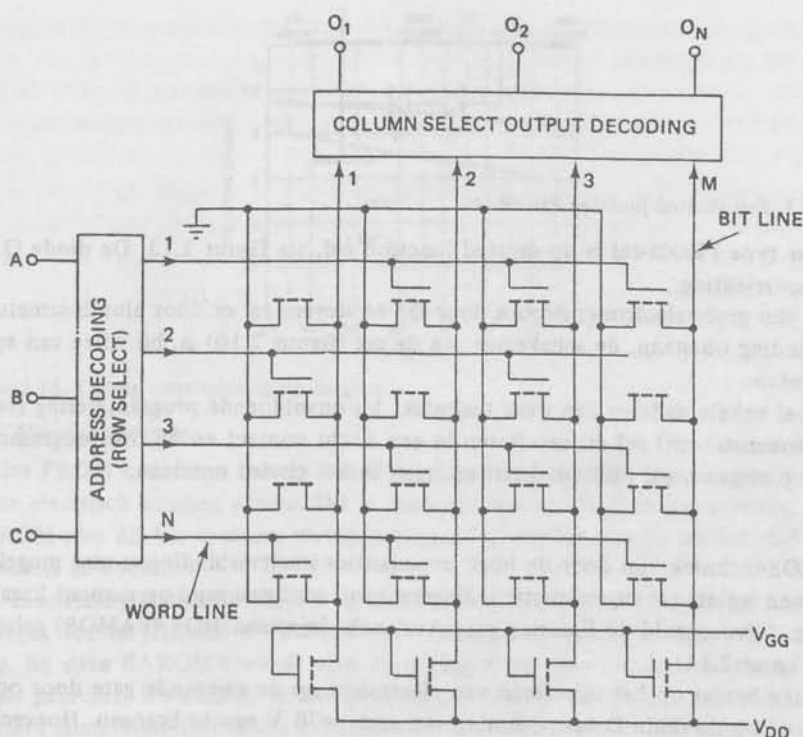
Het principe van de ROM-geheugenfunctie is het al of niet gesloten zijn van het schakelcontact in de diode-keten, die een cel vertegenwoordigt. Hier zal een gesloten keten een logische '1' en een open keten een logische '0' voorstellen. De technologie bepaalt, hoe deze contacten zijn verwezenlijkt.

De snelste geheugens (10–40 ns) zijn bipolair. Tot de goedkoopste ROM's behoren de MASK ROM's, zeker voor grote aantallen met elk dezelfde inhoud. Bij een MASK ROM geeft de klant op welke informatie de ROM moet hebben en de fabrikant maakt aan de hand daarvan een fotomasker, waarmee, bij wijze van spreken, de gesloten contacten bij de fabricage weggeëtst worden.

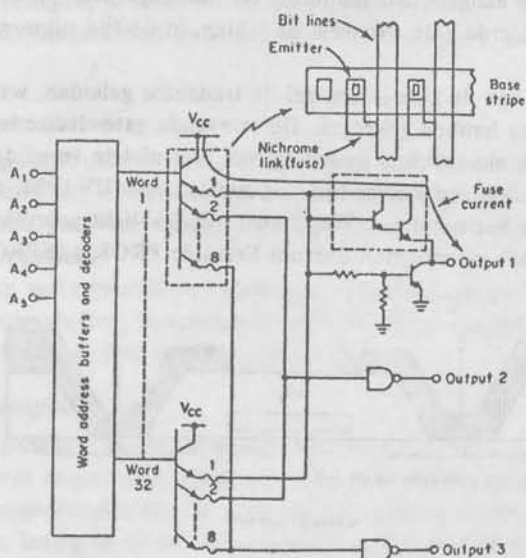
MOS ROM's zijn door de hoge integratiedichtheid, eenvoudige fabricage en gering energieverbruik, goedkoop. De MOS-techniek houdt echter in, dat deze ROM's relatief langzaam zijn, 250–600 ns. In figuur 2.11 zien we een matrix van PMOS ROM-cellen. Iedere cel bestaat uit 1 MOS transistor, die op de gate wordt gestuurd door de rij-selectiedraad (word line). Het principe van informatie-opslag is het al of niet doorverbinden van de gate met de word line. Is de gateverbinding aanwezig, dan zal de word line (bijvoorbeeld W2) de verbonden transistoren activeren, waardoor de kolomdraden 1 en 3 op een laag niveau komen. Zijn de gateverbindingen open, dan blijven de kolomdraden hoog. Het onderbreken van de gateverbinding, dus het programmeren, gebeurt tijdens het integratieproces. Aan de hand van de, door de klant gewenste, informatie maakt de fabrikant een masker voor het al of niet aanbrengen van de verbinding.

PROM's

Naast de genoemde ROM's zijn er nog de bipolaire PROM's (Programmable ROM's), die de gebruiker zelf kan voorzien van informatie. Dit gebeurt door in de geheugen-cellen, waar een logische '1' moet komen (figuur 2.12), een verbinding 'op te blazen'. We zien dat deze PROM is opgebouwd uit 32 woorden van 8 bits. Een cel wordt ge-

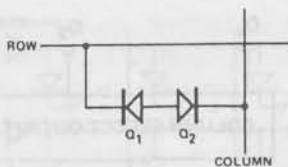


Figuur 2.11. Opbouw PMOS ROM.



Figuur 2.12. Een bipolaire fusible link PROM.

wordt door een transistor met een smeltverbinding in de emitter. Door deze verbinding door te smelten, geven we de cel de logische waarde '1'. Het programmeren van de PROM gebeurt door de gewenste informatie van een geselecteerd adres op de uitgang aan te brengen en gedurende een bepaalde tijd een pulstrein van steeds bredere stroompulsen door de smeltverbinding te sturen.



Figuur 2.13. Een shorted junction PROM-cel.

Een ander type PROM-cel is de shorted junction cel, zie figuur 2.13. De diode Q1 staat in sperrichting.

Door nu een grote elektronenstroom door Q1 te sturen, zal er door aluminiumatomen een verbinding ontstaan, de schakelaar van de cel (figuur 2.10) is, bij wijze van spreken, gesloten.

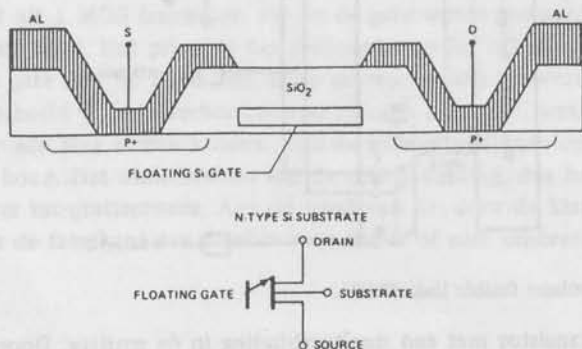
Er zijn wel enkele nadelen aan deze techniek: bij onvoldoende programmering (te weinig elektronenstroom) zal dit resulteren in een slecht contact en bij overprogrammering kunnen op ongewenste plaatsen kortsluitingen in het circuit ontstaan.

EPROM's

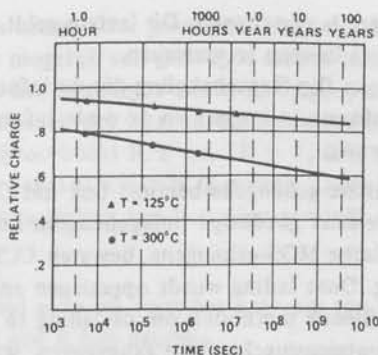
Bij de MOS-techniek zijn door de hoge impedanties smeltverbindingen niet mogelijk. Wel kunnen we als geheugenfunctie ladingtransport met een semi-permanent karakter toepassen, bijvoorbeeld de Floating gate Avalanche-injection MOS (FAMOS) geheugen-cel (zie figuur 2.14).

De werking berust op het injecteren van elektronen op de zwevende gate door op de source S of op de drain D een spanning van circa -30 V aan te brengen. Hoeveel lading op deze wijze op de zwevende gate wordt aangebracht, hangt af van de amplitude en de tijdsduur van de aangeboden spanning. Na het weghalen van de spanning blijft de lading op de geïsoleerde gate. Hoeveel de lading, in de tijd uitgezet, vermindert, is in figuur 2.15 te zien.

Door het aanbrengen van de gate-lading zal de transistor geleiden, waardoor we weer een gesloten schakelaar hebben gekregen. De zwevende gate-electrode heeft geen galvanische verbindingen, elektrisch is de lading dan ook niet te verwijderen. Maar door de FAMOS-cel gedurende een zekere tijd te bestralen met UV-licht, zal de lading van de gate afvloeien naar het substraat. We kunnen dit UV-licht gebruiken om de PROM te wissen. Deze PROM's noemt men daarom Erasable PROM's (EPROM's).



Figuur 2.14. FAMOS geheugencel.



Figuur 2.15. Lading vermindering op de gate.

Read Mostly Memories

Behalve PROM's, die met behulp van UV-licht worden gewist, zijn er ook PROM's, die we elektrisch kunnen wissen. Dit is duidelijk een voordeel in het gebruik, omdat de PROM niet uit het systeem verwijderd hoeft te worden om de PROM van nieuwe informatie te voorzien.

Deze Electrically Alterable ROM's (EAROM's) hebben echter een relatief lange toegangstijd, 350 ns (Hitachi 48016) tot 5 μ s. Voor het wissen zijn vaak 1 à 2 seconden nodig. Bij deze EAROM's wordt voor de ladingopslag (dus informatie-opslag) de gate-isolatie gebruikt. We kunnen de cel programmeren door op de gate een positieve of negatieve spanningspuls (10 ms à 20 μ s) aan te brengen, die een bepaald kritisch polarisatieniveau te boven gaat. Door op de gate een spanningspuls met een tegengestelde polariteit aan te brengen, kunnen we de geheugencel wissen.

Gezien het gebruik, noemt men deze geheugens Read Mostly Memories (RMM's).

ROM-gebruik

ROM's worden gebruikt voor:

- code-omzetting, bijvoorbeeld het omzetten van ASCII naar de IBM Selectric code voor het besturen van elektrische schrijfmachines of het omzetten van ASCII naar een 5×7 matrix patroon, waarmee we karakters op een LED-display of een CRT-display kunnen zetten,
- het opslaan van vaste gebruikersprogramma's of microprogramma's,
- het bewaren van tabellen, bijvoorbeeld een sinustabel, waarmee snel de sinuswaarde van een bepaalde hoek kan worden opgezocht.

2.4.3. Seriële geheugens

Een laatste type geheugen is het seriegeheugen, gebaseerd op het schuifregisterprincipe, de informatie wordt sequentieel ingeklokt. We kennen statische en dynamische schuifregisters. De geheugenfunctie van de dynamische registers wordt gevormd door een, al of niet aanwezige, lading in de schuifregistersectie. Hierbij kan ladingverlies optreden, reden waarom de inhoud regelmatig dient te worden verversd. Een methode voor het verversen is, na de registers van informatie te hebben voorzien, de ingang met de uitgang te verbinden en de informatie continu te laten rondschuiven. Het aantal gegeven klokpulsen is een referentie voor het adres.

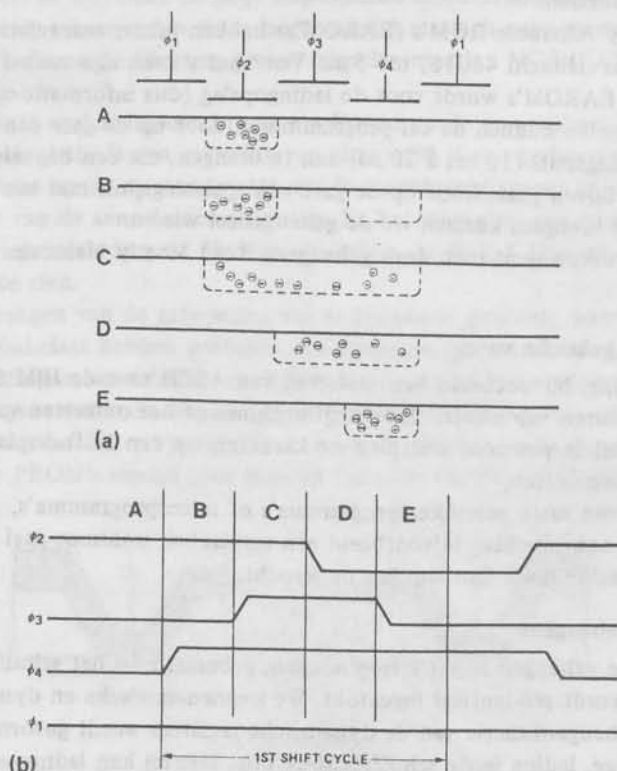
Seriegeheugens zijn niet 'random accessible' maar kunnen voor de buitenwereld schijnbaar willekeurig toegankelijk gemaakt worden. Dit vraagt wel een langere toegangstijd, er moet immers worden gewacht tot de informatie van het gewenste adres

aan de uitgang van het register is verschenen. Dit laatste geldt ook voor de statische registers, die we overigens niet hoeven te verversen.

De statische geheugencel is een flip-flopschakeling die de informatie bewaart tot een volgende klokpuls nieuwe informatie invoert en de oude informatie overbrengt naar de volgende cel, indien aanwezig.

Tot de groep seriele dynamische geheugens behoort ook het Charge Coupled Device (CCD) geheugen, waarmee relatief goedkope bulkgeheugens kunnen worden samengesteld. Evenals bij de dynamische MOS-geheugens, bewaren CCD-geheugens de informatie in de vorm van lading. Deze lading wordt opgeslagen en verplaatst langs, of in het substraat. Er zijn verschillende methoden om de lading te verplaatsen, zo is in figuur 2.16 een ladingsverplaatsingsmechanisme aangegeven, waarbij vier klokpuls-electroden worden toegepast.

In de tijdsperiode A is alleen de ϕ_2 -gate op een hoog potentiaal, waardoor eventuele aanwezige lading zich daaronder verzamelt. Door de gates als in figuur 2.16b aan te sturen, kunnen we de eventuele lading van gate ϕ_2 naar gate ϕ_4 transporteren en vervolgens van ϕ_4 naar ϕ_2 van de volgende set gates. De CCD structuur is dynamisch en de informatie dient regelmatig te worden ververs.



Figuur 2.16. Ladingtransport bij CCD-geheugens.

2.5. Opbouw van een geheugensysteem

2.5.1. Inleiding

Voor de gewenste geheugenuitvoering is meestal geen kant en klare chip verkrijgbaar, men zal het geheugensysteem moeten opbouwen uit een aantal verschillende geheugen-

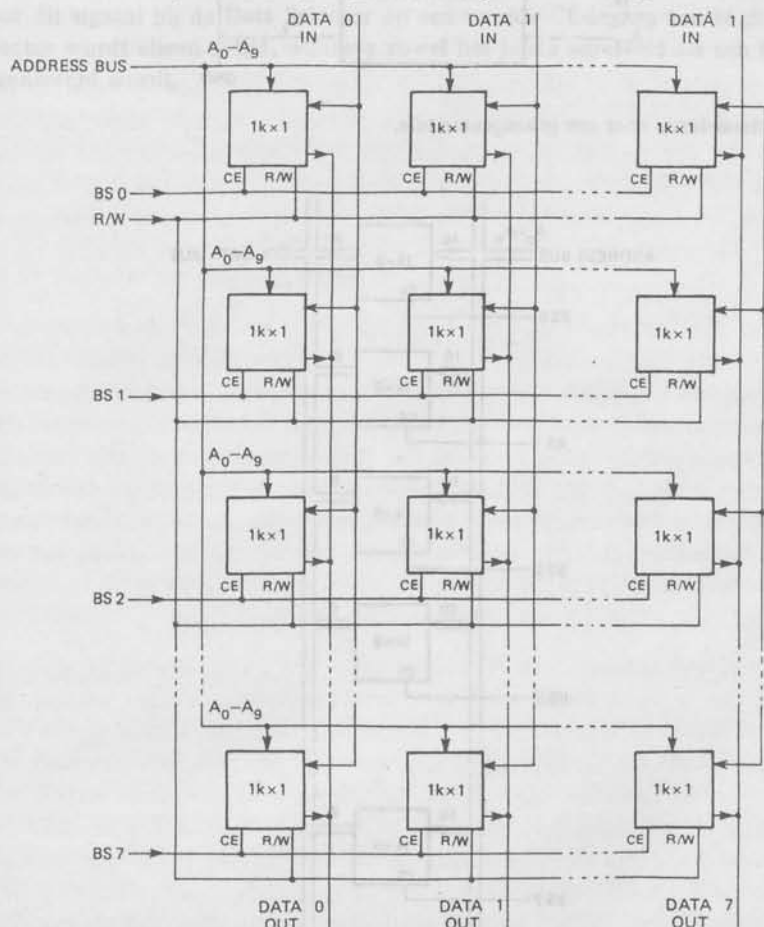
modulen, die gevormd worden door geheugen-IC's, interface-, selectie- en besturingslogica. Het is zodoende mogelijk een geheugen samen te stellen, dat voldoet aan de vereiste woordlengte en het benodigde aantal geheugenplaatsen.

De woordlengte kan verwezenlijkt worden door een aantal geheugenchips parallel te nemen, gebruiken we bijvoorbeeld IC's van $1K \times 1$, dan hebben we 8 IC's parallel nodig om een geheugen van $1K \times 8$ bits te maken. Door dit met een factor m te vermenigvuldigen wordt de capaciteit van het geheugen $mK \times 8$.

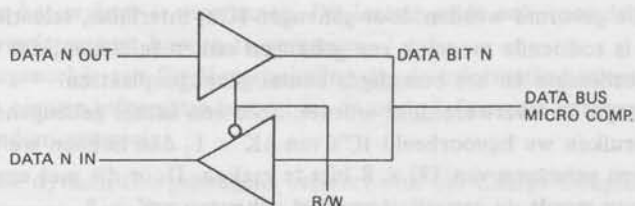
2.5.2. Geheugenmodule

In het volgende voorbeeld gaan we voor het ontwerp van een geheugenmodule uit van een 16 bits adresbus en een 8 bits databus.

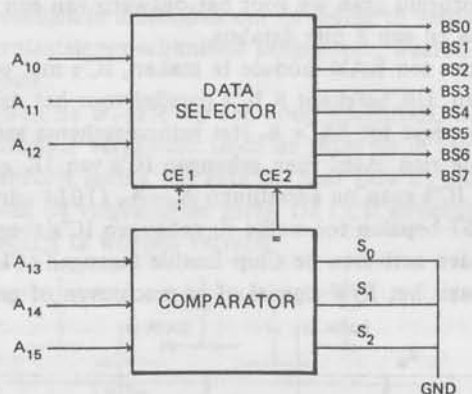
Neem aan, dat er, om een RAM-module te maken, IC's met een capaciteit van $1K \times 1$ ter beschikking staan. Dit betekent 8 IC's parallel voor het datapad. De capaciteit van de module wordt beperkt tot $8K \times 8$. Het prinsipschema van figuur 2.17 laat de opbouw van de module zien. Acht rijen geheugen IC's van $1K \times 1$ bit. Iedere rij is 8 IC's lang. Naar alle IC's gaan de adreslijnen A_0-A_9 (1024 adressen). De bankselectdraden BS0 t/m BS7 bepalen tot welke rij geheugen IC's toegang wordt gevraagd. Deze bankselectdraden activeren de Chip Enable ingangen (CE) van de geselecteerde rij IC's. Verder bepaalt het R/W signaal of er geschreven of gelezen wordt. De hier



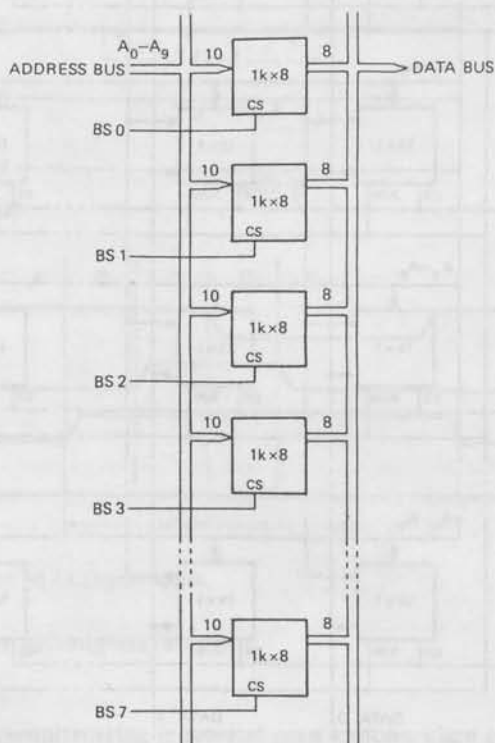
Figuur 2.17. Opbouw van een RAM-geheugenmodule.



Figuur 2.18. Interface geheugenmodule met de databus.



Figuur 2.19. Adresselectie voor een geheugenmodule.



Figuur 2.20. Opbouw van een ROM-geheugenmodule.

gebruikte IC's hebben een aparte in- en uitgang voor de data. Er bestaan ook circuits, waarbij de data over één in/uitgang getransporteerd wordt. Het R/W signaal bepaalt daarbij of het kanaal als in- of uitgang functioneert.

De microcomputer beschikt meestal over een bidirectionele databus. Om deze geheugenmodule te laten samenwerken met de databus zijn busdrivers nodig (zie figuur 2.18). Het R/W signaal bepaalt in welke richting de drivers actief zijn.

De bankselectsignalen zijn afkomstig van de *adreselector*. De adreselector bepaalt tevens in welk adresveld van het geheugenbereik de module werkzaam is. Een voorbeeld van een adreselector is te vinden in de selectieschakeling van figuur 2.19. De Data Selector levert de signalen BS0–BS7 en activeert één van deze 8, afhankelijk van de adreslijnen A_{10} – A_{12} en de Chip Enable.

In welk adresveld we de module willen plaatsen kan ingesteld worden met de schakelaars S_0 t/m S_2 .

Komt de ingestelde waarde overeen met de waarde van de adreslijnen A_{13} t/m A_{15} , dan zal de comparator de Data Selector activeren. Eén van de bankselectdraden wordt actief en selecteert de gewenste rij geheugen-IC's om te lezen of te schrijven. Een vergelijkbare opzet is te maken voor een ROM-module, waarbij in het voorbeeld van figuur 2.20 ROM's met een capaciteit van $1K \times 8$ gebruikt worden. Daar de ROM alleen gelezen kan worden kunnen we in de selectie ook het R/W signaal betrekken door dit signaal bij de Data Selector op een tweede CE-ingang aan te sluiten. De selector wordt alleen actief, wanneer zowel het juiste adresveld als een leesopdracht gegenereerd wordt.

de spronginstructies naar een subroutine kunnen conditioneel zijn. Het al of niet uitvoeren van de sprong gaat dan analoog aan de conditionele spronginstructies. Verder kennen de meeste processors nog instructies om de inhoud van een register op de stapel te zetten. Dat gaat met een PUSH-opdracht. Het register is daarna vrij bruikbaar omdat de inhoud tijdelijk is weggezet. Na afloop van het gebruik van het register moet de originele inhoud weer teruggezet worden in het register, dit gebeurt met een POP-instructie. Ook de conditievlaggen, die de waarde van de conditiecodes weer geven zijn vaak op de stapel te zetten.

Voorbeelden: CALL, CALL COND=TRUE, CALL COND=FALSE, PUSH REG,
PUSH FLAGS, POP REG, POP FLAGS.

De *IN- en UITVOER-instructies*, waarmee uitwisseling van informatie tussen processor en buitenwereld mogelijk is. Het betreffende apparaat moet worden geselecteerd en de gegevens overgedragen. Daartoe moet de instructie worden voorzien van een I/O-adres en de specificatie van een register waar de gegevens heen moeten of vandaan komen.

Voorbeelden: INPUT REG, I-ADDRESS,
OUTPUT REG, O-ADDRESS.

Verder is er nog een groep instructies die gevangen wordt onder de kop *MISCELLANEOUS*. Hieronder valt bijvoorbeeld de instructie HALT, waarmee de processor wordt stilgezet en de instructie NOP, een no-operation instructie. Ook instructies die te maken hebben met de interrupt vallen onder deze groep. Voor een nette verwerking van bepaalde stukken programma is het soms nodig de processor 'doof' te kunnen maken voor optredende interrupts. Daarin voorzien de instructies DISABLE INT en ENABLE INT, respectievelijk om de interrupt 'af' en weer 'aan te zetten'. Verder nog alle mogelijke instructies die een fabrikant als 'extra' aanprijst en die dan sterk gebonden zijn aan het type processor dat diezelfde fabrikant levert. Hoewel er hier zeer nuttige instructies bij zijn, hebben we met de indeling volgens de bovengenoemde groepen zeker 90% van de algemeen voorkomende instructies te pakken.

De uitbreiding met speciale instructies hangt ten nauwste samen met de keuze die de fabrikant maakt voor de hoeveelheid bits in een woord (meestal een byte, dus 8 bits) die bestemd zijn voor het specificeren van de operatiecode. Dit gekoppeld met het aantal interne registers geeft dan bijvoorbeeld de volgende situatie: 5 bits operatiecode en 3 bits voor aanwijzen van de, bijvoorbeeld 8, interne registers. Dan ontstaan er 32 verschillende operatiecodes of instructies.

Kiest dezelfde fabrikant voor een 8 bits operatiecode, dan geeft dit 256 verschillende operaties, dus 256 verschillende instructies. Hij zal dan (zeker) proberen er zoveel mogelijk te gebruiken, wat kan leiden tot gezochte instructies en dus ook een kromme instructieset.

3.2. Conditie flip-flops

De conditie flip-flops zijn een aantal flip-flops in de processor, die een waarde krijgen (0 of 1), afhankelijk van het resultaat van bewerkingen die in de processor plaatsvinden. De waarde van de flip-flops geeft dus een idee over de interne toestand van de processor, daarom worden ze ook wel aangeduid als *toestandsvlaggen*. De bewerkingen die een invloed op de conditie flip-flops hebben, zijn de rekenkundige en logische bewerkingen die met de ALU gebeuren. Een spronginstructie of een instructie uit de groep MOVE-instructies laat de conditie flip-flops ongemoeid. Bij de beschrijving van een microprocessor zal de fabrikant bij iedere instructie opgeven of deze de conditie flip-flops beïnvloedt en zo ja, welke.

Met een conditionele instructie is het nu mogelijk om een verder verloop van het programma te laten afhangen van de waarde van een bepaalde conditie flip-flop. Een veel voorkomende toepassing is bijvoorbeeld de *loop*, er wordt een lus doorlopen totdat aan een voorwaarde is voldaan. Als voorwaarde fungeert de inhoud van één van de conditie flip-flops.

De meest voorkomende toestandsvlaggen zijn:

- De *zero* vlag, deze vlag wordt op '1' gezet als het resultaat van de bewerking nul oplevert en wordt of blijft '0' als het resultaat ongelijk nul is.
- De *sign* vlag, deze vlag geeft aan of het resultaat van de bewerking een negatief getal is, wordt dan '1', en is '0' als het resultaat een positief getal is. Deze vlag heet *sign* omdat het niets anders is dan een kopie van het meestwaardige bit van het resultaat, het *sign bit* als we werken met negatieve getallen in de two's complement notatie.
N.B.: Ook als het programma niet met (negatieve) getallen werkt, wordt dit bit beïnvloed alsof de gegevens wel getallen zijn.
- De *carry* vlag, een vlag die aangeeft, door '1' te worden, dat bij een rekenkundige bewerking een overloop (carry) is opgetreden. Andere rekenkundige of logische bewerkingen en het niet-optreden van een overloop resulteren in het '0' worden van deze vlag. Bij optellingen is nu simpel te testen of het resultaat van de optelling nog 'past' in de woordbreedte waarmee wordt gewerkt. Overigens kunnen bepaalde schuifinstructies (SHIFT of ROTATE) ook een invloed op de waarde van het carry bit hebben.
- De *parity* vlag, deze vlag krijgt na een bewerking een zodanige waarde dat het totaal van vlag en resultaat een (meestal) even pariteit heeft. Kent bijvoorbeeld het resultaat 3 enen dan wordt de parity vlag een '1', zitten er 6 enen in het resultaat dan wordt de parity vlag een '0'.
- De *half carry* vlag (intermediate carry). Veel typen microprocessors laten, beperkte, bewerkingen toe op getallen in BCD-notatie. In een 8-bits woord staan dan steeds twee decimale cijfers, voorgesteld door de hoogste vier en de laagste vier bits in het woord. Voeren we nu bewerkingen uit op getallen die BCD zijn gecodeerd, dan moet er ook op worden gelet of er een overloop optreedt vanuit het laagste 4-bitsgedeelte. Daar dient de half carry vlag voor, deze wordt '1' als er zo'n overloop optreedt. Programmatisch kan hier op worden getest en indien nodig, het hoogstwaardige gedeelte worden bijgewerkt.

3.3. Adresseringsmethoden

3.3.1. Plaats van de operand

Naast de opdracht die moet worden uitgevoerd, bevat de instructie ook nog een operand, dat wil zeggen datgene *waarop* de operatie moet worden uitgevoerd. Een bit-combinatie in de totale instructie geeft aan welke operand is bedoeld. De mogelijke operanden zijn in drie klassen te verdelen:

- de operand is een register in de processor,
- de operand is een plaats in het geheugen,
- de operand is te vinden in (bijvoorbeeld een register in) een randapparaat.

Ieder van deze klassen heeft zijn eigen gebruiksaspecten. Registers in de processor zijn snel toegankelijk en kunnen met een beperkt aantal bits worden geadresseerd. Geheugenplaatsen moeten met een volledig adres worden aangewezen en zijn met een

geheugencyclus toegankelijk. Randapparaten vragen ook uitgebreidere adressen en wat timing betreft zijn we afhankelijk van de reactiesnelheid van het randapparaat. Adressen van registers en van randapparaten zijn éénduidig, van geheugenelementen kan dat niet gezegd worden. Het adres dat in de instructie als operand adres staat vermeld noemen we het *schijnbare* (virtuele) *adres*. Dit adres kan, voordat het door de processor op de computerbus wordt gezet, een modificatie ondergaan. Het adres, zoals we dat op de bus zien, heet nu het *effectieve adres*. Tenslotte kan het adres nog een aanpassing ondergaan voor het in het centrale geheugen een plaats aanwijst. Dit laatste adres, het 'echte' adres, heet dan het *hardware adres*. Deze adressen kunnen, maar hoeven niet, aan elkaar gelijk te zijn. Tussen twee opvolgende fasen kan een adresmodificatie of -vertaling optreden. Figuur 3.1 laat dit in een blokschema zien.



Figuur 3.1. Blokschema voor een adresmodificatie.

De verschillende manieren waarop we komen van het operand adres tot het uiteindelijke hardware adres heten *adresseringsmethoden* en geven een indruk van de *processing power* van een bepaalde computer. Juist in adresseringsmethoden heeft een enorm sterke ontwikkeling plaatsgevonden gedurende de afgelopen jaren. De huidige microcomputers (en zeker ook de mini's) kennen en kunnen op dit gebied meer dan de 'grote' computers van een tiental jaren terug.

3.3.2. Redenen voor adresmodificatie

Het is natuurlijk veruit het eenvoudigst om, vanuit het operand adres, direct over te gaan naar het hardware adres. Er is dan helemaal geen adresmodificatie nodig en operand adres = hardware adres. Een instructie uit de MOVE groep waarbij een waarde uit een geheugenplaats wordt gehaald en in de accumulator gezet, ziet er dan als volgt uit:

MOVE AC,1984.

Het getal 1984 is het adres van de geheugenplaats waaruit we de inhoud moeten laden in AC. Dit getal is dan het hardware adres.

Hoewel deze methode goed bruikbaar is, en in de meeste computers dan ook voorkomt, is ze in bepaalde gevallen niet erg efficiënt en zeker niet flexibel. Er zijn een drietal redenen aan te voeren waarom en/of wanneer dit speelt.

1. Het is bij deze manier van adresseren altijd nodig om in de instructie voor het adres van de operand evenveel bits te reserveren als het fysische geheugen vereist. Hebben we een geheugen van 64K, dan is voor adressering een adres van 16 bits nodig. Ieder adres van een operand die in een instructie voorkomt, moet dan *ook* uit 16 bits bestaan. Dit gevoegd bij een aantal bits voor de operatiecode, levert een totaal aantal bits per instructie op, dat op minimaal 20 uitkomt. Nu is dit, bijvoorbeeld bij een 8-bits microprocessor, met een 3-bytes instructie wel te doen maar het levert programma's op die een onevenredige hoeveelheid geheugen beslaan en die bij executie ook onnodig veel verwerkingstijd vragen. Verder blijkt in de praktijk dat we maar met een beperkte hoeveelheid operanden werken, zodat een adres van 6 à 8 bits meer dan voldoende zou zijn om deze te adresseren. Alles bij

elkaar levert een volledige specificatie van het operand adres in de instructie een allerminst efficiënte constructie op.

2. Een computer die een N -bits brede adresbus kent, adresseert hiermee 2^N geheugenplaatsen. Bij oudere typen computers was dit nauwelijks een beperking, ook al omdat toen de prijs van het geheugen een wezenlijk deel van de totaalprijs uitmaakte en een uitbreiding van het geheugen al snel niet verantwoord was. In de tegenwoordige computersystemen is de geheugenprijs minder van belang, reden om een groter geheugen aan te schaffen dan direct door de processor te adresseren is. Adressering ervan vereist dan speciale technieken van geheugenbeheer (memory management). Het totale geheugen wordt opgedeeld in blokken en een mechanisme voorziet in de modificatie van een N -bit naar een M -bit adres (met $M > N$).

In bovengenoemd geval zou er dan bij het ontwerp van een computer vanaf het begin mee moeten worden gerekend dat het aantal adres bits in de instructie M moet zijn. Praktisch blijkt dan M altijd één kleiner te zijn dan wenselijk is.

3. In programma's komen veelvuldig stukken voor die zich kenmerken door een bepaalde *lokaliteit*. Dat stuk programma speelt zich af binnen een beperkt gedeelte van het geheugen en nadat, lokaal, een functie is verricht wordt in een ander gedeelte verder gegaan. Als voorbeeld een testlus, er wordt steeds getest op het voorkomen van een bepaalde waarde of toestand, een bepaalde actie genomen en weer teruggegaan naar de plaats waar wordt getest net zolang totdat wel een bevredigend resultaat ontstaat, waarna doorgedaan wordt met het verdere programma. Wat dan gebeurt is, dat er steeds met een spronginstructie wordt teruggesprongen over een beperkt adresbereik. Het is zonde om voor een verschil van bijvoorbeeld 16 adressen toch steeds het volledige sprongadres te moeten opgeven, terwijl 4 bits hadden volstaan.

Ook bij de werking met tabellen vinden we een inefficiëntie als volledige adressen worden opgegeven. We maken dan gebruik van *wijzers* of *pointers* naar de plaats in de tabel waar we moeten zijn. De wijzers zijn vaak zelf ook weer geheugenplaatsen. Al lopend door een tabel heen moet de inhoud van een wijzer steeds worden veranderd. In de opzet met een volledige adresspecificatie betekent dit dat we iedere keer het volledig adres van de wijzer moeten opgeven als we die met 1 willen verhogen of verlagen, zoals dat gebeurt bij het doorlopen van een tabel. Dan hebben we nog maar de helft bereikt van wat we willen, namelijk de waarden uit de tabel. Met de wijzer wordt weer een adres aangegeven en in de inhoud hiervan zijn we geïnteresseerd. Het liefst hebben we een adresseringsmechanisme dat voor ons, via een opgegeven wijzeradres, zelf daarna de inhoud van die wijzer weer als adres gebruikt en daar de inhoud van ophaalt. Dit levert een indirecte manier van adresseren op, namelijk via het adres van een wijzer, het adres van de tabelinhoud.

Bovengenoemde redenen zijn er mede oorzaak van geweest dat computers een aantal vormen van adressering kennen. Hoewel niet iedere computer ook iedere adresseringsmethode kent, rekenen zelfs microprocessors steeds meer soorten adressering tot de mogelijkheden. De verschillende soorten adressering worden in de volgende paragrafen behandeld.

3.3.3. Onmiddellijke adressering (immediate addressing)

Bij onmiddellijke adressering staat, zoals de naam al zegt, de *waarde* van de operand onmiddellijk bij de instructie zelf en maakt deel uit van die instructie. Als voorbeeld mag dienen de instructie

MOVE AC,30.

De operand heeft de waarde 30 en staat in de instructie zelf, zodat hij onmiddellijk bekend is. Bij de executie van de instructie wordt na het ophalen van de instructie (de instruction fetch) direct de operand ook opgehaald. Bij een 8-bits microprocessor betekent dit, dat op de geheugenplaats direct volgend op de plaats waar de instructie staat, de waarde 30 te vinden is. Het geheugen ziet er dan als volgt uit:

adres n	: Instructiecode	/ code voor MOVE AC,
adres n + 1	: 30	/ waarde van de operand
adres n + 2	: Instructiecode	/ volgende instructie.

3.3.4. Directe adressering (direct addressing)

Bij deze adresseringsvorm bevat de instructie ook het *adres* van de operand. Deze vorm van adressering is de meest algemene en komt dan ook bij alle typen computers voor. Omdat het volledige adres van de operand is gespecificeerd wordt deze vorm ook wel aangeduid als *absolute adressering*. Het adres van de operand kent meer bits dan de woordbreedte van de processor. Voor een 8-bits processor betekent dit dat de totale instructie dan 3 woorden in het geheugen vraagt. Neem als voorbeeld de instructie:

MOVE AC,1984.

In het geheugen staat dan:

adres n	: Instructiecode	/ code voor MOVE AC,
adres n + 1	: 19	/ eerste adresdeel
adres n + 2	: 84	/ tweede adresdeel
adres n + 3	: Instructiecode	/ volgende instructie
.	.	.
adres 1984	: 30	/ waarde van operand

Na het ophalen van de instructiecode, wordt in een tweetal slagen het adres van de operand opgehaald. Dit adres wordt geplaatst in het *memory address register* (zie figuur 1.2), dan aangeboden aan de adresbus en de binnenkomende data is de operand en komt in AC terecht.

Voor een 8-bits microprocessor betekent dit $4 \times$ een geheugencyclus, namelijk $3 \times$ voor de instructie en $1 \times$ voor de operand.

3.3.5. Indirecte adressering (indirect addressing)

Bij indirecte adressering hebben we een opbouw van de instructie die er hetzelfde uitziet als bij directe (of absolute) adressering. Het adres dat nu in de instructie is vastgelegd wijst echter niet direct naar het adres van de operand, maar naar een (verwijs)-adres waarin het adres van de operand te vinden is. Neem als voorbeeld de instructie:

MOVE *AC,1984

(* betekent: *indirect*).

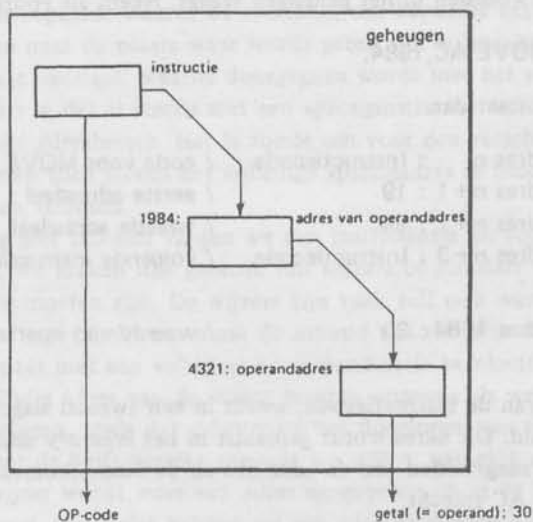
In het geheugen staat:


```

adres n      : Instructiecode / code voor MOVE *AC,
adres n+1 : 19                / eerste adresdeel
adres n+2 : 84                / tweede adresdeel
adres n+3 : Instructiecode / volgende instructie
.
.
.
adres 1984 : 43                / eerste deel operandadres
adres 1985 : 21                / tweede deel operandadres
.
.
.
adres 4321 : 30                / waarde van operand

```

Ten opzichte van de directe adressering houdt dit twee extra geheugencycli in, namelijk het ophalen van het adres van de operand. Bij (micro)processors waar in de instructie niet voldoende bits zijn gereserveerd om het gehele geheugen *direct* te adresseren, kan dit door *indirecte* adressering wel worden bereikt. Een andere toepassing van indirecte adressering vinden we bij het werken met tabellen zoals vermeld in paragraaf 3.3.2 onder punt 3. Een schematische aanduiding van de indirecte adressering geeft figuur 3.2.



Figuur 3.2. Schematische aanduiding van de indirecte adressering.

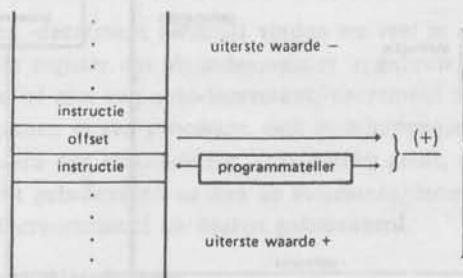
In principe is er geen reden om deze manier van indirectie niet over meer plaatsen voort te zetten zodat 'het adres van het adres van het adres . . .' wordt genomen. Sommige computers laten deze manier van adresseren toe, maar bij het nut van een dergelijke constructie kunnen we de nodige vraagtekens plaatsen. Bij deze laatste manier van adresseren moet dan wel in het adresdeel met een bit worden aangegeven of indirecte adressering gewenst is of niet. Als in de instructie zelf een bit hiervoor is gereserveerd, dan kan indirectie maar éénmaal worden toegepast.

3.3.6. Relatieve adressering (relative addressing)

Ook bij relatieve adressering is het aantal bits in de instructie, dat het adres van de operand aanwijst, beperkt en wordt een uiteindelijk adres bepaald ten opzichte van een referentiewaarde. Voor de referentiewaarde zijn er een aantal mogelijkheden.

Veelvoorkomend is het gebruik van de *programmateller* (PC) als referentie. In de instructie is bijvoorbeeld een byte opgenomen als offset-waarde; bij de waarde van de PC, na uitvoer van de instructie, wordt de waarde van het byte opgeteld en de uitkomst is het adres van de operand. Door de waarde van het byte in het two's complement te noteren wordt een relatieve verplaatsing ten opzichte van de PC bereikt die loopt van -128 tot +127. Deze methode voldoet goed omdat in veel gevallen de te bereiken geheugenplaatsen niet ver van de lopende instructie afliggen, bijvoorbeeld bij sprong-instructies.

Figuur 3.3 geeft schematisch aan, hoe relatieve adressering ten opzichte van de programmateller verloopt.



Figuur 3.3. Schematische aanduiding van relatieve adressering.

Bij bepaalde typen computers komt een opdeling voor van het geheugen in zogenaamde pagina's. De woordbreedte van de computer is dan niet voldoende om het totale geheugen direct te adresseren en het geheugen wordt dan verdeeld in pagina's van een bepaalde grootte. Binnen een pagina kan dan met minder bits worden gewerkt om een plaats aan te wijzen; zo kan in pagina's van bijvoorbeeld 256 woorden met 8 adresbits worden volstaan. In zo'n geval kan er ook gebruik worden gemaakt van een relatieve adressering, waarbij als referentie het begin van een bepaalde *pagina* wordt genomen. Deze relatieve adressering ten opzichte van een pagina kent een drietal varianten.

■ Relatieve adressering ten opzichte van *pagina nul*.

Dan wordt steeds de onderste pagina geselecteerd. Met deze adresseringsmethode kan men vanuit een willekeurige pagina een adres in pagina nul bereiken. Deze pagina krijgt in de programmering dan ook een bijzondere behandeling en zal met name gebruikt worden voor het opslaan van constanten, tussenresultaten, enz. Het effectief adres wordt gevormd door het opgegeven, relatieve adres aan de *bovenkant* aan te vullen met nul.

■ Relatieve adressering ten opzichte van *de lopende pagina*.

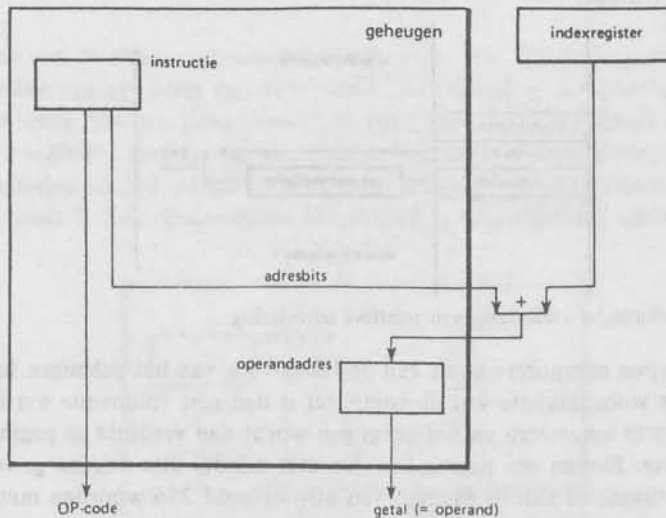
Bij deze vorm van adresseren is de referentie gelijk aan het begin van de pagina waarin men op dat moment aan het werk is. Dit is gedaan met het idee dat men vaak zal werken met variabelen die in de werkomgeving te vinden zijn. Bij kleine pagina's is dit niet altijd even efficiënt, anderzijds verplicht het wel tot meer modulair programmeren. Het effectief adres ontstaat nu door het opgegeven, relatieve adres aan te vullen met het paginanummer.

■ Relatieve adressering ten opzichte van *een willekeurige pagina*.

Sommige typen microprocessors laten het toe om een willekeurige pagina te kiezen als degene waar men een geheugenreferentie wil doen. De processor is daartoe uitgerust met een paginaregister. Dit register kan door een instructie met een willekeurige waarde worden gevuld. Deze manier omvat de beide voorgaande methoden en is het meest flexibel. Het effectief adres ontstaat nu door de inhoud van het paginaregister aan te vullen met het relatieve adres.

3.3.7. Geïndexeerde adressering (indexed addressing)

Bij deze vorm van adressering maakt men gebruik van een speciaal register, dat dan ook *indexregister* heet. Soms is dit een alleen voor deze toepassing toegevoegd register en in andere gevallen mag ieder, in de CPU voorkomend register, als indexregister gebruikt worden. Ook bij geïndexeerde adressering bevat de instructie maar een beperkt aantal adresbits. Het effectieve adres ontstaat door deze adresbits op te tellen bij de inhoud van het indexregister, zie figuur 3.4. In plaats van indexregister komt ook wel de benaming *basisregister* (en *base addressing*) voor. Deze manier van adresseren heeft vooral nut bij tabellen.



Figuur 3.4. Adresseren met behulp van een indexregister.

Het indexregister wordt gevuld met het beginadres van de tabel en door het indexregister steeds te verhogen met '1' worden alle elementen achtereenvolgens geadresseerd.

N.B.: Als we ons in plaats van het indexregister de PC denken, dan komt deze vorm van adresseren overeen met relatieve adressering ten opzichte van de programmateller.

3.3.8. Auto-increment en -decrement

In de voorgaande paragrafen is al een aantal malen het werken met tabellen genoemd. Het komt nu eenmaal vaak voor dat een hoeveelheid gegevens achtereenvolgens bewerkt moet worden en dat leidt al snel tot een tabelvorm. Bij tabellen moet dan steeds voor een volgend element uit de tabel het adres met '1' worden verhoogd. Onder *indirecte adressering* is al aangegeven hoe dat kan. Niettemin bleef er een behoefte bestaan om een 'automatische' aanpassing van het volgende adres te krijgen zonder iedere keer zelf een optelling te verrichten. Dit is bereikt door de *auto-increment en -decrement*. Dit mechanisme komt voor in samenwerking met de *indirecte adressering*. Er wordt in de instructie meegegeven of de wijzer (inhoud van een register of geheugenplaats) die wordt gebruikt voor de adressering bij elk gebruik automatisch met '1' (soms '2') verhoogd of verlaagd moet worden. In feite vinden er dus steeds twee operaties plaats bij dergelijke instructies: het indirect adresseren van een computerwoord en het wijzigen van de inhoud van de wijzer. Welke van deze operaties het eerst gebeurt, is niet altijd gelijk. We kennen vier mogelijkheden:

1. Indirect auto-increment, eerst wordt de operand opgehaald en daarna de wijzer verhoogd (post-increment).
2. Auto-increment indirect, eerst wordt de wijzer verhoogd en daarna de operand opgehaald (pre-increment).
3. Indirect auto-decrement, eerst de operand ophalen en daarna de wijzer verlagen (post-decrement).
4. Auto-decrement indirect, eerst de wijzer verlagen en dan de operand ophalen (pre-decrement).

Gelukkig is het wel zo, dat een bepaald type computer *of* de mogelijkheden 1 en 4 kent *of* de combinatie 2 en 3 en niet alle vier.

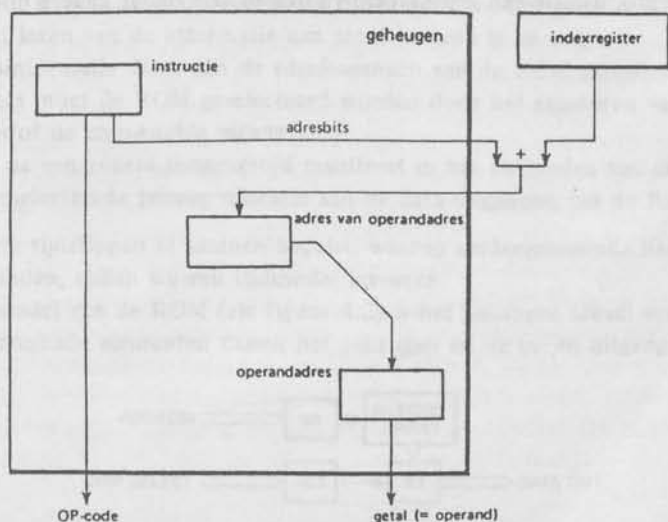
De auto-increment en -decrement faciliteit vinden we veel in combinatie met geïndexeerde adressering. Het register dat als indexregister in gebruik is heeft dan (programma-tisch) de optie om al of niet een auto-increment/-decrement te verrichten. Als we een indexregister tegenkomen in een processor, ook in microprocessors, dan is deze combinatie eerder standaard dan uitzondering. Ook hierbij geldt, afhankelijk van het type processor, eerst wordt geïndexeerd en dan de increment/decrement of eerst wordt geïncrementeerd/geedecrementeerd en daarna geïndexeerd.

3.3.9. Combinaties van adresseringen

De mogelijkheid bestaat om adresseringsmethoden in een instructie te combineren. Welke toegelaten zijn hangt van de complexiteit van de processor af, maar één ervan moet genoemd worden vanwege het relatief veel voorkomen. Dat is de combinatie van *geïndexeerde* en *indirecte* adressering. De combinatie van deze twee kan op verschillende manieren plaatsvinden, afhankelijk van wat er eerst wordt gedaan (zie ook paragraaf 3.3.8 bij auto-increment en -decrement).

1. Geïndexeerde indirecte adressering (pre-index)

Hierbij vindt eerst de indexering plaats en daarna de indirecte adressering. De inhoud van het indexregister wordt opgeteld bij de adresbits uit de instructie en deze waarde levert het adres waar de wijzer staat (weer een adres) die de operand adresseert, zie figuur 3.5.

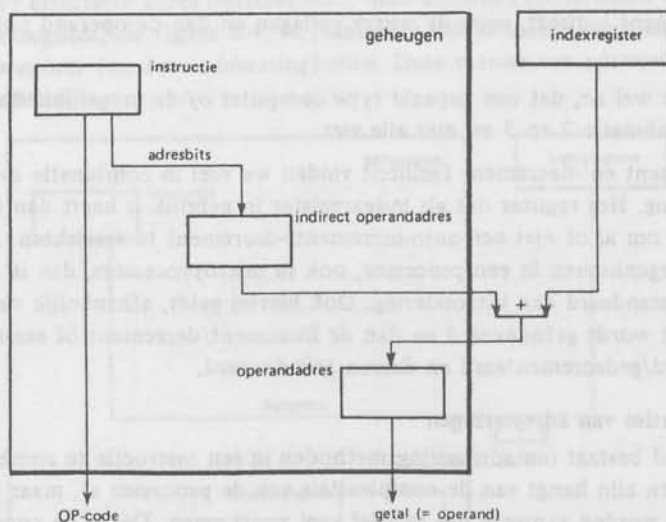


Figuur 3.5. Geïndexeerde indirecte adressering (pre-index).

2. Indirect geïndexeerde adressering (post-index)

Nu vindt eerst een indirectie plaats en dan pas een indexering. Dus de bits bij de instructie verwijzen naar een wijzer; de inhoud van de wijzer wordt opgeteld bij de inhoud van het indexregister, de som levert het effectieve adres van de operand.

Zie figuur 3.6.



Figuur 3.6. Indirect geïndexeerde adressering (post-index).

Andere combinaties van adresseringsmethoden komen soms voor, waarbij het eerder een exotisch trekje van een bepaalde processor is dan een duidelijk nuttige bijdrage aan de adresseringsfaciliteiten.

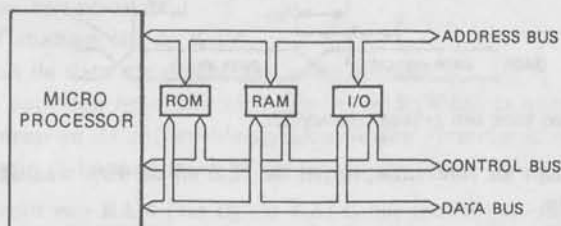
Adresmodificaties, zoals die van of door een *operating system* worden gedaan, denk aan paginerings en segmentering, vallen buiten de hier bedoelde adresveranderingen. Waar in deze leerstof over *virtuele technieken* wordt gesproken, zal nader worden ingegaan op deze modificatie van gegenereerde adressen (zie onder andere hoofdstuk 9 en 10).

4. Tijdaspecten in een microcomputer

4.1. Inleiding

In een microcomputer vindt een intensieve uitwisseling van informatie plaats tussen de diverse blokken van de microcomputer, zie figuur 4.1. Onder normale omstandigheden loopt alle informatie via de microprocessor.

De blokken hebben ieder een eigen specifieke functie. In het blok leesgeheugen (ROM) bewaren we programmatuur en constanten. Het lees-/schrijfgeheugen (RAM) kan gebruikt worden als kladgeheugen en voor het opslaan van variabelen. Bovendien kan dit geheugen dienen voor de opslag van programmatuur, die uit een achtergrondgeheugen geladen wordt.



Figuur 4.1. Blokschema van een microcomputer.

Alle blokken zijn aangesloten op een lokaal bussysteem. De microcomputer kan via de adresbus kenbaar maken met welk blok en met welk deel van dat blok hij wil communiceren. De databus zorgt onder besturing van de controlbus voor het informatietransport tussen de blokken en de microprocessor. Wil dit bussysteem betrouwbaar werken, dan moet men er voor zorgen, dat informatie en signalen in de juiste *volgorde* en op de juiste *tijdstippen* op de bus verschijnt.

4.2. Tijdbepalende elementen bij geheugens

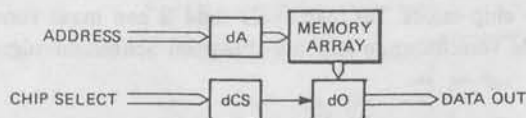
4.2.1. ROM IC's

Een ROM IC is, wat de aansturing betreft, eenvoudig in het gebruik. De volgorde, die men bij het lezen van de informatie aan moet houden is als volgt:

- de adresinformatie dient aan de adresingangen van de ROM aangeboden te worden,
- vervolgens moet de ROM geselecteerd worden door het aansturen van de chip-select en/of de chip-enable ingang(en),
- hetgeen na een zekere toegangstijd resulteert in het aanbieden van de informatie uit de geselecteerde geheugenlocatie aan de data-uitgangen van de ROM.

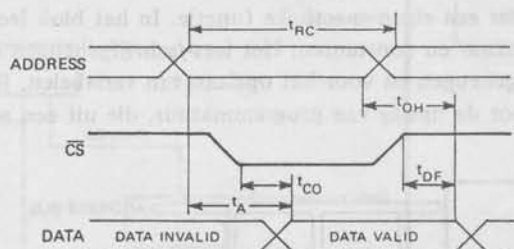
Om de juiste tijdstippen te kunnen bepalen waarop eerdergenoemde handelingen moeten plaatsvinden, zullen we een tijdmodel invoeren.

In het tijdmodel van de ROM (zie figuur 4.2) is het geheugen ideaal verondersteld en zijn tijdvertragende elementen tussen het geheugen en de in- en uitgangen geplaatst.



Figuur 4.2. Tijdmodel voor een ROM.

In het model vinden we de ingangsvertraging d_A voor de adresingangen en d_{CS} voor de selectie-ingangen. In d_A zijn alle vertragingstijden van de ingangsbuffers, de kolom-decoders, de rij-decoders en de geheugencellen ondergebracht. De vertraging d_{CS} wordt veroorzaakt door de looptijden van de selectielogica en d_O is de tijdvertraging in de outputbuffers. Aan de hand van het tijdmodel kunnen we een tijddiagram opstellen voor de leescyclus van een ROM, zie figuur 4.3.



Figuur 4.3. Tijddiagram voor een geheugenleescyclus.

In het tijddiagram zijn als referentiepunten de 10% en de 90% waarden van de signaal-overgangen genomen.

Nemen we aan dat de selectie van de geheugenchip stabiel is, dan zal de tijd, die verloopt tussen adressering en het betrouwbaar aanwezig zijn van de data aan de uitgangen gelijk zijn aan t_A . Dit is de som van d_A en d_O in het tijdmodel. Zo is, op het moment dat de adressering van de ROM stabiel is, de vertraging tussen het selecteren van de ROM en het verschijnen van betrouwbare data aan de uitgangen gelijk aan

$$t_{CO} = d_{CS} + d_O.$$

In dit tijddiagram is het CS-sigitaal vertraagd ten opzichte van de adressering, dit is geen noodzaak. Echter in de selectie van een geheugenchip zit meestal ook een deel van de adressering verwerkt. De logica, die de chip select verzorgt, is mede oorzaak van genoemde vertraging.

Deze vertraging hoeft geen probleem te zijn daar t_{CO} meestal kleiner is dan t_A . Dit komt omdat de chip-select de outputbuffers rechtstreeks stuurt.

Vraagt de processor toegang tot een geheugen, dan biedt de processor het adres samen met een aantal besturingssignalen aan het geheugen aan en moet dan enige tijd (t_A) wachten voordat de uitgaande informatie van het geheugen overgenomen mag worden. Bij langzame geheugens kan deze toegangstijd langer zijn dan de tijd die de processor normaal nodig heeft. De processor moet dan zolang in zijn werkzaamheden worden opgehouden tot de benodigde toegangstijd verstreken is. De meeste geheugensystemen zijn daartoe uitgerust met een *Memory Ready* signaal. Dit signaal meldt aan de processor, wanneer de, voor het betreffende geheugendeel vereiste, wachttijd voorbij is. In het tijddiagram zien we nog de tijden t_{OH} , t_{DF} en t_{RC} . De *output hold time* t_{OH} en de *chip deselect to output float time* t_{DF} geven aan hoe lang de gevraagde informatie nog betrouwbaar aan de uitgangen aanwezig is na het veranderen van respectievelijk het adres en de chip-select. De read cycle time is een maat voor de herhalings-frequentie waarmee de verschillende geheugenplaatsen achtereenvolgens uitgelezen kunnen worden.

4.2.2. RAM IC's

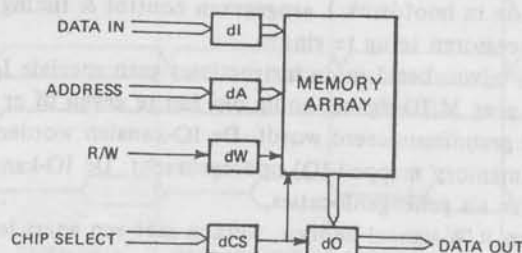
Bij de RAM's, RWM's of lees-/schrijfgeheugens onderscheiden we statische en dynami-

sche geheugens, deze laatste categorie is wat de aansturing betreft minder eenvoudig in het gebruik. Immers een dynamisch geheugenelement verliest na verloop van tijd zijn opgeslagen informatie wanneer deze niet regelmatig herschreven wordt. Daarom is er op vaste tijdstippen een refresh cyclus nodig. De besturingssignalen voor deze refresh cyclus worden of op de geheugenchip zelf, of door de microprocessor, of door extra toegevoegde logica gegenereerd.

Het lezen van statische geheugens verloopt op een vergelijkbare wijze als bij de ROM's. Indien de processor in het geheugen wil schrijven dan wordt dit kenbaar gemaakt door middel van het R/W-signaal, dat meestal hoog is bij het lezen en laag bij het schrijven. Ook bij het schrijven van geheugens zal een juiste volgorde en timing van de signalen nodig zijn, de volgorde is:

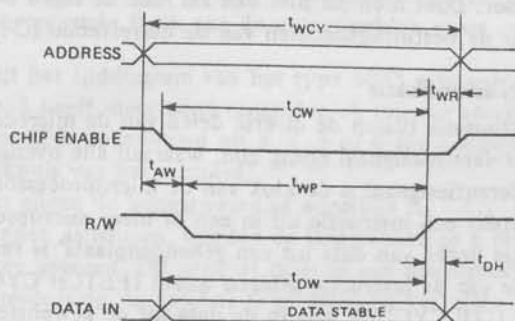
- aanbieden van het adres,
- selecteren of enablen van de RAM,
- aanbieden van de data die geschreven moet worden,
- data inlezen door een negatieve klokpuls op de R/W-lijn te geven,
- waarna de adres- en de chipenable-signalen mogen veranderen teneinde een andere geheugenlocatie te kunnen lezen of schrijven.

Het tijdmodel voor een RAM (zie figuur 4.4) is het model van de ROM, uitgebreid met de vertragingen dI voor de inkomende data en dW voor het schrijfsignaal.



Figuur 4.4. Tijdmodel voor een statische RAM.

Beschouwen we het tijddiagram voor het schrijven van een lees-/schrijfgeheugen, dan zien we het READ/WRITE-signaal in de vorm van een schrijfpuls met de tijdsduur t_{WP} . Deze t_{WP} moet aan een bepaald minimum voldoen om er zeker van te zijn dat de informatie goed is overgenomen. Voor de meeste geheugens dient de toestand van de datalijnen pas stabiel te zijn bij de nul/één-overgang van de schrijfpuls, daar de geheugens de informatie overnemen op een flank van de puls overeenkomstig een edge-triggered flip-flop. Het R/W-signaal heeft een tijd dW nodig om in de geheugenmatrix



Figuur 4.5. Schrijfcyclus voor een lees-/schrijfgeheugen.

effectief te worden, terwijl de vertraging dI de tijd is, die de data nodig heeft om van de ingangspennen naar de geselecteerde geheugencellen te komen. De data-ingangen moeten daarom stabiel zijn in een tijd $dI - dW$ voor de opgaande flank van R/W .

In het tijddiagram zijn dit de tijden *data setup* (t_{DW}) en *data hold* (t_{DH}). Na het stabiel zijn van de adresingangen zal het intern nog even duren voor de juiste geheugenlocatie geselecteerd is. Om nu te voorkomen, dat verkeerde geheugenlocaties door de schrijfpuls beïnvloed worden, mag de schrijfpuls pas een tijd t_{AW} na het stabiel zijn van de adresingangen verschijnen. Het adres mag om dezelfde reden pas veranderen een tijd t_{WR} na de opgaande flank van de schrijfpuls.

De chip-enable moet een zekere tijd t_{CW} voor de nul/één-overgang van de schrijfpuls aanwezig zijn.

De hele schrijfcyclus duurt t_{CWY} en bepaalt daarmee de snelheid waarmee achtereenvolgens geheugenplaatsen geschreven kunnen worden.

4.3. Communicatie tussen CPU en geheugen

4.3.1. Inleiding

De CPU is uitgerust met een aantal besturingssignalen om met de diverse delen van de microcomputer te kunnen communiceren. Deze signalen zijn nodig voor de synchronisatie met en de selectie van de betreffende delen. Ieder type processor communiceert op zijn eigen wijze, de in hoofdstuk 1 aangegeven control & timing signalen zijn dan ook niet bij alle processoren terug te vinden.

Processoren, waarbij bijvoorbeeld in de instructieset geen speciale I/O-instructies zijn opgenomen hebben geen M/IO-sigitaal nodig om aan te geven of er met een geheugenlocatie of IO-kanaal gecommuniceerd wordt. De IO-kanalen worden bij dit type in het geheugenadresveld (memory mapped-IO) ondergebracht. De IO-kanalen functioneren dan op dezelfde wijze als geheugenlocaties.

Processoren, die geen R/W-sigitaal hebben, werken met een apart lees- (RD) en een apart schrijfsigitaal (WR).

Wat bij de ene processor het SYNC-sigitaal is, is bij een ander type een samenstel van signalen.

Tenslotte, niet iedere processor heeft een READY-ingang die gebruikt kan worden om de processor in zijn werkzaamheden op te houden bij langzaam werkende geheugens of I/O-kanalen.

Gezien bovengenoemde verschillen is het erg moeilijk om standaard geheugen- en/of interface-modulen te ontwerpen, die met alle typen processoren kunnen samenwerken. Iedere fabrikant hoopt, dat men voor de modulen gebruik maakt van de IC's uit de familie van de processor. Doet men dit niet dan zal men de eigen besturingssignalen moeten omzetten naar de besturingssignalen van de betreffende IC-familie.

4.3.2. Validering en synchronisatie

Om een goede synchronisatie tussen de diverse delen van de microcomputer te kunnen realiseren, zal er een referentiesigitaal nodig zijn, waaraan alle overige signalen gerelateerd worden. Dit referentiesigitaal is de klok van de microprocessor.

Een microprocessor voert een instructie uit in een of meer microprocessor cycli, bijvoorbeeld de instructie 'lezen van data uit een geheugenplaats' is verdeeld in een lees-cyclus waarin de code van de instructie gelezen wordt (FETCH CYCLE), gevolgd door een leescyclus (EXECUTE CYCLE), waarin de data uit de gewenste locatie opgehaald wordt.

Hoewel er processoren zijn, die iedere cyclus uitvoeren in één klokpulsperiode, gebruiken de meeste typen processoren meer dan een klokpuls per cyclus.

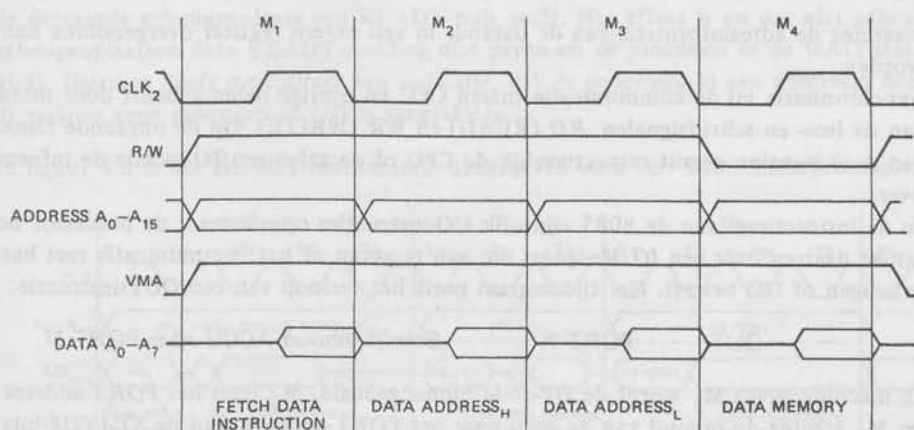
Het type 6800 microprocessor werkt met één klokpuls per cyclus, zie figuur 4.6. Dit type heeft geen M/I/O-sigitaal, de I/O-kanalen liggen in het geheugenadresveld. Wel heeft de processor een *VMA*-sigitaal (Valid Memory Address), waarmee het aangeeft dat alle bussen voorbereid zijn voor communicatie met een geheugenlocatie.

Het *R/W*-sigitaal bepaalt of de locatie gelezen of geschreven moet worden. Beide signalen, *VMA* en *R/W*, zijn statisch en kunnen niet dienen voor de synchronisatie. Het *SYNC*-sigitaal is hier te realiseren met een AND-functie van *VMA* en *CLK*.

In het tijddiagram van figuur 4.6 is een schrijfpdracht naar een geheugen- of I/O-locatie weergegeven,

STA A ADDRESS_{HL} Zet de inhoud van ACCU A in
geheugenlocatie ADDRESS_{HL}.

Voor deze instructie zijn vier machine-cycli nodig.



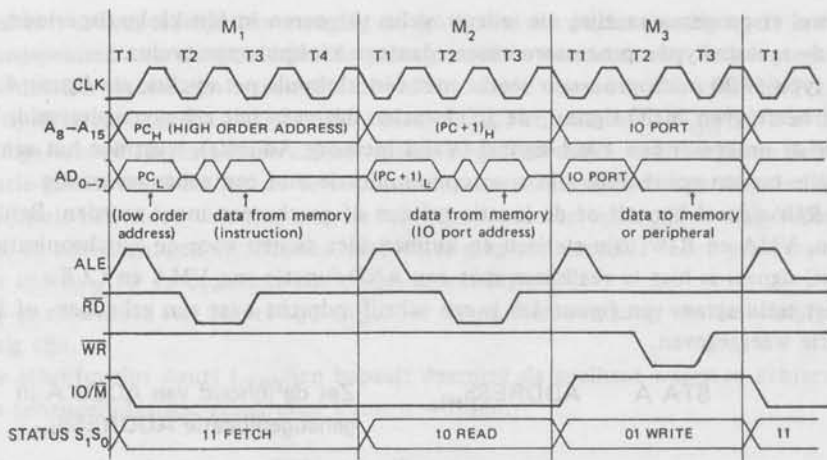
Figuur 4.6. Tijddiagram voor een schrijfpdracht in een geheugen- of I/O-locatie bij een type 6800 microprocessor.

In de cyclus M₁ wordt de *OPCODE* van de instructie gelezen uit de geheugenplaats aangegeven door de PC (Program Counter), de processor haalt de data op de neergaande flank van CLK₂ van de databus en plaatst de data in het IR (Instruction Register). In M₂ en M₃ wordt respectievelijk de DATA ADDRESS_H en de DATA ADDRESS_L uitgelezen en in het Memory Address Register geplaatst. In cyclus M₄ wordt tenslotte de inhoud van de ACCU op de databus gezet, waarna het geheugen de DATA op de neergaande flank van de processorklok moet overnemen.

Veel complexer zit het tijddiagram van het type 8085 microprocessor in elkaar, zie figuur 4.7. De 8085 heeft meer controlsignalen en minder adreslijnen.

Een machinecyclus M is opgebouwd uit 3, 4, 5 of 6 toestandsperioden T, die samen vallen met de klokpuls van het systeem.

De adresbus levert alleen de hoogstwaardige adresbits A₈-A₁₅. Om toch een adresveld van 16 bits te kunnen aansturen, worden de laagstwaardige 8 bits van het adres via de databus naar buiten gevoerd. Dit gebeurt door in een microprocessorcyclus eerst het laagstwaardige adresdeel op de databus te plaatsen en vervolgens de data over de bus te sturen. Door deze constructie is er een *ALE*-sigitaal (Address Latch Enable) nodig,



Figuur 4.7. Tijddiagram voor een I/O-opdracht bij een type 8085 microprocessor.

waarmee de adresinformatie van de databus in een extern register overgenomen kan worden.

Synchronisatie bij de communicatie tussen CPU en overige delen gebeurt door middel van de lees- en schrijfsignalen, *RD* (READ) en *WR* (WRITE). Op de opgaande flank van deze signalen neemt respectievelijk de CPU of de geheugen/IO-locatie de informatie over.

In de instructieset van de 8085 zijn ook I/O-instructies opgenomen, de processor beschikt daarom over een *IO/M*-signaal om aan te geven of het communicatie met het geheugen of I/O betreft. Het tijddiagram geeft het verloop van een *OUT*-instructie:

OUT PORT N Schrijf inhoud ACCU naar PORT N.

In machinecyclus M_1 wordt de OP-code binnengehaald, M_2 leest het *PORT*-address en M_3 schrijft de inhoud van de accu naar het *PORT*-address. Aan de *STATUS*-bits S_1S_0 kan men zien met welke operatie de processor bezig is:

S_1	S_0	$IO/\bar{M}$	Status
0	1	0	Memory write
1	0	0	Memory read
0	1	1	I/O write
1	0	1	I/O read
1	1	0	Opcode fetch
1	1	1	Interrupt acknowledge
0	0	Tri	HALT, IO/M is in tri-state toestand

S_1 is te gebruiken als voorbericht voor een lees- of schrijfactie.

4.4. Communicatie tussen CPU en 'langzame componenten'

4.4.1. Communicatie met behulp van het READY-mechanisme

Tussen de componenten, waar de processor informatie mee uitwisselt, kunnen zogenaamde langzame componenten zitten, ROM's, I/O-kanalen of dynamische RAM's, die op gezette tijden bezig zijn met een refresh-cyclus.

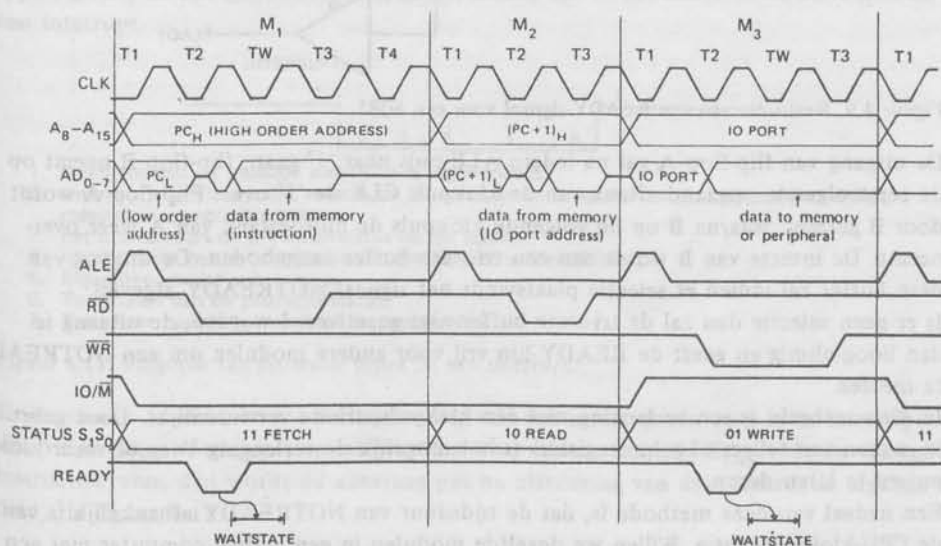
Om betrouwbare informatie-uitwisseling te garanderen moet er een mechanisme wor-

den gerealiseerd, waarmee de machinecyclus kan worden verlengd tot de vereiste toegangstijd verstreken is. Veel processoren zijn hiertoe uitgerust met een **READY**-ingang, die zolang deze ingang hoog is, de processor zijn normale machinecyclus laat afmaken. Is de **READY**-ingang laag dan zal de processor in een 'wacht toestand' komen en dit melden met een **WAIT**-signaal. In deze toestand doet de processor niets.

Bij processoren, die dit **READY**-mechanisme niet hebben, zal de klokpuls beïnvloed moeten worden.

Het **READY**-signaal wordt door de logica rond de langzame componenten gegenereerd. Er zijn twee basismethoden aan te geven voor de synchronisatie. Het **READY**-signaal kan direct op de processor aangesloten worden en deze zal in de **WAIT**-toestand komen, indien bij het aanroepen van een adres-/IO-locatie **NOT READY**-signaal opgewekt wordt. Dit betekent, dat ook de aanroep van niet gebruikte adresplaatsen door de processor snel verwerkt wordt (er is geen logica, die een **NOT READY**-melding verzorgt), het lijkt zinloos maar men kan het gebruiken om te testen op welke plaatsen geheugen of I/O zit. Bij de tweede methode genereert logica bij de CPU altijd een signaal **NOTREADY** tot de gevraagde geheugenplaats een **READY**-puls geeft. Het effect is nu dat niet gebruikte geheugenplaatsen deze **READY**-melding niet geven en de processor in de **WAIT**-status blijft. Hiermee heeft men direct een indicatie, dat de processor in een adresveld bezig is, waarop geen geheugen of I/O is aangesloten.

In figuur 4.8 is het **READY**-mechanisme aangegeven voor een 8085 microprocessor.



Figuur 4.8. Tijddiagram voor een I/O-instructie met een **WAIT**-status bij de **OPCODE** FETCH en de I/O-write cyclus.

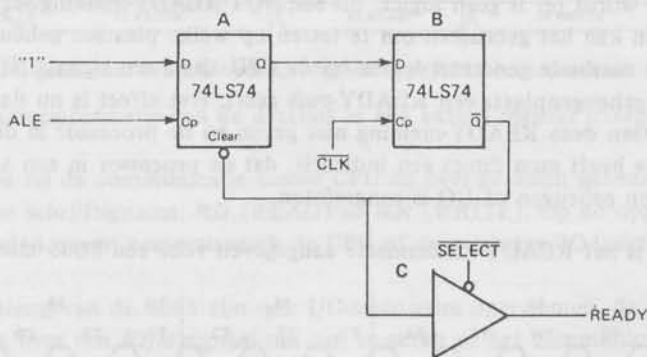
De **OPCODE** wordt uit een langzaam geheugen gelezen. Na de selectie van de locatie zal het geheugen **NOTREADY** melden. De processor test op het **READY**-signaal in machinestatus T_2 , hij vindt de toestand **NOTREADY** en brengt zich in de **WAIT**-toestand.

De processor blijft in deze toestand tot het **READY**-signaal verschijnt, waarna de processor in machinestatus T_3 komt en de opcode fetch verder afmaakt. In het tijddiagram is ook het schrijven naar een langzaam I/O-kanaal aangegeven. Het verlengen van de machinecyclus gebeurt op dezelfde wijze als bij de opcode-fetch. Daar er geen limiet

bestaat aan de tijd dat de READY-lijn laag gehouden wordt, kunnen we deze lijn ook gebruiken om met behulp van een druktoetsschakelaar een SINGLE STEP-faciliteit te realiseren. Met deze single step kunnen we de statische werking van de microcomputer stap voor stap verifiëren.

Bij processoren, die geen READY-ingang hebben, is het alleen mogelijk de machinecyclus te verlengen door het ophouden van de klokpuls voor de processor. Hebben we te maken met een dynamische processor, zoals de 6800, dan is daar een limiet aan verbonden, het ophouden van de klok mag niet langer dan $10\ \mu\text{s}$ duren.

Er zijn een aantal methoden om het READY-sigitaal op te wekken. Zo is er een methode, die gebruik maakt van de klokpuls om in discrete stappen, gelijk aan de klokpulsperiode, de tijd te verlengen. In figuur 4.9 is de realisatie gegeven voor het verlengen van de machinecyclus met één klokpulsperiode, zie ook figuur 4.8.



Figuur 4.9. Realisatie van een READY-sigitaal voor een 8085.

De uitgang van flip-flop A zal na iedere ALE-puls naar '1' gaan, flip-flop B neemt op de eerstvolgende opgaande flank van de klokpuls CLK de '1' over. Flip-flop A wordt door B gereset, waarna B op de volgende klokpuls de nultoestand van A weer overneemt. De inverse van B wordt aan een tri-state buffer aangeboden. De uitgang van deze buffer zal indien er selectie plaatsvindt het sigitaal NOTREADY afgeven. Is er geen selectie dan zal de tri-state buffer niet geactiveerd worden, de uitgang is dan hoog-ohmig en geeft de READY-lijn vrij voor andere modules om een NOTREADY te melden.

In dit voorbeeld is een verlenging met één klokpulsperiode verwezenlijkt. Door gebruik te maken van tellers of schuifregisters is het mogelijk de verlenging twee of meer klokpulsen te laten duren.

Een nadeel van deze methode is, dat de tijdsduur van NOTREADY afhankelijk is van de CPU-klopfrequentie. Willen we dezelfde modules in een andere computer met een afwijkende klopfrequentie gebruiken dan kan het zijn, dat de tijd te kort is, waardoor het systeem niet functioneert, of de tijd veel te lang is en de processor onnodig lang opgehouden wordt.

Een ander bezwaar is, dat de tijdsduur van NOTREADY slechts in discrete stappen (gelijk aan de klokpulsperiode) ingesteld kan worden, een optimale instelling is daarbij niet mogelijk.

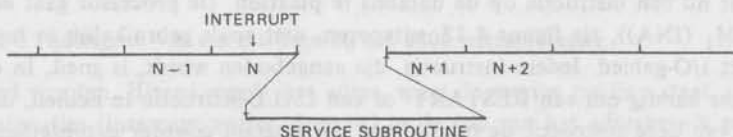
Aan deze bezwaren kan tegemoetgekomen worden door de flip-flops A en B te vervangen door een monostabiele trekker, die gestart wordt op de ALE-puls. De tijdsduur is in te stellen met een RC-netwerk en dus onafhankelijk van de klopfrequentie van het systeem.

4.4.2. Synchronisatie met behulp van het INTERRUPT-mechanisme

Iedere microprocessor is voorzien van één of meer INTERRUPT-ingangen. Randapparaten kunnen via deze ingangen de aandacht van de processor vragen. De processor test, mits hij niet 'doof' is gemaakt voor een interrupt, aan het eind van elke instructiecyclus of er een INTERRUPT REQUEST is geplaatst.

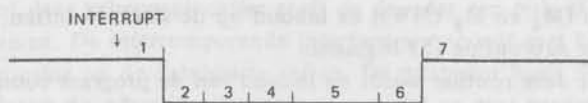
De mogelijkheid om de processor te interrumpen kan ingesteld worden met het setten respectievelijk resetten van het INTERRUPT ENABLE of INTERRUPT INHIBIT bit in het statuswoord van de processor.

Bij een interrupt maakt de processor altijd eerst de instructie af, voordat begonnen wordt met het afhandelen van de interrupt. In figuur 4.10 zien we hoe het hoofdprogramma door een interrupt onderbroken wordt. Eerst maakt de processor instructie N af en gaat vervolgens verder met de interrupt service subroutine. Na de service subroutine vervolgt de processor het geïnterrumpeerde programma bij N + 1.



Figuur 4.10. Onderbreking van het programma door een interrupt.

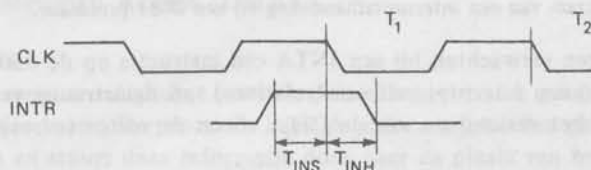
De afhandeling van de interrupt is voor ieder type processor anders, maar voor alle typen gelden de in figuur 4.11 aangegeven acties, die de processor moet uitvoeren bij een interrupt.



1. Wachtijd tot de lopende instructie is afgehandeld.
2. Interrupt machinecyclus, waarin de inhoud van de program counter opgeslagen wordt, meestal in combinatie met:
3. het opslaan van de processorstatus op de stack,
4. Identificatie van het randapparaat dat geïnterrumpeerd heeft,
5. Eigenlijke service subroutine.
6. Terughalen van de processorstatus.
7. Terugkeer naar het onderbroken programma.

Figuur 4.11. Volgorde van processor acties bij een interrupt.

De *minimum wachttijd* voor de honorering van een interruptaanvraag is de interrupt setup time T_{INS} . Verschijnt de interruptaanvraag binnen een tijd T_{INS} vóór de volgende instructiecyclus, dan wordt de aanvraag pas na uitvoering van deze instructie afgehandeld, zie figuur 4.12.

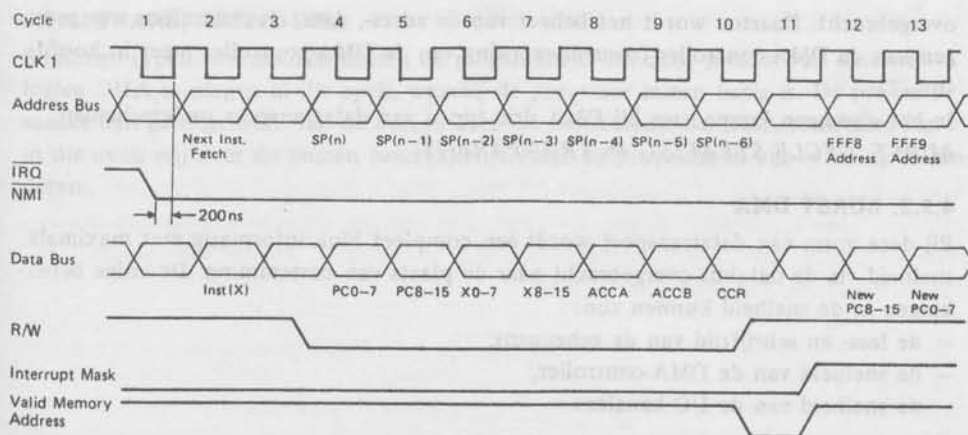


Figuur 4.12. Interrupt setup time.

Voor een 8085A processor is de minimum setup time gelijk aan

$$T_{INS} = \frac{1}{2}T + 200 \text{ ns},$$

waarin T de tijdsduur van één machinecyclus is.



Figuur 4.14. Tijddiagram van een interrupt bij een 6800 microprocessor.

uitgevoerd worden. Hierna wordt het adres, waar de service routine staat, uit twee geheugenlocaties (interrupt vectoradressen) in de top van het adresbereik gehaald. De 6800 kent geen INTA-sigitaal. Wil men weten, wie geïnterrupteerd heeft (4), dan moet men de status van alle interfaces opvragen (pollen), hetgeen meer tijd kost naarmate er meer interfaces zijn.

Toch kan men de identificatie versnellen door met extra logica zelf een INTA-sigitaal op te wekken. Men decodeert daartoe de interrupt vectoradressen. Zodra de processor toegang vraagt tot deze geheugenlocaties geeft de decoder een puls af, die als een INTA is te gebruiken. De interrumpende interface antwoordt met het beginadres van zijn service routine op de databus te zetten. De processor haalt dit adres in twee slagen binnen, plaatst dit adres in zijn program counter en gaat verder met de betreffende service routine.

Na de service routine (5) worden de inhoud van de op de stack geplaatste registers weer in de registers overgenomen (6), hetgeen 7 leescycli kost. Al met al gaat er veel tijd verloren, voordat er met de werkelijke interruptafhandeling begonnen kan worden. Het is echter niet altijd noodzakelijk genoemde registers allemaal op de stack weg te zetten. De fabrikant heeft daarom in een later type processor (6809) een FAST INTERRUPT REQUEST (FIRQ) mogelijkheid toegevoegd. Bij FIRQ's worden alleen de program counter en het conditiecodelregister op de stack gezet. Deze constructie is bij de 6809 zeker nodig, daar deze processor over een uitgebreide set registers beschikt, die bij een IRQ in zijn geheel naar de stack gaat, hetgeen 12 schrijfcycli vraagt.

4.5. Direct Memory Access (DMA)

4.5.1. Inleiding

In een microcomputer vindt het normale datatransport onder programmabesturing plaats via de processor. De processor haalt de informatie uit het geheugen of het I/O-kanaal binnen en stuurt deze informatie door naar de plaats van bestemming. Deze vorm van datatransport is tijdrovend, de hoeveelheid informatie, die per tijdseenheid overgebracht kan worden, is gering. Bovendien kan de processor gedurende de tijd, waarin het transport plaats vindt, niets anders doen. Men heeft voor dit probleem een oplossing gevonden in de vorm van *Direct Memory Access* (DMA). De data wordt hierbij direct uit de plaats van herkomst naar de plaats van bestemming

overgebracht. Daartoe wordt het beheer van de adres-, data- en controlbus overgedragen aan de DMA-controller (over de werking van de DMA-controller meer in hoofdstuk 7).

In het algemeen kunnen we bij DMA drie typen van datatransport onderscheiden, *BURST*, *CYCLE STEALING* en *TRANSPARENT*.

4.5.2. BURST DMA

Bij deze vorm van datatransport wordt een compleet blok informatie met maximale snelheid via de databus overgebracht naar de plaats van bestemming. De enige beperkingen in de snelheid kunnen zijn:

- de lees- en schrijftijd van de geheugens,
- de snelheid van de DMA-controller,
- de snelheid van de I/O-kanalen.

Het verloop van het datatransport is als volgt:

1. De processor voorziet de DMA-controller van:
 - het beginadres in het geheugen van waar de informatie komt of waar de informatie naar toe gaat, én
 - het aantal woorden dat overgebracht moet worden.
2. Wanneer de I/O-kanalen gereed zijn voor het transport zal de controller een DMA-request naar de processor sturen.
3. De processor zendt een DMA ACKNOWLEDGE terug, maakt de adres-, data- en de voor het transport benodigde controluitgangen hoog-ohmig, zodat de controller daar gebruik van kan maken.
4. De controller neemt de besturing over en verzorgt de juiste signalen voor het lezen/schrijven van geheugenplaatsen en I/O-kanalen, de controller brengt nu zonder onderbreking het hele blok informatie over.
5. Is het gewenste aantal woorden overgezonden dan sluit de controller de DMA-sessie af, haalt de DMA-aanvraag weg en zal eventueel een interrupt naar de processor genereren om aan te geven, dat het DMA-transport heeft plaatsgevonden.

BURST DMA wordt veelal toegepast bij het overbrengen van informatie van en naar floppy-disk achtergrondgeheugens of bij het verversen van beeldschermgeheugens.

Bij niet alle procestypen kan zonder meer *BURST DMA* toegepast worden, met name de dynamische typen kunnen niet voor langere tijd in hun werking worden opgehouden zonder hun inwendige status te verliezen. Wel kunnen ze in een *HALT* toestand gebracht worden, hierbij zal gelijk bij een interrupt de gehele processorstatus naar de stack overgebracht moeten worden. In plaats van *BURST DMA* is het ook mogelijk de volgende vorm van DMA toe te passen.

4.5.3. CYCLE STEALING DMA

Bij *CYCLE STEALING* wordt de processor niet stilgelegd maar vindt het datatransport plaats tussen de door de processor uit te voeren instructies. Het verloop van het datatransport is gelijk aan het hierboven gegeven overzicht bij de *BURST DMA* uitgezonderd punt 4. Hier zal de controller na ieder overgebracht woord de DMA-aanvraag weg halen en het proces bij punt 2 vervolgen.

De naam zegt het al, bij deze methode steelt de DMA-controller van de processor cycli (toestandstijden), waarin het datatransport kan plaatsvinden. De werking van de processor wordt daardoor vertraagd, bovendien verloopt het DMA-proces minder snel dan bij *BURST DMA*.

4.5.4. TRANSPARENT DMA

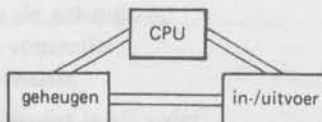
Sommige typen processoren bieden de mogelijkheid om door middel van externe logica DMA te plegen in die cycli, waarbij de processor intern bezig is. De processor maakt dan geen gebruik van de adres-, data- en controlbus. De DMA-controller kan in die cycli vrij over de bussen beschikken zonder de processor in zijn werking te hinderen.



5. Communicatie in de microcomputer

5.1. Dataflow

In eerdere hoofdstukken hebben we al gezien dat in een (micro)computer verschillende delen kunnen worden onderscheiden naar de functie die ze hebben. Zo zagen we als belangrijkste delen de *CPU*, het *geheugen* en de *I/O*. Wanneer we spreken over *dataflow*, letterlijk 'gegevensstroom', in een computer, dan gaat het over de uitwisseling van informatie tussen deze computerdelen onderling. In figuur 5.1 is deze samenhang, wat dataflow betreft, nog eens weergegeven.



Figuur 5.1. Schema van de dataflow in een (micro)computer.

De lijnen stellen de verbindingen tussen de delen voor, waarlangs het datatransport plaatsvindt. Alle noodzakelijke signalen, zoals adres, data en besturing, vinden we bij *ieder* van deze verbindingen. Realiseren we een computer met een dergelijke opzet dan heeft dat als groot voordeel dat parallel transporten kunnen plaatsvinden tussen bijvoorbeeld CPU en geheugen en tussen CPU en I/O; dit resulteert in een hoge verwerkingsnelheid. We hebben dan een opzet waarbij de feitelijke doorverbindingen overeenstemmen met de gewenste gegevensstromen. Dit is een benadering die wel wordt aangewend in grotere computers. Toch weegt het voordeel meestal niet op tegen de nadelen:

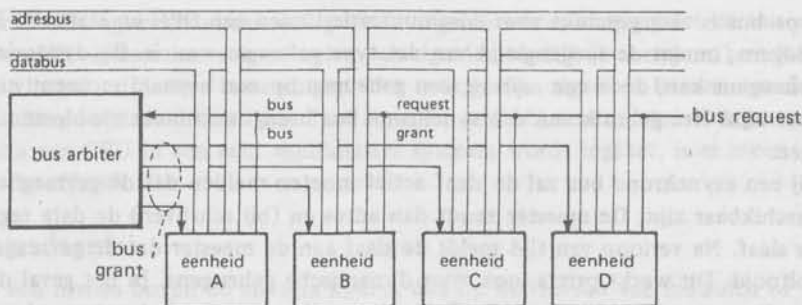
- Verschillende, onderling onafhankelijke gegevensstromen vloeien in en uit het geheugen. Om toegangsflicten te vermijden is een vrij complexe logica nodig, evenzo voor de synchronisatielogica per kanaal als we er-van uit gaan dat de snelheidsvoordelen van de (onafhankelijke) verbindingen moeten worden verzekerd.
- Aparte verbindingen tussen de verschillende delen geeft een grotere hoeveelheid bedrading, gevolgd door hogere ontwerp- en fabricagekosten. Ook zal het ontwerp weinig flexibel zijn met weinig mogelijkheden tot latere modificaties.

Bovenstaande afwegingen leiden ertoe dat we in de praktijk meestal een opzet tegenkomen zoals al is gegeven in figuur 1.9. De verbindingen worden dan verzorgd door een *bussysteem*. Alle gegevens worden uitgewisseld via een enkel verbindingssysteem, maar dan *na elkaar* in de tijd gezien. Dit heeft als voordelen dat de bedrading eenvoudig kan worden gehouden, het aansluiten van modules op de bus is simpel omdat de busdefinitie vastligt en zelfs kunnen verschillende fabrikaten op dezelfde bus worden aangesloten.

5.2. Busstructuur

5.2.1. Algemene opzet

In hoofdstuk 1 is al getoond dat we een bus van een (micro-)computersysteem opgebouwd kunnen denken uit een adres-, een data- en een controlbus. Voor het geval dat we te maken hebben met een systeem dat uit een geheugen, een I/O-deel en een CPU bestaat is dit een vrij duidelijke opzet omdat in het algemeen alle acties vanuit de CPU geïnitieerd worden. Wat echter, als op die bus nog één (of meer) eenheden zijn aangesloten die ook een actie willen starten? Dan is er een scheidsrechter nodig



Figuur 5.2. Schema toewijzing van de bus door bus arbiter.

die beslist wie de bus mag gaan besturen en hoelang. Deze scheidsrechter wordt een *bus arbiter* genoemd en de controlbus kent een aantal signalen welke speciaal bestemd zijn om de toewijzing van de bus te regelen. In figuur 5.2 is dit schematisch weer-gegeven.

In de figuur zijn de signalen *bus request* en *bus grant* in gebruik om de toestemming tot gebruik van de bus aan te vragen, respectievelijk te verkrijgen. Wie er wanneer deze toestemming krijgt, wordt geregeld door de bus arbiter. Hierin zit eventueel een toekenningsprincipe, waarbij ook nog met een prioriteit rekening kan worden gehouden. De eenheid die de controle over de bus krijgt toegewezen, noemen we de *meester* en de eenheid die door de meester wordt aangestuurd heet dan *slaaf*. In voorkomende gevallen kan er transport plaatsvinden van een meester naar een aantal slaven of kan een meester een datatransport laten starten tussen twee slaven. In al die gevallen blijft er toch steeds maar één eenheid, welke de volledige controle van de bus heeft. Vaak vinden we in praktische omstandigheden geen echte bus arbiter. Dit kan dan twee oorzaken hebben. Of er is maar één element op de bus aangesloten dat als meester kan optreden, zoals bij alle simpele processorsystemen, of de functie van bus arbiter bestaat wel, maar is verdeeld over de logica van de verschillende potentiële meesters. Dit laatste vinden we bijvoorbeeld in gedistribueerde processorsystemen, zie ook hoofdstuk 12.

De drie delen waarin een bus opgedeeld kan worden, zijn: de data-, de adres- en de besturingsbus. De verschillende eenheden die op een bus zijn aangesloten, gedragen zich als geheugenplaatsen. Al deze geheugenplaatsen hebben een uniek individueel identificatienummer, het *adres*. Als een adres op de bus wordt gezet, is dit te zien als een selectie van de overeenkomstige geheugenplaats; meer is er ook niet nodig om toegang (voor lezen of schrijven) tot die plaats te krijgen. De data wordt getransporteerd over de databus. Waar vandaan, waar naartoe en wanneer wordt geregeld door de signalen in de controlbus. Dit zijn voor een datatransport bijvoorbeeld signalen als: *clock*, *read* of *write* en de *bus request* en *bus grant*.

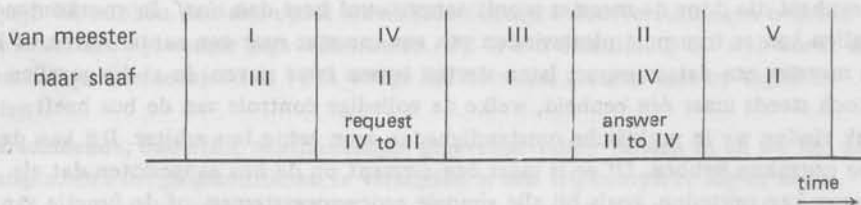
5.2.2. Bustiming

Naar gelang van de aard van de bustiming spreken we over *synchrone* of *asynchrone* bussen. Bij synchroon transport gebeuren alle handelingen onder besturing door een centrale klokfrequentie, bij asynchroon transport gebeurt dit via een vraag en antwoord mechanisme. Een synchrone bus stelt voor ieder datatransport een vaste tijd ter beschikking. Zo zal bij een leesopdracht de meester het adres van de slaaf op de bus zetten en na een vaste tijd er van uit gaan dat nu de slaaf de data op de databus heeft gezet. De informatie van de databus wordt dan ingelezen door de meester. Dit

type bus is zeer geschikt voor communicatie tussen een CPU en statische RAM geheugens, omdat de toegangstijd van dat type geheugen vast is. Bij dynamische RAM geheugens kan, door een *refresh*, een geheugen op een bepaald moment niet beschikbaar zijn. Het gebruik van een synchrone bus brengt dan meer problemen met zich mee.

Bij een asynchrone bus zal de slaaf actief moeten melden dat de gevraagde gegevens beschikbaar zijn. De meester zendt dan adres en (bij schrijven) de data tegelijk naar de slaaf. Na verloop van tijd meldt de slaaf aan de meester dat de gevraagde actie is voltooid. Dit werkt prima, ook voor dynamische geheugens. In het geval dat er actie wordt gevraagd tijdens een *refresh*, betekent dit enkel dat het antwoord van de slaaf wat langer op zich laat wachten.

Bij systemen met ondeelbaar datatransport gaat dit eventueel ten koste van de totale verwerkingscapaciteit van het systeem. Een systeem dat met deelbare datatransporten werkt, heeft hier minder last van. In dit laatste geval stuurt een meester een bevel naar een slaaf en geeft de bus vrij. Als de slaaf klaar is met de gevraagde actie, vraagt deze zelf de bus aan en zendt naar de vroegere meester (nu slaaf) het resultaat van de gevraagde actie. Tussen deze beide acties in kan er door de bus nog van alles getransporteerd zijn. Vooral in grote systemen zal men op een dergelijke manier een quasi simultaan gebruik van de bus voorstaan. De individuele bustransporten duren dan veel korter dan dat de geheugentoeegangstijden lang zijn. In figuur 5.3 is dit nog eens verduidelijkt.



Figuur 5.3. Bustransporten bij deelbare datatransporten.

Hoe groter het verschil tussen de bustransfertijd en de toegangstijd, des te meer transporten zullen er plaatsvinden tussen het moment van aanvragen en het moment waarop het antwoord komt. In de figuur is er maar één enkel ander transport aangegeven tussen deze beide acties, meestal zal dit aantal groter zijn. Overigens houdt de opzet van een dergelijk systeem wel in dat iedere eenheid in staat moet zijn de aangeboden adressen en data lokaal op te slaan ter verwerking. De informatie op de buslijnen zal immers vrijwel direct daarna veranderen.

De bussen, zoals we die tegenkomen bij de microprocessorcircuits, zijn voor een deel al behandeld in hoofdstuk 4 en komen verder nog ter sprake in hoofdstuk 10, op dit moment gaan we daar niet verder op in.

5.2.3. Multimaster bussen

We kennen twee gevallen waarin systemen voorkomen met meer dan één meester. Dat is ten eerste een systeem waarin een CPU voorkomt en daarnaast randapparatuur met een mogelijkheid van DMA-transport. Een tweede geval doet zich voor bij systemen met meerdere microprocessors op dezelfde bus.

Bij *DMA-transporten* wordt de besturing van de bus tijdelijk overgenomen door de DMA-controller, deze bestuurt dan de bus en bepaalt de waarde van adres-, data- en besturingslijnen. Een DMA-controller is dus zonder meer te zien als een processor

en er is een beslissingsmechanisme nodig om te regelen wie er toegang tot de bus mag hebben. Een CPU is daartoe vaak voorzien van signalen als BR (bus request) en BA (bus acknowledge), of andere benamingen voor soortgelijke signalen. Bij een enkele DMA aansluiting volstaat dit om met deze signalen de DMA-transporten te laten verlopen. Zodra een CPU in een echt multimaster systeem wordt ingezet, is er meer nodig. (Zie hoofdstuk 12).

5.3. Busstandaards

Als we op één niveau boven de circuits kijken, dus op het niveau van modules of printkaarten, dan zou het ideaal zijn als er één enkele definitie bestond van alle voorkomende bussignalen. Willekeurige merken konden dan door elkaar gebruikt worden, zonder problemen. Helaas hebben de fabrikanten, om de klanten aan hun fabrikaat te binden, ieder een eigen definitie van een bus uitgebracht. In de loop van de jaren zijn er hieruit enkele naar voren gekomen als meer succesvol dan anderen. In dit hoofdstuk worden er twee behandeld, beiden inmiddels tot IEEE-standaard verheven. Ten eerste de IEEE 696, afgeleid van de uit de hobbywereld afkomstige S-100 bus. De tweede is de IEEE 796, welke is voorgedragen door Intel en afgeleid van de Multi-bus die met name door Intel al jaren werd gevoerd.

5.3.1. De IEEE 696 (S-100)

De S-100 is ingevoerd bij de introductie van de ALTAIR door de firma MITS. Dit modulaire microprocessorsysteem kan worden beschouwd als de eerste personal computer, vrijwel direct gevolgd door een vergelijkbaar model van IMSAI. Bij de opbouw van beide machines werd een 100-polige connector gebruikt en daarmee samenhangend ook een 100 lijnen brede doorverbindingbus. De grote vlucht die de ALTAIR nam, zorgde ervoor dat een groot aantal fabrikanten, interfacekaarten ging uitbrengen met dezelfde busdefinitie. Andere fabrikanten brachten hun eigen systemen uit, ook weer gebaseerd op de S-100 en zo ontstond een de facto standaard voor printkaarten. Helaas was in eerste aanleg de S 100 niet gelukkig, en ook niet volledig gedefinieerd. Zo waren er een groot aantal lijnen niet gespecificeerd en ook wat de ligging van bepaalde lijnen ten opzichte van de naastliggende lijnen betreft was de combinatie wel eens hoogst ongelukkig. Gedeeltelijk is dit te wijten aan de plaats waar de S 100 ontstond, namelijk in de hobbywereld. Toch zijn er tientallen printkaarten verkrijgbaar die, soms met kleine aanpassingen, onderling zijn te verbinden. Midden 1978 heeft een committee van IEEE een voorstel gedaan voor een standaard waarbij aan een aantal bezwaren werd tegemoet gekomen en waarbij de bus volledig werd gedefinieerd, dit was de IEEE 696. Binnen deze bus kent men twee soorten modules, meesters en slaven. Meesters zijn verantwoordelijk voor de busbesturing om een datatransport te realiseren. Er bestaan twee soorten meesters, permanente meesters en tijdelijke meesters. Een permanente meester is uniek (normaal een CPU) en vervult tevens de taak van de bus arbitrage. Een tijdelijke meester (bijvoorbeeld een DMA controller) moet de bus aanvragen aan de permanente meester vóór er gebruik van te maken. Iedere module, die geen meester is, moet voortdurend de bussignalen in de gaten houden om te zien of er niets voor hem (of haar) langskomt.

Zoals al is gezegd kent de bus 100 lijnen, deze zijn verdeeld als volgt:

- Een adresbus van 24 lijnen, alle meesters worden geacht in staat te zijn minstens 16 adreslijnen te sturen.
- Een databus van 16 lijnen. In 8-bit systemen worden dan 8 lijnen gebruikt voor

data 'in' en 8 lijnen voor data 'uit'. Bij een 16-bits systeem zijn de 16 lijnen als bidirectionele bus in gebruik.

- Een statusbus, gestuurd door de meester, aangevend welke transporten plaatsvinden. Er wordt onderscheid gemaakt tussen OPCODE FETCH, MEMORY READ en MEMORY WRITE, INPUT, OUTPUT, INTERRUPT ACKNOWLEDGE en HALT ACKNOWLEDGE. Al deze 7 signalen zijn overgenomen van de S-100 bus en gebaseerd op een type 8080 microprocessor.
- Een invoercontrole bus, 6 lijnen, toegevoegde signalen om het een slaaf mogelijk te maken data over te nemen.
- Een uitvoercontrole bus, 5 lijnen, toegevoegde signalen die de geadresseerde slaaf voorzien van extra timing informatie.
- Een DMA-controle bus, 8 lijnen met signalen welke speciaal zijn toegevoegd voor het aanvragen van toestemming voor DMA-transporten door de verschillende meesters en de toewijzing.
- Een bus voor vectored interrupt, 8 lijnen. Op deze lijnen is het mogelijk, door er een vector op te plaatsen, direct een interrupt te plegen.
- Een 26-tal lijnen onder de noemer 'divers'. Hierin zijn begrepen: de lijnen voor de voedingsspanning (+8, +16, -16), de klok, lijnen voor RESET en POWER FAIL en nog een aantal die zijn gereserveerd voor toekomstige uitbreidingen/verbeteringen.

Tabel 5.1 geeft een overzicht van de pen aansluitingen volgens de IEEE 696 standaard.

Pin no.	Signal	Description
1	+8 Volt	Unregulated power supply voltage ($> 7V$, $< 25V$).
2	+16 Volt	Unregulated power supply voltage ($> 14,5V$, $< 21,5V$).
3	XRDY	One of two READY inputs to the current bus master. The bus is ready when both these READY inputs are true (see pin 72).
4	VI 0	Vectored interrupt line 0
5	VI 1	idem 1
6	VI 2	idem 2
7	VI 3	idem 3
8	VI 4	idem 4
9	VI 5	idem 5
10	VI 6	idem 6
11	VI 7	idem 7
12	NMI	Non maskable interrupt.
13	PWRFAIL	Powerfail bus signal.
14	DMA3	Temporary master priority bit 3.
15	A18	Extended address bit 18
16	A16	Extended address bit 16
17	A17	Extended address bit 17
18	SDSB	Control signal to disable the eight status signals.
19	CDSB	Control signal to disable the five control output signals.
20	GND	Ground, common with pin nr. 100.
21	NDEF	Not defined. Manufacturer must specify any use in detail.
22	ADSB	Control signal to disable the sixteen address signals.
23	DODSB	Control signal to disable the eight data output signals.
24	O (B)	Master timing signal for the bus.
25	pSTVAL	Status valid strobe.
26	pHLDA	Control signal, controls in conjunction with HOLD, bus master transfer operations.
27	RFU	Reserved for future use.
28	RFU	idem
29	A5	Address bit 5
30	A4	idem 4
31	A3	idem 3
32	A15	idem 15
33	A12	idem 12
34	A9	idem 9

Pin no.	Signal	Description
35	DO1/DATA1	Data out bit 1, bidirectional data bit 1.
36	DO0/DATA0	Data out bit 0, bidirectional data bit 0.
37	A10	Address bit 10.
38	DO4/DATA4	Data out bit 4, bidirectional data bit 4.
39	DO5/DATA5	idem 5, idem 5
40	DO6/DATA6	idem 6, idem 6
41	DI2/DATA10	Data in bit 2, idem 10
42	DI3/DATA11	idem 3, idem 11
43	DI7/DATA15	idem 7, idem 15
44	sM1	Status signal, indicating that the current cycle is an op-code fetch.
45	sOUT	Status signal, identifying the data transfer bus cycle to an output device.
46	sINP	Status signal, identifying the data transfer bus cycle to an input device.
47	sMEMR	Status signal, identifying bus cycles which transfer data from memory to a bus master, which are not interrupt acknowledge instruction fetch cycles.
48	sHLTA	Status signal which acknowledges that a HLT instruction has been executed.
49	CLOCK(B)	2MHz (0.5%) 40–60% duty cycle. Not required to be synchronous with any other bus signal.
50	GND(B)	Ground, common with pin no. 100.
51	+8 Volt	Common with pin no. 1.
52	–16 Volt	Unregulated power supply voltage ($< -14.5\text{ V}$, $> -35\text{ V}$).
53	GND(B)	Ground, common with pin no. 100.
54	SLAVE CLR(B)	A reset signal to reset bus slaves. Must be active with POC (see pin nr. 99) and may also be generated by external means.
55	DMA0	Temporary master priority bit 0.
56	DMA1	idem 1.
57	DMA2	idem 2.
58	sXTRQ	Status signal which requests 16 bit slaves to assert SIXTN.
59	A19	Extended address bit 19.
60	SIXTN	Signal generated by 16 bit slaves in response to the signal sXTRQ.
61	A20	Extended addressbit 20.
62	A21	idem 21.
63	A22	idem 22.
64	A23	idem 23.
65	NDEF	Not defined.
66	NDEF	Not defined.
67	PHANTOM	A bus signal which disables normal slave devices and enables phantom slaves; primarily used for bootstrapping.
68	MWRT	$pWR + sOUT$ (logic equation). This signal must follow pWR by not more than 30ns.
69	RFU	Reserved for future use.
70	GND	Ground, common with pin no. 100.
71	RFU	Reserved for future use.
72	RDY	See comments for pin no. 3.
73	INT	Primary interrupt request signal.
74	HOLD	Control signal used in conjunction with pHLDA to coordinate bus master transfer operations.
75	RESET	Reset signal to reset bus master devices. This signal must be active with POC and may also be generated with external means.
76	pSYNC	Control signal identifying BS1.
77	pWR	Control signal signifying the presence of valid data on D0 bus or data bus.
78	pDBIN	Control signal requesting data on the DI bus or data bus from the currently addressed slave.
79	A0	Address bit 0.
80	A1	idem 1.
81	A2	idem 2.
82	A6	idem 6.
83	A7	idem 7.
84	A8	idem 8.
85	A13	idem 13.
86	A14	idem 14.
87	A11	idem 11.

Pin no.	Signal	Description
88	DO2/DATA2	Data out bit 2, bidirectional data bit 2.
89	DO3/DATA3	idem 3, idem 3.
90	DO7/DATA7	idem 7, idem 7.
91	DI4/DATA12	Data in bit 4, bidirectional data bit 12.
92	DI5/DATA13	idem 5, idem 13.
93	DI6/DATA14	idem 6, idem 14.
94	DI1/DATA9	idem 1, idem 9.
95	DI0/DATA8	idem 0, idem 8.
96	sINTA	Status signal identifying the bus input cycle(s) that may follow an accepted interrupt request on INT.
97	sWO	Status signal identifying a bus cycle which transfers data from a bus master to a slave.
98	ERROR	Bus status signal signifying an error condition during present bus cycle.
99	POC	Power on clear signal for all bus devices; when this signal goes low, it must stay low for at least 10 ms.
100	GND	System ground.

Tabel 5.1. Penaansluitingen van bus volgens IEEE 696.

5.3.2. De IEEE 796 bus (Multibus)

De IEEE 796 is een busdefinitie, die is afgeleid van de Multibus, een al jarenlang door Intel gebruikte busdefinitie. Deze firma maakt niet alleen circuits maar ook printkaarten, complete systemen en software. In vrijwel alle apparatuur gebruikt Intel voor de printkaarten een standaard busdefinitie. Deze Multibus is in de loop der jaren door andere fabrikanten eveneens overgenomen als zij compatibele kaarten uitbrachten voor Intel systemen. Net als de S-100 is ook de Multibus als standaard bij IEEE voorgedragen en geaccepteerd onder nummer 796. De definitie van de Multibus was beter doordacht dan de S-100. Dit blijkt onder andere uit de toewijzing van relatief hoogfrequente signalen aan pennen liggend tussen niet-actieve lijnen en de meer symmetrische ligging van de voedingslijnen. Ook zijn de lijnen active-low gecodeerd, wat resulteert in een ruimere storingsmarge.

Op deze bus kunnen twee soorten modules worden aangesloten: meesters en slaven. Meesters kunnen busbesturingssignalen genereren. Vóór een meester de bus gebruikt, moet dat worden gevraagd aan een arbiter. Deze kent vaste en/of programmeerbare prioriteiten en kan de bus voor één of meer transfers aan een meester toewijzen. Arbitrage vindt plaats aan de hand van een bus-kloksignaal. Synchron met deze klok wordt bepaald of en welke module de bus krijgt toegewezen. Slaven moeten continu de bus bekijken en mogen alleen maar reageren als dat gewenst is.

De IEEE 796 kent in totaal 86 lijnen die als volgt verdeeld zijn:

- Een adresbus van 20 lijnen (binnenkort uitgebreid naar 24).
- Een bidirectionele databus van 16 lijnen.
- Een achttal buscontrole signalen voor de begeleiding van datatransport over de bus.
- Interruptafhandelingsignalen, acht stuks voor parallel interrupt request en één voor de acknowledge.
- Bus arbitrage lijnen, zes lijnen waarmee de aanvraag en toewijzing van de bus wordt geregeld.
- Overige lijnen, waaronder onder andere vallen: voedingslijnen, systeemklok en gereserveerde lijnen voor toekomstige uitbreidingen.

In tabel 5.2 is een overzicht gegeven van de aansluitingen van de pennen volgens de IEEE 796 standaard.

Group	Pin no.	Signal	Description	
Power supplies	1	GND	Signal ground	
	2	GND	Signal ground	
	3	+5 V	+5 Volt dc	
	4	+5 V	+5 Volt dc	
	5	+5 V	+5 Volt dc	
	6	+5 V	+5 Volt dc	
	7	+12 V	+12 Volt dc	
	8	+12 V	+12 Volt dc	
	9	RFU	Reserved for future use	
	10	RFU	Reserved for future use	
	11	GND	Signal ground	
	12	GND	Signal ground	
Bus controls	13	BCLK	Bus clock.	
	14	INIT	Initialize.	
	15	BPRN	Bus priority in	
	16	BPRO	Bus priority out	
	17	BUSY	Bus busy	
	18	BREQ	Bus request	active
	19	MRDC	Memory read command	low
	20	MWTC	Memory write command	
	21	IORC	I/O read command	
	22	IOWC	I/O write command	
	23	XACK	Transfer acknowledge	
	24	INH1	Inhibit 1 (disable RAM)	
Bus controls and addresses	25	RFU	Reserved for future use/Lock/	
	26	INH2	Inhibit 2 (disable ROM)	
	27	BHEN	Byte high enable	
	28	A16	Address bit 16.	
	29	CBRQ	Common bus request.	active
	30	A17	Address bit 17	low
	31	CCLK	Constant clock.	
	32	A18	Address bit 18.	
	33	INTA	Interrupt acknowledge	
	34	A19	Address bit 19	
Interrupts	35	INT6	Parallel interrupt requests	
	36	INT7	idem	
	37	INT4	idem	
	38	INT5	idem	active
	39	INT2	idem	low
	40	INT3	idem	
	41	INT0	idem	
	42	INT1	idem	
Addresses	43	A14	Address bit 14	
	44	A15	idem 15.	
	45	A12	idem 12.	
	46	A13	idem 13	
	47	A10	idem 10.	
	48	A11	idem 11	
	49	A8	idem 8.	
	50	A9	idem 9	active
	51	A6	idem 6.	low
	52	A7	idem 7.	
	53	A4	idem 4.	
	54	A5	idem 5.	
	55	A2	idem 2	
	56	A3	idem 3	
	57	A0	idem 0	
	58	A1	idem 1	
Data	59	D14	Data bit 14	
	60	D15	idem 15	
	61	D12	idem 12	
	62	D13	idem 13	
	63	D10	idem 10	

Group	Pin nr.	Signal	Description
	64	D11	Data bit 11
	65	D8	idem 8
	66	D9	idem 9
	67	D6	idem 6
	68	D7	idem 7
	69	D4	idem 4
	70	D5	idem 5
	71	D2	idem 2
	72	D3	idem 3
	73	D0	idem 0
	74	D1	idem 1
Power supplies	75	GND	Signal ground
	76	GND	Signal ground
	77	RFU	Reserved for future use
	78	RFU	Reserved for future use
	79	-12 V	-12 Volt dc
	80	-12 V	-12 Volt dc
	81	+5 V	+5 Volt dc
	82	+5 V	+5 Volt dc
	83	+5 V	+5 Volt dc
	84	+5 V	+5 Volt dc
	85	GND	Signal ground
	86	GND	Signal ground

Tabel 5.2. Penaansluitingen volgens IEEE796.

5.3.3. Overige bussen

Naast de twee hiervoor genoemde bussen zijn er nog een aantal andere in gebruik. Iedere firma die printkaarten levert doet dit volgens een eigen definitie. Immers, ieder heeft wel opmerkingen over de busdefinitie van de anderen. Zo zijn er bussen van Philips, Rockwell, Zilog, Motorola, DEC, RCA, enzovoorts. Voor iedere bus brengt de fabrikant een aantal compatibele kaarten uit; voor een afnemer betekent dit, dat hij vastzit aan die definitie en dus aan die leverancier.

In het begin van 1982 zijn een aantal fabrikanten tot een gezamenlijke definitie gekomen van een bus, die zij allen zullen gaan ondersteunen. Dit zijn: Philips (Signetics), Motorola, Mostek en Thomson-CSF. De bus heet VME-bus en is gebaseerd op printkaarten van het Eurokaart formaat (DIN 41612 en 41494). Zowel enkele (160×100 mm) als dubbele (160×233,4 mm) Eurokaarten worden ondersteund. De bus is geschikt voor 16 en 32 bits (multi-) processorsystemen. Een enkele Eurokaart voert alle lijnen voor een 16 bits systeem via een connector P1, een dubbele Eurokaart kent additionele signalen voor uitbreiding tot een 32 bits data en 32 bits adres systeem. Op die tweede connector zijn dan tevens nog, naar keuze, 64 toegevoegde I/O-lijnen te gebruiken. Iedere connector heeft drie rijen van 32 aansluitpennen, respectievelijk rij A, B en C. De VME-bus is asynchroon en vraagt bevestiging van iedere data transfer over de bus. De bus kent vier niveau's van bus arbitrage, eventueel met daisy-chain prioriteit per niveau en een door de gebruiker te definiëren prioriteit tussen de niveau's onderling. Om een optimaal gebruik van de bus bandbreedte te maken in een multi-processor systeem, is tevens voorzien in een aparte, seriële link voor klok en data, tussen kaarten onderling. De read-modify-write cyclus is ondeelbaar en laat daardoor het gebruik van semaforen toe bij multi-processor systemen. Data transport is mogelijk met een snelheid van 5M 32-bit transporten per seconde ofwel 20Mbyte/s. De bus is bij IEEE voorgedragen als standaard en zal daar zeker invloed hebben op de, nu in studie zijnde, *future bus* onder nummer P896.

De verdeling van de signalen op de bus is weergegeven in tabel 5.3.

Group	Signal	Description
Connector P1: Primary Signals		
Address bus	A0 to A23	Primary address range = 16 Mbytes.
Data bus	D0 to D15	Primary data width = 16 bits.
Address control	AS	Address strobe.
	AM0 to AM5	Address modifiers (extended address, short I/O address, block transfer, supervisor/user, data/program, user definable options).
Data control	DS0, DS1	Data strobes (upper/lower byte, word select).
	WRITE	Data direction.
	DTACK	Data transfer acknowledge.
	LWORD	Long word (32 bit) data transfer.
Interrupts	IRQ1 to IRQ7	Interrupt request, 7 levels.
	IACK, IACKIN, IACKOUT	Interrupt acknowledge and daisy-chain.
Bus arbitration	BR0 to BR3	Bus request, 4 levels.
	BG0IN to BG3IN	Bus grant IN
	BG0OUT to BG3OUT	Bus grant OUT for daisy-chain
Interprocessor link	SERDAT	Serial data
	SERCLK	Serial clock
Error signals	SYSFAIL	System failure
	BERR	Bus error
	ACFAIL	Power failure
Power	+5, +/-12 V, 5 standby, gnd	Includes standby power line for battery back-up.
Miscellaneous	SYSCLK	16MHz clock.
	SYSRESET	System reset.
Connector P2: Expansion Signals		
Extended address bus	A24 to A31	Extended address range = 4 Gbytes.
Extended data bus	D16 to D31	Extended data width = 32 bits.
Power	+5, gnd	Additional power and ground lines.
Input/output	User I/O	64 pins in the two outside rows A and C available for I/O functions.

Tabel 5.3. Verdeling van de signalen op de VME-bus.

Als de plannen van de bovengenoemde fabrikanten doorgaan en de bus wordt ook nog als IEEE standaard geaccepteerd in de voorgestelde vorm, dan is er een aanzienlijke kans dat de VME-bus binnen enkele jaren de standaard zal zijn voor bus-georiënteerde processor systemen.



6. Interfacing

6.1. Inleiding

Een computer zonder randapparatuur is ondenkbaar en zinloos. Er zal altijd een uitwisseling van gegevens plaatsvinden tussen de computer en andere apparatuur. Wat heeft men aan een computer, die informatie binnenhaalt, deze informatie verwerkt en aan de buitenwereld niet kan melden dat de informatie is ontvangen en verwerkt. Bovendien wenst de buitenwereld de resultaten van de verwerkte informatie terug te ontvangen. De computer zal daarom een of meer in- en uitvoerkanalen moeten hebben. Communiceren betekent voor de computer informatie van en naar het geheugen brengen, gebeurt het communiceren met de buitenwereld dan spreekt men over Input en Output, I/O.

In dit hoofdstuk zullen drie veel toegepaste in- en uitvoermethoden aan de orde komen, te weten:

- I/O onder programmabesturing,
- I/O onder programmabesturing en interrupt,
- I/O onder hardware besturing.

Een ander onderscheid in I/O is te maken aan de hand van de selectie van de I/O-kanalen. Men kan een I/O-kanaal beschouwen als een geheugenplaats waarin geschreven en waaruit gelezen kan worden. De I/O-kanalen liggen dan in het adresveld van het geheugen en men spreekt over *Memory Mapped I/O*.

Veel processoren beschikken over speciale I/O-instructies welke het datatransport via de I/O-kanalen verzorgen. De processor heeft een M/IO-uitgang die bij deze instructie aangeeft dat het om een I/O-actie gaat en niet om een geheugen-access. Omdat de I/O-kanalen niet in het adresveld van het geheugen liggen, spreekt men van *Isolated I/O* of *Standard I/O*.

6.2. Wat is een interface

In de wereld van de automatisering wordt onder het begrip interface niet altijd hetzelfde verstaan. Daarom hier de afspraak dat de interface in de microcomputer de taak heeft de aanpassing te verzorgen tussen het interne communicatiekanaal van de processor met de communicatiekanalen van de buitenwereld. De aanpassing kan op de volgende punten nodig zijn.

1. Signaalmedium

De databus van de computer maakt gebruik van spanningsniveau's, vaak TTL compatibel, om een logische '0' of '1' aan te geven. In het communicatiekanaal buiten de computer kunnen andere media worden toegepast, zoals:

- afwijkende spanningsniveau's,
- elektrische stroom,
- frequentiemodulatie,
- licht (opto-elektronisch),
- lucht-/vloeistofdruk.

2. Synchronisatie

Het tijdstip, waarop twee communicerende systemen informatie aanbieden en overnemen, zal nooit gelijk zijn. Er zal een synchronisatie en een aanpassing van transmissiesnelheid gerealiseerd moeten worden.

3. Datapadbreedte

De breedte van het datapad in de microcomputer ligt vast en hangt af van het type

CPU, voor de buitenwereld kan de breedte variëren. Dit betekent dat de interface de data 'in en uit moet pakken' om de gewenste dataformaten op elkaar af te stemmen.

In dit hoofdstuk zal nader ingegaan worden op de interfacing betreffende synchronisatie en datapadbreedte.

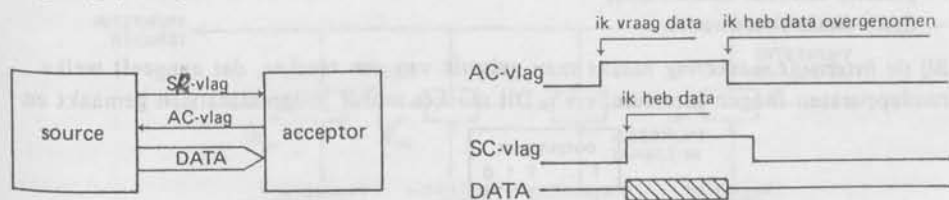
6.3. I/O onder programmabesturing

De in- en uitvoer van informatie wordt door een speciaal I/O-programma geregeld. Dit programma voert datatransport via de I/O-kanalen uit. Ieder I/O-kanaal krijgt een adres toegewezen, hetgeen in het geheugen- of I/O-adresveld ligt.

Het eenvoudigste I/O-kanaal heeft een minimum aan intelligentie en kan slechts informatie doorgeven. Dit kanaal wordt toegepast voor *Statische I/O*, transport van informatie die een toestand aangeeft en op gezette tijden opgevraagd kan worden, bijvoorbeeld een reeks vlaggen of schakelaarstanden. Deze vorm van I/O wordt vaak in een *ongeconditioneerde* vorm bedreven, het programma bepaalt wanneer de I/O plaatsvindt. Voor de invoer van statische informatie kan volstaan worden met een eenvoudige busdriver (zie voor de interface-circuits hoofdstuk 7) zonder geheugenfunctie, de processor selecteert het betreffende invoerkanaal, de busdriver wordt door de selectielogica geactiveerd en zet de informatie van de buitenwereld op de interne bus. Omgekeerd zal voor uitvoer van statische informatie wel een geheugenfunctie nodig zijn, immers het selecteren van een I/O-kanaal duurt maximaal een processorcyclus.

Bij ongeconditioneerde I/O weet de processor niet of de uitgevoerde informatie door de buitenwereld al geraadpleegd is en de buitenwereld weet niet of de processor de in te voeren informatie al verwerkt heeft. Bij *geconditioneerde* I/O wordt door het opzetten van vlaggen zowel naar buiten als naar binnen bekend gemaakt of I/O plaatsgevonden heeft. Deze conditievlaggen zelf zijn een sprekend voorbeeld van ongeconditioneerde statische I/O!

Met behulp van deze conditievlaggen kan een *handshake* procedure opgezet worden om informatie-uitwisseling tussen processor en buitenwereld te realiseren. Het principe van de handshake is dat de *acceptor* vraagt (met de AC-vlag) om data. De *source* detecteert dit en meldt met de SR-vlag wanneer hij data beschikbaar heeft. De acceptor test op deze vlag, neemt de data over en meldt met het weghalen van de AC-vlag terug, wanneer hij dit gedaan heeft. Hierop test de source en haalt vervolgens data en de SR-vlag weg (zie figuur 6.1).



Figuur 6.1. Handshake principe.

Wie source of acceptor is hangt af van de richting van datatransport, bij informatie-uitvoer is de processor source en bij invoer is de processor acceptor. Voor iedere datatransportrichting zijn twee extra statische I/O-lijnen nodig, daarom is het verstandig om meer intelligente interface circuits te gebruiken met control- en statusregisters aan boord, die door de processor opgevraagd en ingesteld kunnen worden.

Deze interface circuits hebben meestal de mogelijkheid van *Strobed I/O*, waarbij met

behulp van *Strobe*-pulsen kenbaar gemaakt wordt wanneer informatie beschikbaar of overgenomen is. Deze pulsen zorgen in het circuit voor het zetten danwel resetten van de benodigde vlaggen.

6.4. I/O onder programmabesturing en interrupt

Werkt men met geconditioneerde I/O dan kan men in plaats van de signaalvlaggen programmatisch te testen, ook gebruik maken van het interruptmechanisme. De vlaggen activeren de interruptrequest zodra er zich een verandering voordoet en de processor kan dan onmiddellijk reageren op iedere verandering. Toch is het niet wenselijk om direct op iedere verandering te reageren, de processor kan met een belangrijke I/O-afhandeling bezig zijn die niet onderbroken mag worden. Om dit te voorkomen zijn er systemen bedacht, die regelend optreden wanneer een aantal interfaces of randapparaten de processor proberen te interrumperen. Al deze systemen zijn er op gericht om een bepaalde prioriteit in de afhandeling te realiseren. Hierbij zijn twee soorten priority-interruptafhandeling te onderscheiden, namelijk:

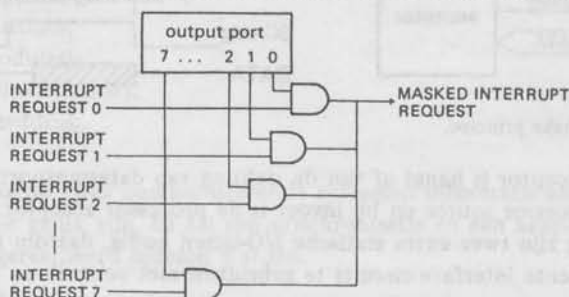
- de processor wordt eerst geïnterrupteerd alvorens de prioriteit van de afhandeling vast te stellen,
- echter bij de zuivere priority-interruptafhandeling wordt er voor gezorgd dat de processor niet geïnterrupteerd kan worden zolang een interruptafhandeling van een hogere prioriteit niet afgerond is.

Bij de interruptafhandeling met een zoekprogramma (*polling*) wordt de processor eerst in zijn werkzaamheden gestoord om vervolgens via het zoekprogramma in een bepaalde volgorde (*prioriteit*) te onderzoeken welk randapparaat geïnterrupteerd heeft. Komt er vlak hierop weer een interrupt maar van een lagere prioriteit dan zal het afhandelingsprogramma waar de processor mee bezig was worden onderbroken, en het zoekprogramma wordt opnieuw doorlopen tot het randapparaat waar de processor mee bezig is en de processor gaat vervolgens weer met dit randapparaat verder. Is de gegeneerde interrupt van een hogere prioriteit dan zal de processor met de afhandeling van het randapparaat met de hogere prioriteit verder gaan.

Het idee van een zuivere priority-interruptafhandeling is de interrupt-request naar de processor te *blokkeren* indien de processor met een belangrijker I/O-afhandeling bezig is. Enkele veel gebruikte methoden voor deze prioriteitsregeling zijn:

- interrupt masking (hard- en software),
- priority encoder (hardware),
- daisy chain (hardware).

Bij de *interrupt masking* maakt men gebruik van een masker, dat aangeeft welke randapparaten mogen interrumperen. Dit masker wordt programmatisch gemaakt en

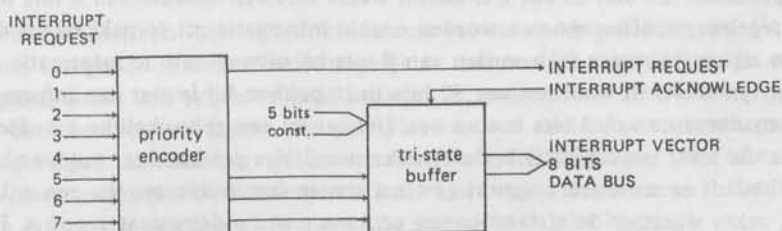


Figuur 6.2. Interrupt masking.

door middel van een Statische I/O via een I/O-poort naar buiten gebracht (zie figuur 6.2). Iedere I/O-lijn zorgt voor het al of niet doorgeven van de interrupt-request van het betreffende randapparaat naar de processor.

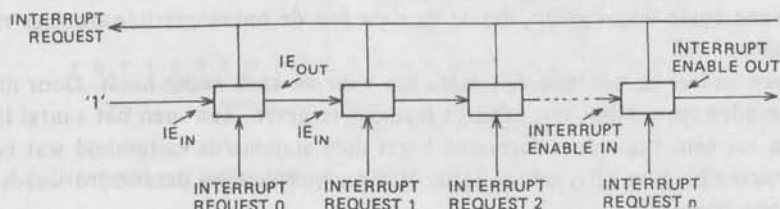
Na iedere interruptafhandeling zal het masker bijgewerkt moeten worden.

Een voorbeeld voor het gebruik van een *priority encoder* is in figuur 6.3 aangegeven. De encoder is een stuk hardware waarin de prioriteit vast ingebakken is. De inkomende interruptlijnen worden bij een request op hun prioriteit beoordeeld, is deze hoger dan wordt de aanvraag gehonoreerd en doorgegeven naar de processor. In dit voorbeeld is een encoder voor 8 lijnen getekend. Aan de uitgang komt gecodeerd het lijnnummer dat met 5 al of niet instelbare bitwaarden de interruptvector vormt die, wanneer de processor een Interrupt Acknowledge genereert, op de databus gezet wordt als referentie voor de interrupt service routine.



Figuur 6.3. Priority interrupt encoder.

Tenslotte de *daisy chain*, een prioriteitsketen die onderbroken wordt op de plaats waar een randapparaat een interrupt-request heeft geplaatst. Wil men de prioriteitsgedachte zuiver houden, dan mag deze schakel pas weer hersteld worden nadat de betreffende interrupt-request geheel is afgehandeld. Zilog, de fabrikant van de Z80 processor, heeft daartoe in haar interface-circuits die met een Daisy Chain zijn uitgerust een databus-loerder ingebouwd, deze kijkt naar het moment waarop de *Return from Interrupt (RETI)*-instructie langs komt. Met deze instructie wordt alleen de onderbroken schakel met de hoogste prioriteit hersteld, er kunnen immers nog interruptaanvragen van een lagere prioriteit zijn die op afhandeling wachten. Apparaten die in de keten dicht bij het begin liggen hebben een hogere prioriteit. Apparaten die voorbij de eerst verbroken schakel willen interrupteren krijgen geen toegang tot de requestlijn van de processor, maar onderbreken wel de keten op de overeenkomstige plaats.



Figuur 6.4. Daisy Chain.

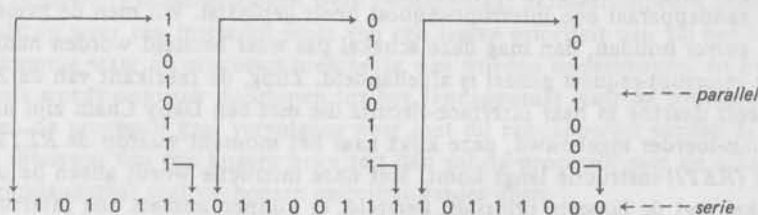
6.5. I/O onder hardware besturing

De in dit hoofdstuk behandelde I/O-methoden hebben de processor nodig om de communicatie te kunnen realiseren. In deze voorbeelden liep alle te transporteren data via een register in de CPU. Bij uitvoer van informatie haalt de processor de data per woord uit het geheugen, plaatst dit in een intern register en brengt het vervolgens naar het

I/O-kanaal. Omgekeerd wordt bij invoer van informatie de data per woord uit het I/O-kanaal opgehaald en via het interne register van de CPU naar de gewenste geheugenplaatsen overgebracht. Hier heeft de processor veel werk aan en het kost veel tijd. Effectiever is het daarom gebruik te maken van een DMA-controller die de communicatie buiten de processor om verzorgt. Enkele methoden van DMA zijn in hoofdstuk 3 besproken. Daar een DMA-controller vrij complex en relatief duur is, kan als mogelijke oplossing het gebruik van een speciale I/O-processor overwogen worden, een tweede CPU met software die de datacommunicatie in blokken verzorgt.

6.6. Seriële communicatie

Naast de aanpassing van de processor met de buitenwereld wat de synchronisatie betreft, is er ook een aanpassing nodig voor de datapadbreedte. Is de datapadbreedte van de processor 32 bits en het I/O-kanaal werkt met een breedte van 8 bits dan zullen er maatregelen getroffen moeten worden om de informatie uit te pakken uit de 32 bits woorden en om te zetten in woorden van 8 bits bij uitvoer van de informatie en de woorden van 8 bits in woorden van 32 bits in te pakken bij invoer van informatie. Een datapadbreedte van 8 bits is voor een I/O-kanaal een gebruikelijke breedte, daar veel data uit tekst (dus ASCII) bestaat, waarvoor 8 bits per karakter voldoende zijn. Vaak is het uit economisch oogpunt gewenst om te communiceren via een enkele datalijn, zeker wanneer de afstand tussen processor en randapparaat groot is. In de interface dient dan een omzetting van parallel naar serie en omgekeerd plaats te vinden. Deze omzetting is programmatisch te verwezenlijken, maar kost de processor tijd en werk. Beter is om voor de omzetting een seriële communicatiecircuit te nemen.

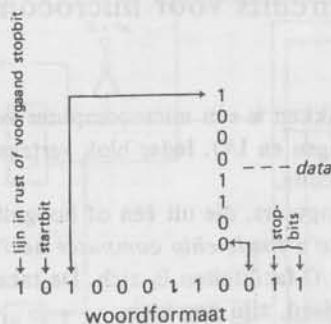


Figuur 6.5. Relatie tussen parallelle en seriële datawoorden.

In figuur 6.5 is de relatie weergegeven tussen drie 8-bits parallelle woorden naar serie. Zendt men deze data seriël over een lijn in combinatie met een besturingslijn voor de zenderkloppuls en zijn de looptijden (vooral van belang bij hoge transmissiesnelheden) voor beide lijnen gelijk, dan is de data aan de ontvangstzijde storingsvrij binnen te halen.

Echter een nadeel is, dat men een extra lijn voor de klok nodig heeft. Door nu aan de over te zenden informatie een bepaald formaat te geven, kan men het aantal lijnen beperken tot één. Voor deze formaten heeft men standaard vastgesteld wat betreft transmissiesnelheid en bitsynchronisatie. Het oorspronkelijke datawoord wordt ingepakt in een frame.

Men onderscheidt twee basistypen voor seriële datatransmissie, synchrone en asynchrone transmissie. Van deze typen is nog een derde afgeleid, het isochrone. Bij de *asynchrone* datatransmissie wordt de data per woord van maximaal 8 bits overgezonden. De data is in een frame ingepakt. Dit frame begint met een startbit, ook wel startelement genoemd. Hierna volgen vijf tot acht databits, afhankelijk van het gebruik, eventueel een checkbit (even of oneven parity) en ter afsluiting van het frame één, anderhalf of twee stopelementen (zie figuur 6.6).

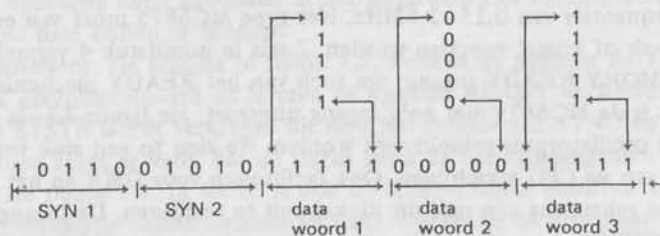


Figuur 6.6. Voorbeeld van een asynchroon dataformaat met een startbit, 8 databits en twee stopbits.

Elk element heeft een vaste tijdsduur. In rust zal het zendsignaal '1' zijn. Het startbit is '0' om een goede synchronisatie mogelijk te maken. Het stopbit is gelijk aan het rustsignaal, zodat ook synchronisatie mogelijk is wanneer een frame direct gevolgd wordt door een volgend frame. Het paritybit kan een even of een oneven parity vertegenwoordigen. Dit betekent dat in een frame respectievelijk een even of een oneven aantal enen voorkomt. De ontvangende kant kan hierop testen en weet dat de overgezonden informatie verminkt is wanneer de parity-check niet klopt.

Het frame eindigt op stopelementen. In het verleden hadden randapparaten (Teletype's, printers) op grond van mechanische beperkingen behoefte aan meer dan één stopelement, men gaf dan anderhalf stopelement mee. Bij sturing uit een elektronische schakeling maakte men daar twee elementen van.

Bij de *synchrone* datatransmissie worden de datawoorden in blokken gegroepeerd alvorens ze worden overgezonden. Ieder blok informatie begint met één of meer synchronisatiewoorden die elk uniek zijn, dat wil zeggen, ze komen niet in de over te zenden data voor. De ontvanger zal binnengekomen informatie onderzoeken op de synchronisatiekarakters en schakelt daarna over op ontvangst van data (zie figuur 6.7). Indien er bij de zender stagnatie in de toelevering van data optreedt of in perioden van rust, zal de zender overgaan op het uitzenden van synchronisatiekarakters.



Figuur 6.7. Voorbeeld van een synchroon dataformaat.

De *isochrone* vorm van datatransmissie maakt gebruik van de methode van inpakken van data zoals bij de asynchrone datatransmissie gebeurt, dus met start- en stopbits, terwijl de klok van de ontvanger in de pas wordt gehouden door de 0→1 en 1→0 overgangen in de elementenstroom.

7. Ondersteuningscircuits voor microcomputers

7.1. Inleiding

In de voorgaande hoofdstukken is een microcomputer voorgesteld door een aantal blokken, zoals CPU, geheugen en I/O. Ieder blok vertegenwoordigt een groot aantal logische en functionele circuits.

Toch zijn er ook microcomputers, die uit één of hooguit twee chips bestaan, zie daarvoor paragraaf 10.4. Zo'n *single chip computer* heeft een geringe hoeveelheid RAM, ROM en minimale I/O-faciliteiten in zich. De taken, die op dit moment voor deze computers zijn weggelegd, zijn nog klein.

Microcomputers van een grotere configuratie zijn opgebouwd uit meer elementen dan alleen CPU, geheugen en I/O. In dit hoofdstuk zal een aantal van deze elementen nader belicht worden. We kunnen daarbij een onderscheid maken in:

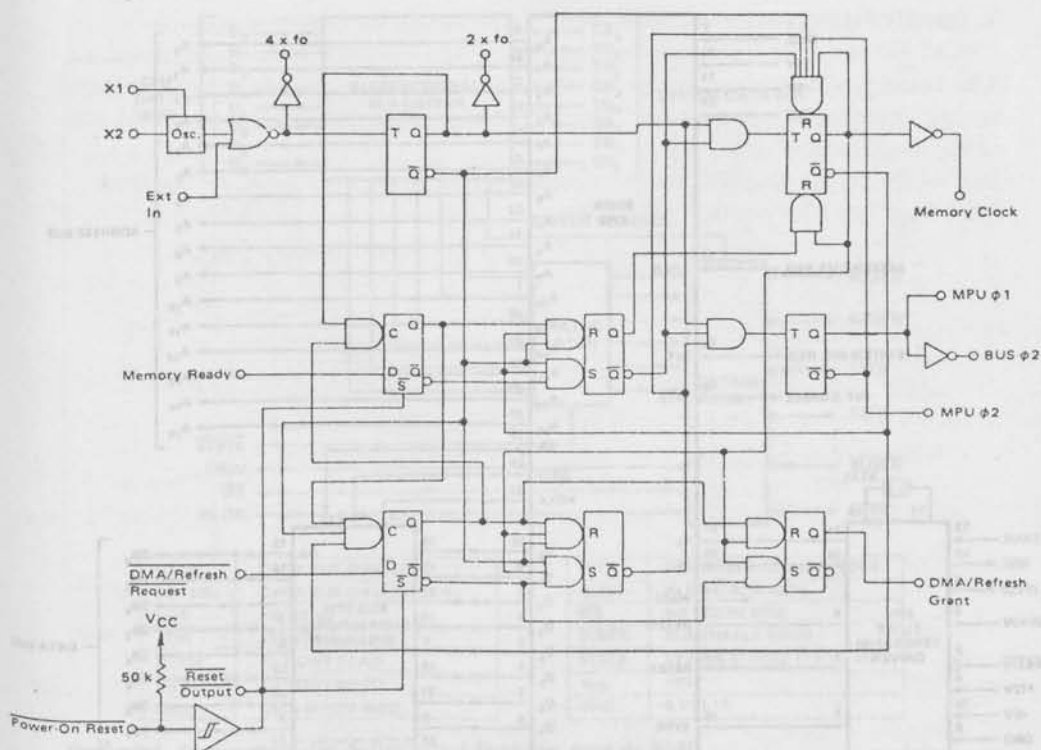
1. elementen, nodig voor het goed functioneren van het interne systeem;
2. elementen voor de interfacing met de buitenwereld;
3. elementen, die bepaalde geprogrammeerde taken van de computer kunnen overnemen.

Iedere microprocessor fabrikant heeft er voor gezorgd, dat de gebruikers kunnen beschikken over hulpcircuits om op een eenvoudige en efficiënte wijze de gewenste configuratie samen te stellen. Het is helaas zo, dat de hulpcircuits voor het ene type processor niet altijd op eenvoudige en elegante wijze te gebruiken zijn bij het andere type. Dat deze behoefte wel bestaat komt, omdat er fabrikanten zijn, die een uitgebreid assortiment slimme hulpcircuits kunnen leveren, die de gebruikers willen toepassen bij de door hen gekozen slimme microprocessor.

7.2. Hulpcircuits voor de interne besturing

7.2.1. Circuits rond de CPU

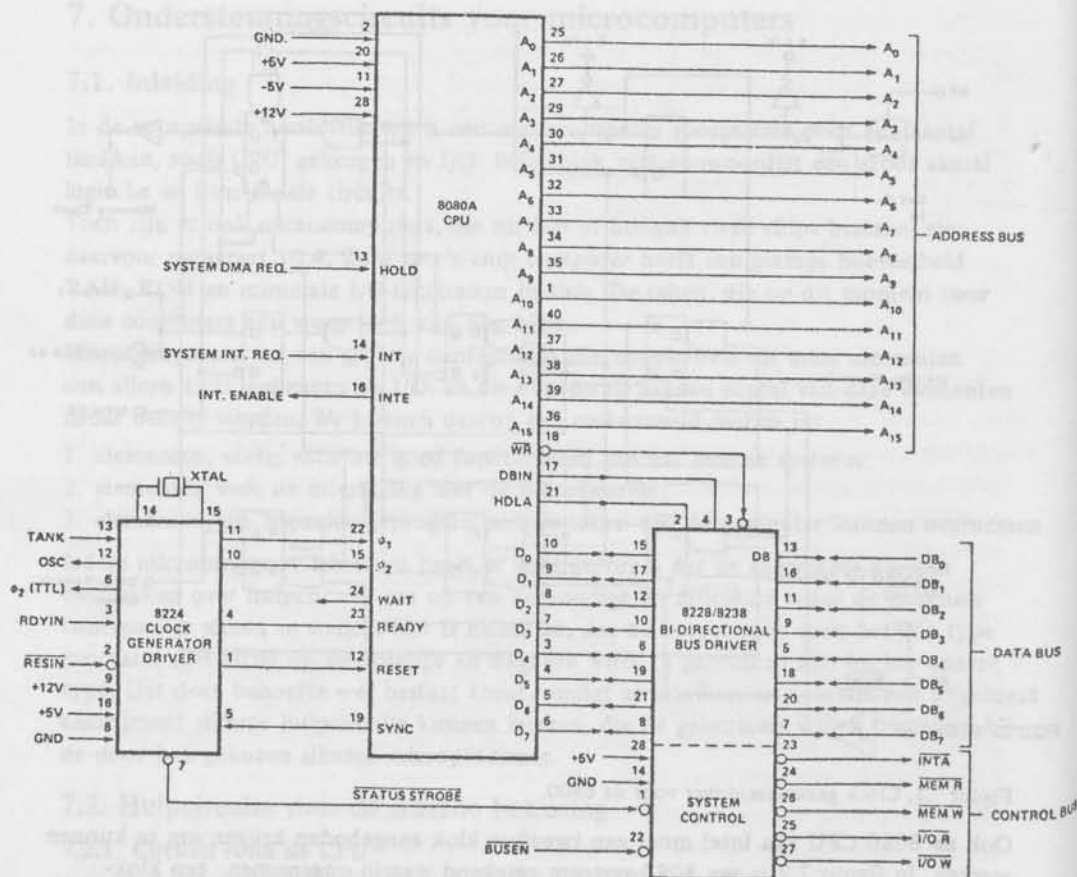
Bij sommige typen micro's is het genereren van de processorklok een uitwendige zaak, hetgeen extra hardware kost. Een 6800 en 6801 hebben beide tweefase klokpulsen CLK1 en CLK2 nodig, die elkaar niet mogen overlappen. Motorola heeft om deze klokpulsen te realiseren verschillende clock generator/driver circuits op de markt gebracht. De typen MC6870 en MC6871 A of -B hebben een ingebouwd kristal, te bestellen voor frequenties van 0,25–2,5 MHz. Het type MC6875 moet van een externe klok, RC-netwerk of kristal voorzien worden. Zoals in hoofdstuk 4 vermeld, heeft de 6800 geen MEMORY READY ingang; om toch van het READY mechanisme gebruik te kunnen maken is de MC6875 met zo'n ingang uitgerust, zie figuur 7.1. Is deze ingang laag dan zal de oscillatorpuls geblokkeerd worden. We zien zo een stuk besturing aan de buitenzijde van de CPU verschijnen. Ook faciliteiten voor DMA en het refreshen van dynamische geheugens zijn met dit klokcircuit te realiseren. De ingang DMA/REFRESH REQUEST zorgt voor het ophouden van de processorklok en genereert op het juiste moment een DMA/REFRESH GRANT, een antwoord, waarop DMA of refresh kan plaatsvinden.



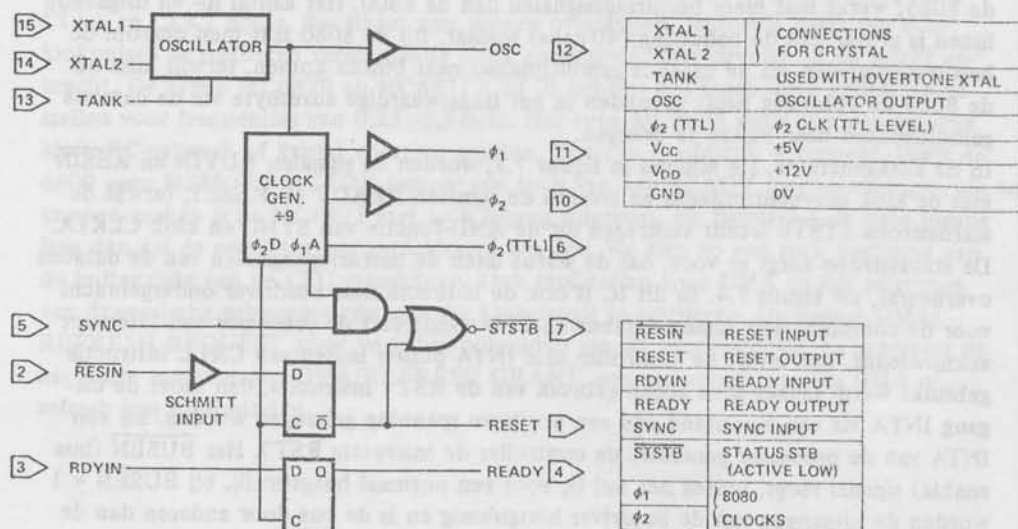
Figuur 7.1. Clock generator/driver voor de 6800.

Ook de 8080 CPU van Intel moet een tweefase klok aangeboden krijgen om te kunnen werken. In figuur 7.2 is een 8080-systeem getekend waarin opgenomen: een klok-generator/driver (8224) en een systeemcontroller/busdriver (8228). De 8080, evenals de 8085, werkt met meer besturingssignalen dan de 6800. Het aantal in- en uitgaande lijnen is groter dan de behuizing (40 pennen) toelaat. Bij de 8080 laat men daarom de besturingssignalen via de databus gemultiplexed naar buiten komen, terwijl men bij de 8085 de oplossing heeft gevonden in het laagstwaardige adresbyte via de databus gemultiplexed naar buiten te brengen.

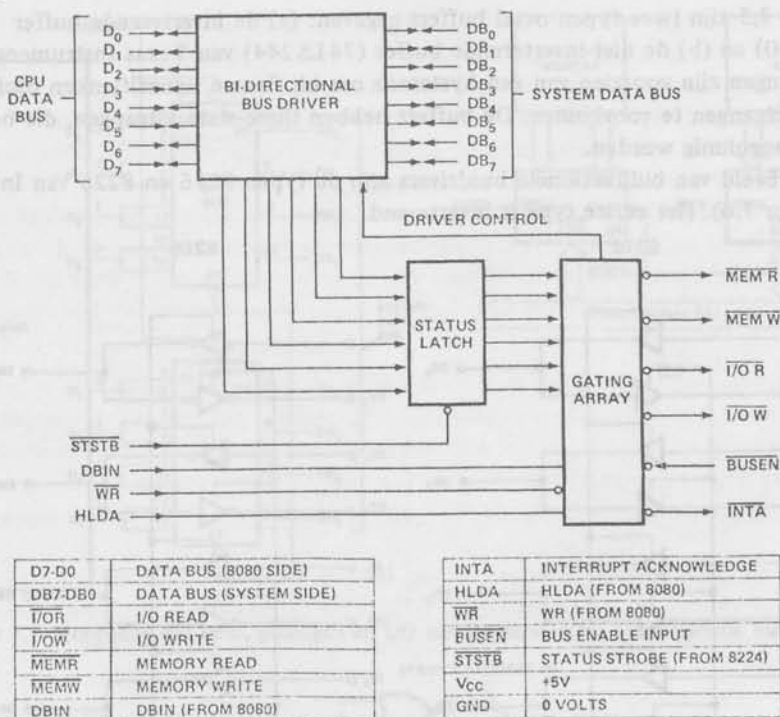
In de klokgenerator, zie schema in figuur 7.3, worden de signalen RDYIN en $\overline{\text{RESIN}}$ met de klok gesynchroniseerd en leveren de signalen READY en RESET, terwijl de statusstrobe $\overline{\text{STSTB}}$ wordt verkregen uit de AND-functie van SYNC en klok CLK1A. De statusstrobe zorgt er voor, dat de status latch de besturingssignalen van de databus overneemt, zie figuur 7.4. In dit IC is ook de bidirectionele busdriver ondergebracht voor de communicatie tussen databus en CPU. Genereert de processor een interrupt acknowledge, dan levert de controller drie INTA pulsen indien een CALL instructie gebruikt werd, maakt men alleen gebruik van de RST7 instructie, dan moet de uitgang INTA via een weerstand aan een positieve spanning gehangen worden. Bij een INTA van de processor genereert de controller de instructie RST7. Het $\overline{\text{BUSEN}}$ (bus enable) signaal zorgt, indien het nul is, voor een normaal busgebruik, bij $\overline{\text{BUSEN}} = 1$ worden de uitgangen van de busdriver hoogohmig en is de bus door anderen dan de CPU te gebruiken.



Figuur 7.2. Een 8080-processor systeem.



Figuur 7.3. Clock generator/driver voor de 8080.

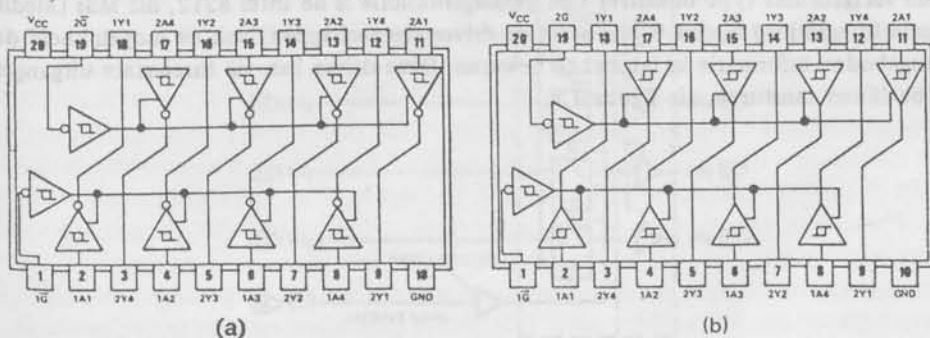


Figuur 7.4. Systemcontroller/bidirectionaal busdriver voor de 8080.

7.2.2. Busdrivers

In een klein processorsysteem zijn intern nauwelijks busdrivers nodig, daar het op de bus aangesloten aantal IC's voldoende klein is om de busbelasting binnen de gestelde specificaties van de sturende onderdelen te houden. Bij grotere systemen zal men voor versterking moeten zorgen. De fabrikanten hebben daarin voorzien door *busdrivers* te maken. Busdrivers zijn parallelle interface IC's, die zowel voor de koppeling van CPU, geheugen, I/O en dergelijke met de interne bussen gebruikt kunnen worden, alsook voor de koppeling van de interne databus met de buitenwereld.

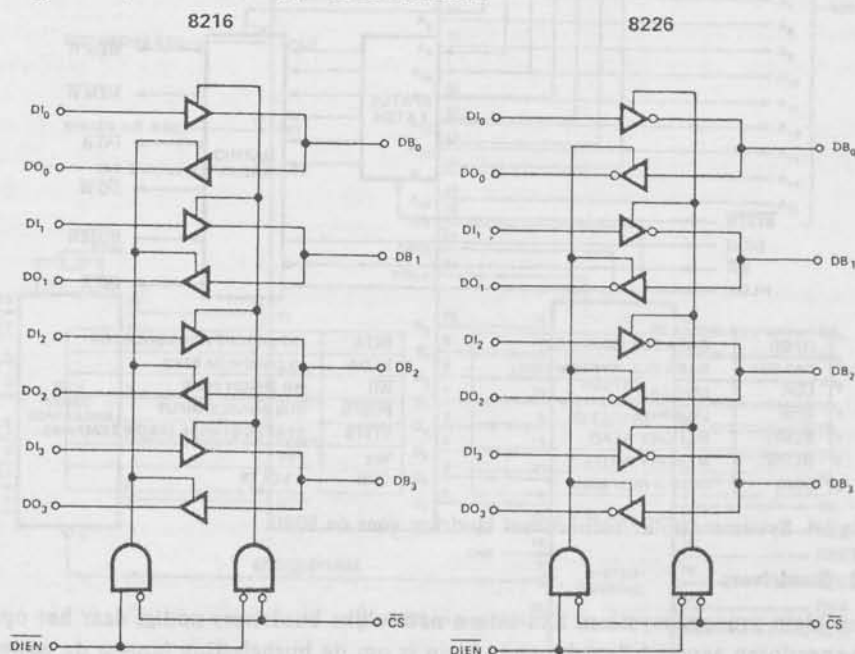
Busdrivers zijn er in diverse uitvoeringen, ze hebben alle de functie om informatie versterkt door te geven, het kenmerk is een lage fanin en een hoge fanout. Bij disabling worden de uitgangen hoogohmig om andere drivers gelegenheid te geven de bus aan te sturen. Er zijn uitvoeringen voor 4 of 8 bits parallel, al dan niet bidirectioneel, en al dan niet voorzien van latches om de informatie geklokt in over te nemen.



Figuur 7.5. Twee typen octaal buffers van TI.

In figuur 7.5 zijn twee typen octal buffers gegeven: (a) de inverterende buffer (74LS240) en (b) de niet-inverterende buffer (74LS244) van Texas Instruments. Alle ingangen zijn voorzien van een hysteresis om bij flauwe signaalfanken oscillaties aan de uitgangen te voorkomen. De buffers hebben three-state uitgangen, die bij disabling hoogohmig worden.

Een voorbeeld van bidirectionele busdrivers zijn de typen 8216 en 8226 van Intel (zie figuur 7.6). Het eerste type is inverterend.

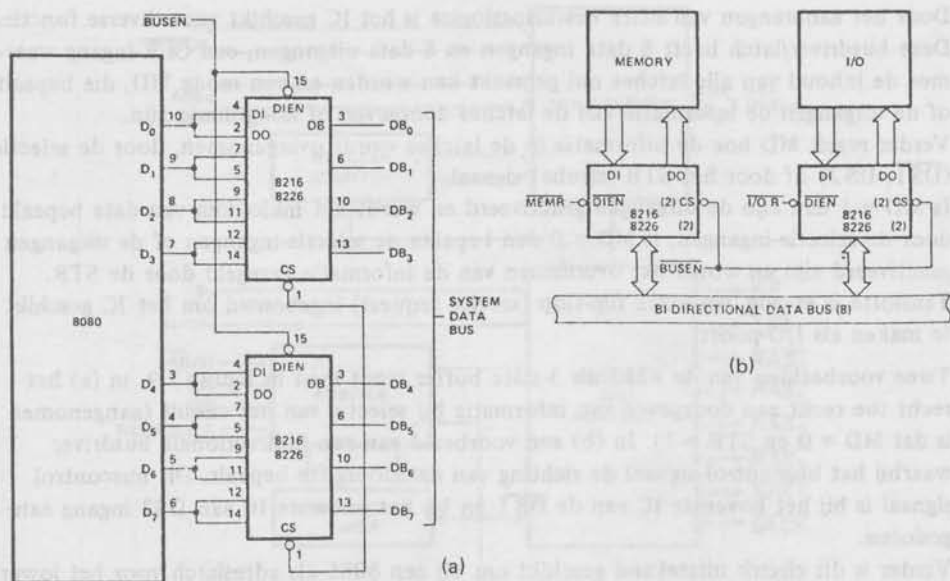


Figuur 7.6. Bidirectionele busdrivers.

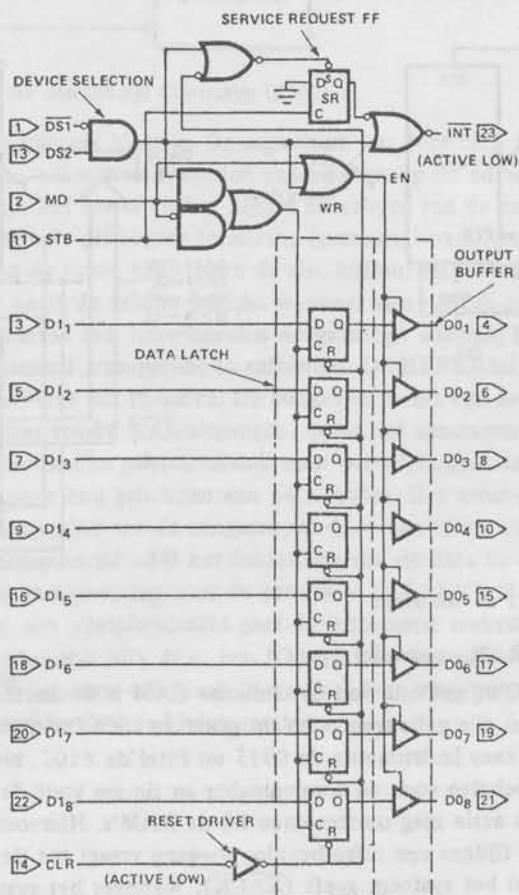
De gecombineerde in- en uitgangen aan de bidirectionele kant zijn aangegeven met DB₀–DB₃, de andere kant met data in DI₀–DI₃ en data out DO₀–DO₃. Is ook deze kant bidirectioneel dan dienen de overeenkomstige in- en uitgangen met elkaar verbonden te worden.

Een voorbeeld is in figuur 7.7a gegeven, waar aan de linkerkant van de drivers de bidirectionele databus van de 8080 is aangesloten. Figuur 7.7b toont deze busdrivers bij de koppeling van geheugen en I/O met de databus. Aan de geheugen en I/O kant is geen bidirectionele bus.

Een veelgebruikt type busdriver met geheugenfunctie is de Intel 8212, dit MSI (Medium Scale Integration) circuit bezit, naast de driver eigenschappen, ook de mogelijkheid de aangeboden informatie in latches te bewaren. Deze driver kan via three-state uitgangen 8 buslijnen aansturen, zie figuur 7.8.



Figuur 7.7. Koppeling van CPU, geheugen en I/O met de databus via bidirectionele busdrivers.



Figuur 7.8. Buffer/latch 8212.

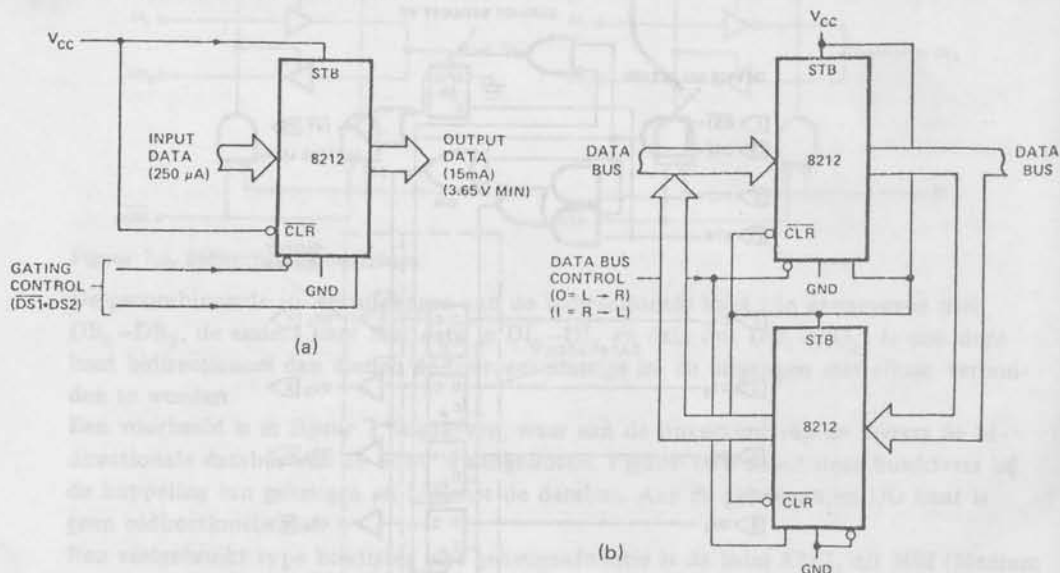
Door het aanbrengen van extra besturingslogica is het IC geschikt voor diverse functies. Deze busdriver/latch heeft 8 data ingangen en 8 data uitgangen, een CLR-ingang waarmee de inhoud van alle latches nul gemaakt kan worden en een mode MD, die bepaalt of de uitgangen de informatie van de latches doorgeven of hoogohmig zijn.

Verder regelt MD hoe de informatie in de latches wordt overgenomen, door de selectie ($\overline{DS1}$, DS2) of door het STB (strobe) signaal.

Is MD = 1 dan zijn de uitgangen geactiveerd en wordt het inklokken van data bepaald door de selectie-ingangen, is MD = 0 dan bepalen de selectie-ingangen of de uitgangen geactiveerd zijn en wordt het overnemen van de informatie geregeld door de STB. Tenslotte is er nog een extra flip-flop (service request) ingebouwd om het IC geschikt te maken als I/O-poort.

Twee voorbeelden van de 8212 als 3-state buffer vindt men in figuur 7.9, in (a) het recht toe recht aan doorgeven van informatie bij selectie van het circuit (aangenomen is dat MD = 0 en STB = 1). In (b) een voorbeeld van een bidirectionele busdriver waarbij het buscontrol signaal de richting van datadoorgifte bepaalt. Dit buscontrol signaal is bij het bovenste IC aan de $\overline{DS1}$ en bij het onderste IC aan DS2 ingang aangesloten.

Verder is dit circuit uitstekend geschikt om bij een 8085 als adreslatch voor het lower adresbyte te dienen, of bij een 8080 als statuslatch in plaats van de eerder genoemde systemcontroller/busdriver 8228.

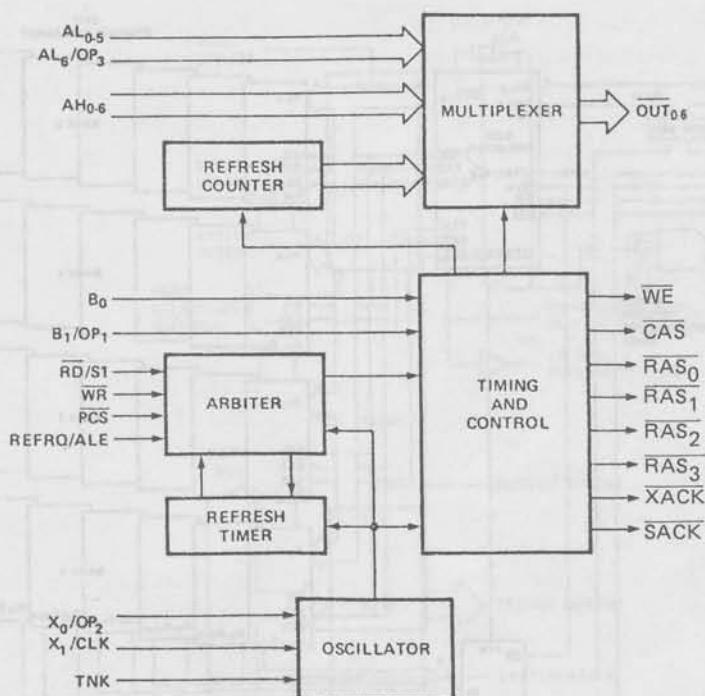


Figuur 7.9. De 8212 als busdriver.

7.2.3. Dynamic RAM controller.

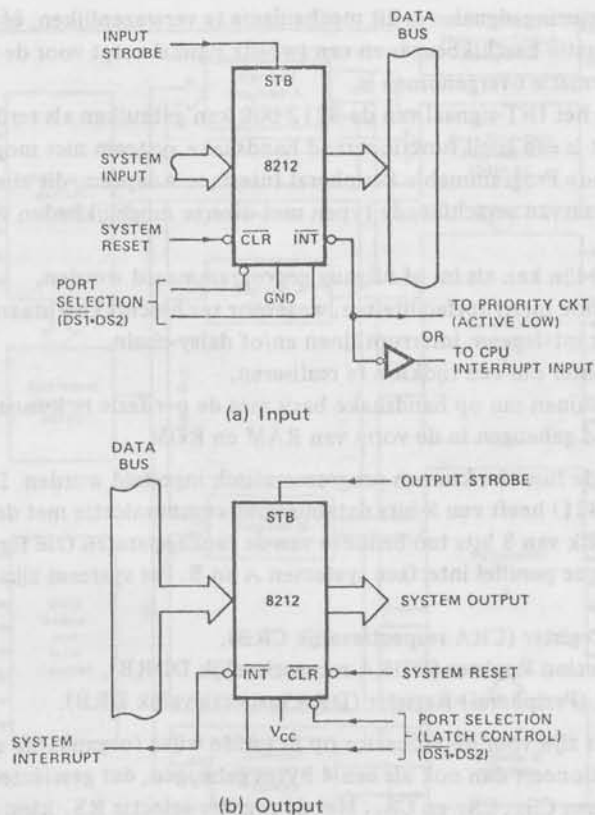
Een onmisbaar IC bij gebruik van dynamische RAM is de controller die er voor zorgt, dat de inhoud van alle geheugencellen op gezette tijden ververscht wordt. Motorola heeft de 3480, Texas Instruments de 9915 en Intel de 8202, zie figuur 7.10.

De 8202 zorgt, behalve voor de stuursignalen en timing voor de RAM's, ook voor de beslissing wie een actie mag ondernemen bij de RAM's. Hiervoor is de arbiter, die wanneer het systeem tijdens een refreshcyclus toegang vraagt tot de geheugens een melding NOT READY aan het systeem geeft ($\overline{XACK}$). Wanneer het systeem in het geheugen bezig is en de controller is aan een refreshcyclus toe dan wordt deze uitgesteld tot het



Figuur 7.10. Blokschema van een DRAM Controller (8202).

systeem met de geheugenaccess klaar is. De capaciteit van deze chip is 64 K DRAM, de adreslijnen A_{0-13} van het systeem worden aangesloten op de adres-ingangen AL_{0-6} en AH_{0-6} respectievelijk het lower en het higher adresbyte van de controller. Via de OUT_{0-6} uitgangen wordt de geheugenadressering (gemultiplexed) verzorgd, de multiplexer levert daartoe op de juiste tijdstippen de rij-, kolom- en refresh-adressen. De REFRESH TIMER geeft de arbiter bericht wanneer een refresh-cyclus gestart moet worden. Ook kan van buiten een refreshcyclus aangevraagd worden, hetgeen de mogelijkheid biedt om zogenaamd transparent te refreshen. De REFRESH COUNTER verhoogt na iedere refreshcyclus het rij-adres. De controller heeft een kolomadres strobe CAS -uitgang en 4 rij-adres strobe RAS -uitgangen. Door het aansturen van de ingangen B_0 en B_1 kan men kiezen welke geheugenbank men wil selecteren. In figuur 7.11 is een voorbeeld gegeven voor een geheugen van 64 K bytes. Het geheugen omvat 4 banks van 16 K DRAM. De controller wordt aangestuurd door een 8085 CPU. Het $XACK$ -signaal zorgt bij een leesopdracht voor het inklokken van de data in de 8212 busdriver/latch en tevens voor de terugmelding naar de processor (READY). Het RD -signaal selecteert de driver. Bij een schrijfopdracht gaat de informatie rechtstreeks naar de data-ingangen van de geheugen IC's. Is er een IO-actie dan wordt de PCS -ingang hoog waardoor de RD - en WR -ingangen afgekoppeld worden. De timing van de controller is afgeleid van de eigen kristalgestuurde klokgenerator.



Figuur 7.12. De 8212 als I/O-poort.

De processor kan data in de latches klokken met behulp van de device-selection ($\overline{DS1} \cdot DS2$) meestal afgeleid van de adresselectie en de write-opdracht. Het randapparaat kan de informatie uit de latches overnemen door het sturen van de STB (strobe) waarmee de output-buffers worden geactiveerd. De strobe veroorzaakt een interrupt INT waarop de processor kan reageren met het onderbreken van het lopende programma om nieuwe informatie in de latches te schrijven. Fungeert de schakeling als Input Port ($MD = 0$, zie figuur 7.12a), dan is de buitenwereld aangesloten aan de ingangen DI_{1-8} en de databus van de processor aan de uitgangen DO_{1-8} . De deviceselection DS wordt afgeleid van de adresselectie met de readopdracht. DS zal de outputbuffers activeren en de data komt op de databus. De schakeling is nu zo geprogrammeerd, dat op een strobe STB de informatie van het randapparaat in de latches wordt overgenomen en tevens een interrupt INT wordt gegenereerd. De interrupt kan weer worden gebruikt om het lopende programma te onderbreken, en wel om de latches uit te lezen.

In plaats van het INT-sigitaal aan de processor aan te bieden, kan men gebruik maken van een statische I/O-poort, die het INT-sigitaal als statusvlag voor de processor bereikbaar maakt.

De processor kan op gezette tijden deze en andere statusvlaggen opvragen en de gewenste acties ondernemen.

In beide voorbeelden voor de I/O-poort ontbreekt de terugmelding (van processor naar randapparaat) dat de processor actie heeft ondernomen. Dit kan alleen goed gaan, indien de verwerkingssnelheid van de processor groter is dan de snelheid van het randapparaat. Is dit niet het geval dan moet men gebruik maken van het 'handshake mechanisme'.

Met twee besturingssignalen is dit mechanisme te verwezenlijken, één signaal geeft aan dat er informatie beschikbaar is en een tweede signaal zorgt voor de terugmelding, wanneer de informatie overgenomen is.

Hoewel men het INT-sigitaal van de 8212 ook kan gebruiken als terugmelding naar het randapparaat is een goed functionerend handshake systeem niet mogelijk. Beter voldoen de zogenaamde Programmable Peripheral Interface Adapters, dit zijn universele parallel-interfaces, waarvan verschillende typen met diverse mogelijkheden verkrijgbaar zijn, zoals:

- iedere I/O-lijn kan als in- of uitgang geprogrammeerd worden,
- verschillende interruptfaciliteiten, waarvoor ter beschikking staan een controlregister met interruptvlaggen, interruptlijnen en/of daisy-chain,
- teller/counter om een tijd klok te realiseren,
- besturingslijnen om op handshake basis met de periferie te kunnen communiceren,
- ingebouwd geheugen in de vorm van RAM en ROM.

Alle genoemde functies kunnen programmatisch ingesteld worden. De PIA van MOTOROLA (M6820/M6821) heeft een 8-bits databus voor communicatie met de processor en twee I/O-kanalen elk van 8 bits ten behoeve van de randapparaten (zie figuur 7.13). De PIA bestaat uit twee parallel interface systemen A en B. Per systeem zijn er drie registers van 8 bits:

- Control Register (CRA respectievelijk CRB),
- Data Direction Register (DDRA respectievelijk DDRB),
- In/output (Peripheral) Register (DRA respectievelijk DRB).

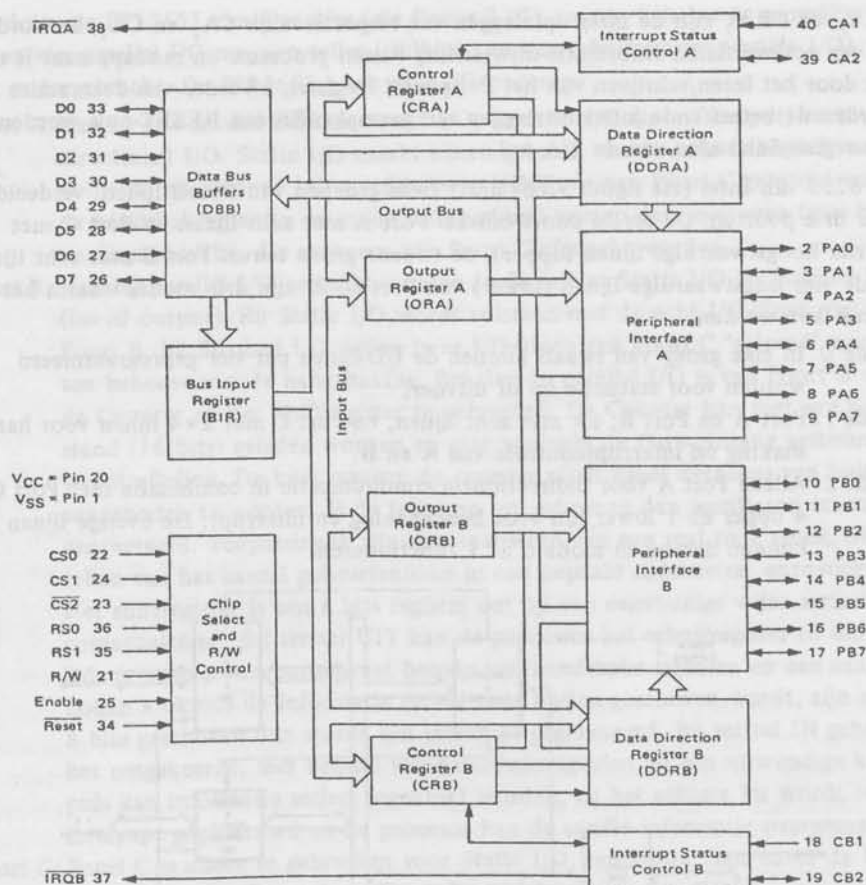
Deze registers zijn voor de processor op dezelfde wijze toegankelijk als een geheugenplaats. De PIA functioneert dan ook als een 4 bytes geheugen, dat geselecteerd kan worden met de selectielijnen CS_0 , CS_1 en $\overline{CS}_2$. Met de register-selectie RS_1 kiest men systeem A of B van de PIA, RS_0 bepaalt of toegang wordt gegeven tot het Control Register of het Data Direction Register c.q. Peripheral Register (zie tabel in figuur 7.14).

In het Data Direction Register kunnen we aangeven welke I/O-lijnen als in- of uitgang moeten functioneren.

De initiatie gebeurt door het betreffende bit van de I/O-lijnen in het DDR respectievelijk 1 of 0 te maken. De PIA heeft vier besturingslijnen CA_1 , CA_2 , CB_1 en CB_2 . CA_2 en CB_2 kunnen als in- of uitgang worden geprogrammeerd. CA_1 en CB_1 kunnen alleen als ingang worden gebruikt.

De bits van het Control Register (zie figuur 7.15) hebben de volgende betekenis:

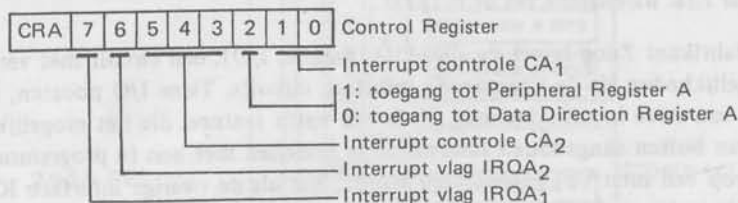
- CRA_0 – indien 1, dan wordt de interruptlijn $\overline{IRQA}$ nul op een actieve flank aan de ingang van CA_1 ;
- indien 0, dan is de lijn $\overline{IRQA}$ niet actief en zal er geen interrupt naar buiten komen.
- CRA_1 – bepaalt, welke flank aan de ingang van CA_1 als actief mag worden beschouwd, de stijgende flank indien CRA_1 1 is of de dalende flank indien CRA_1 0 is.
- CRA_2 – bepaalt tot welk register de processor toegang heeft, het Peripheral Register A, indien CRA_2 0 is en Data Direction Register A bij $CRA_2 = 1$.
- CRA_5 – indien nul, dan functioneert CA_2 als een ingang, de bits CRA_3 en CRA_4 hebben dezelfde betekenis als de bits CRA_1 en CRA_0 ;
- indien 1, dan fungeert CA_2 als een uitgang en kan deze worden gebruikt voor de terugmelding naar de periferie bij gebruik van het handshake mechanisme, door CRA_4 gelijk aan 1 te maken kan men met bit CRA_3 de toestand van CA_2 instellen.



Figuur 7.13. Blokschema van de M6820 PIA.

adressering		control register bit		geselecteerd register
RS1	RS0	CRA-2	CRB-2	
0	0	1	x	Peripheral Register A
0	0	0	x	Data Direction Register A
0	1	x	x	Control Register A
1	0	x	1	Peripheral Register B
1	0	x	0	Data Direction Register B
1	0	x	x	Control Register B

Figuur 7.14. Interne adressering van de PIA-registers.

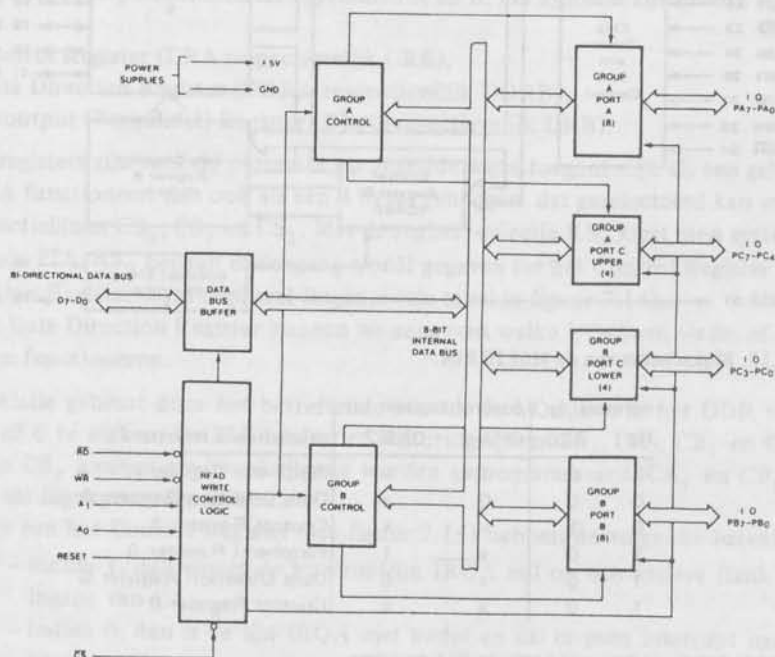


Figuur 7.15. Woorddefinitie van het Control Register.

CRA_6 en CRA_7 zijn de interruptvlaggen van respectievelijk CA_2 en CA_1 , ze worden 1 op een actieve flank. Informatie-uitwisseling tussen processor en randapparaat is mogelijk door het lezen/schrijven van het Peripheral Register, bij ieder van deze acties worden de betreffende interruptvlaggen nul gemaakt. Bij een RESET-puls worden alle registerinhouden van de PIA nul.

De 8255 van Intel (zie figuur 7.16) heeft twee groepen van twaalf lijnen, verdeeld over drie poorten. De eerste poort omvat: Port A met acht lijnen en Port C met de vier hoogstwaardige lijnen (upper), de tweede groep omvat Port B met acht lijnen en de vier laagstwaardige lijnen (lower) van Port C. Er zijn drie modes waarin het circuit kan werken:

- mode 0: In elke groep van twaalf kunnen de I/O-lijnen per vier geprogrammeerd worden voor statische in- of uitvoer.
- mode 1: Port A en Port B, elk met acht lijnen, en Port C met 2×4 lijnen voor handshaking en interruptcontrole van A en B.
- mode 2: Alleen Port A voor bidirectionele communicatie in combinatie met Port C 4 upper en 1 lower lijn voor handshaking en interrupt. De overige lijnen kunnen daarbij in mode 0 of 1 functioneren.

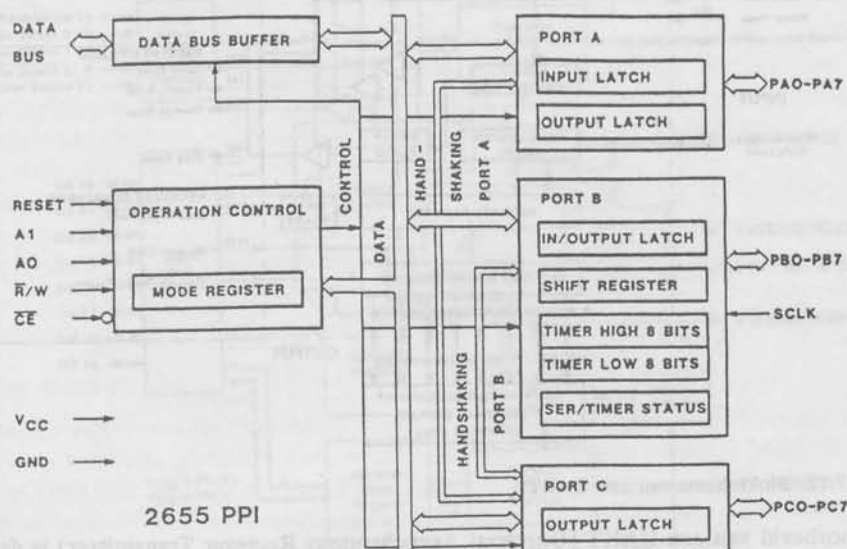


Figuur 7.16. Blokschema van de PPI8255.

De fabrikant Zilog levert de Z80-PIO (parallel I/O), een circuit met vergelijkbare mogelijkheden als de voorgaande interface circuits. Twee I/O poorten, ieder voorzien van twee handshakelijnen. Er is een extra feature, die het mogelijk maakt om de van buiten aangeboden informatie te matchen met een te programmeren masker, waarop een interrupt gegenereerd wordt. Net als de overige interface IC's van deze fabrikant is een Interrupt Daisy Chain te realiseren, het IC is daartoe voorzien van een IEI-ingang en een IEO-uitgang (resp.: Interrupt Enable Input/Output).

Tenslotte de PPI2655 van Signetics (zie figuur 7.17), waarin behalve de mogelijkheden van parallel I/O nog een teller (tijdklok) en een schuifregister (seriële I/O) zijn ondergebracht. De PPI2655 heeft drie I/O poorten:

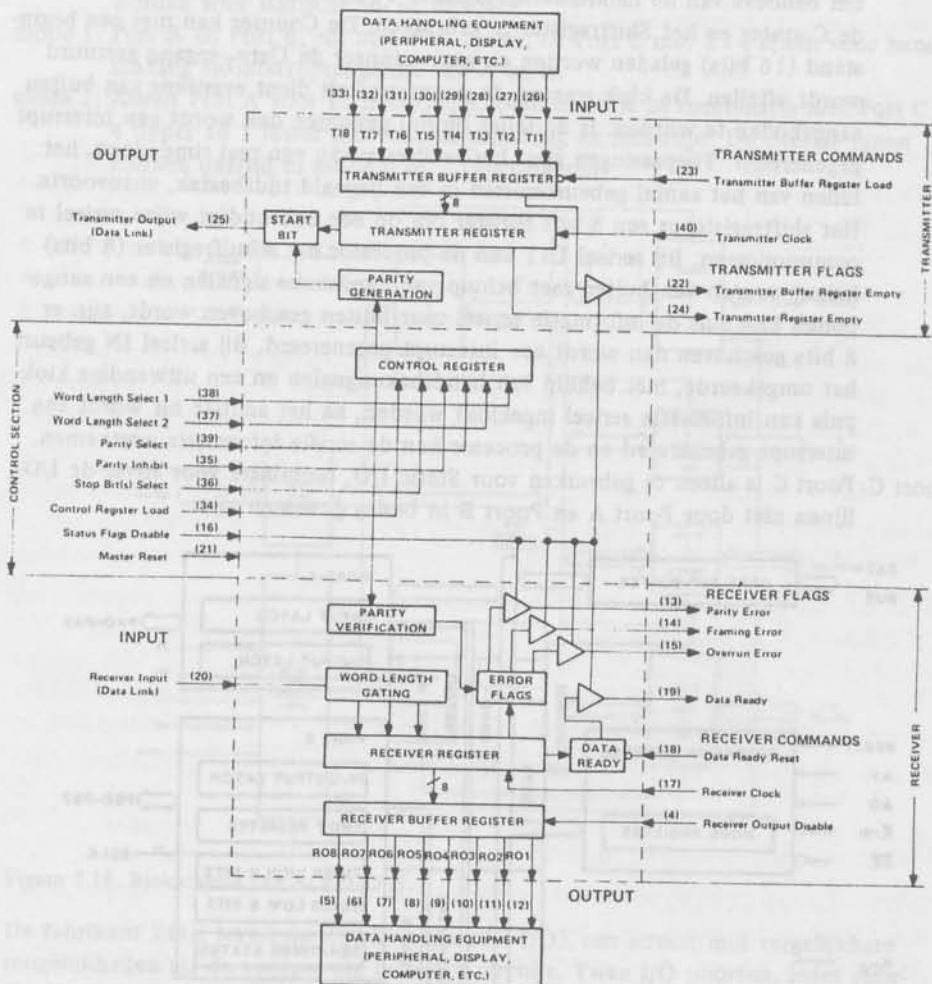
- Poort A:** Kan in drie modes werken, Static I/O, Strobed I/O (in- of output) en bi-directional I/O. Static I/O maakt alleen gebruik van de acht I/O-lijnen van Poort A, bij Strobed I/O worden twee I/O-lijnen van Poort C gebruikt voor de handshaking en in de mode Bidirectional komen daar nog eens twee besturingslijnen bij, die eveneens van Poort C afgepakt worden.
- Poort B:** Kan bij parallel I/O gebruikt worden in de modes Static I/O en Strobed I/O (in- of output). Bij Static I/O wordt volstaan met de acht I/O-lijnen van Poort B, bij Strobed I/O zullen twee I/O-lijnen van Poort C 'geleend' worden ten behoeve van de handshaking. Behalve de parallel I/O is van Poort B ook de Counter en het Shiftregister te gebruiken. De Counter kan met een beginstand (16 bits) geladen worden en gaat wanneer de Gate_T-ingang gestuurd wordt aftellen. De klok waarop de counter aftelt dient eveneens van buiten aangeboden te worden. Is de teller op nul gekomen dan wordt een interrupt gegenereerd. Toepassingen zijn: het realiseren van een real time clock, het tellen van het aantal gebeurtenissen in een bepaald tijdsbestek, enzovoorts. Het shiftregister is een 8 bits register om op een eenvoudige wijze serieel te communiceren. Bij serieel UIT kan de processor het schuifregister (8 bits) laden, waarop van buiten met behulp van handshake signalen en een aangeboden klokpuls de informatie serieel naar buiten geschoven wordt, zijn er 8 bits geschoven dan wordt een interrupt gegenereerd. Bij serieel IN gebeurt het omgekeerde, met behulp van handshake signalen en een uitwendige klokpuls kan informatie serieel ingeklokt worden, na het achtste bit wordt een interrupt gegenereerd en de procesor kan de seriële informatie overnemen.
- Poort C:** Poort C is alleen te gebruiken voor Static I/O, tenminste voor zover de I/O-lijnen niet door Poort A en Poort B in beslag genomen zijn.



Figuur 7.17. Blokschema van de PPI2655.

7.3.2. Circuits voor seriële I/O

Om serieel te kunnen communiceren zijn een aantal microprocessors, Signetics 2650 en Intel 8085, voorzien van aparte in- en uitgangen om serieel data over te brengen. Er wordt in de praktijk echter weinig van deze ingebouwde faciliteit gebruik gemaakt, daar de processor ingezet moet worden bij het datatransport en er geen standaard formaat voor asynchrone of synchrone communicatie mee wordt gerealiseerd. Verschillende fabrikanten hebben speciale interface circuits ontworpen die parallel aangeboden data omzetten naar data geschikt voor asynchrone of synchrone transmissie en terug. Deze circuits dienen vooraf programmatisch te worden voorzien van een aantal vaste gegevens, zoals bij de asynchrone communicatie het aantal data bits, het aantal stop-bits, eventueel een parity bit en de transmissiesnelheid.



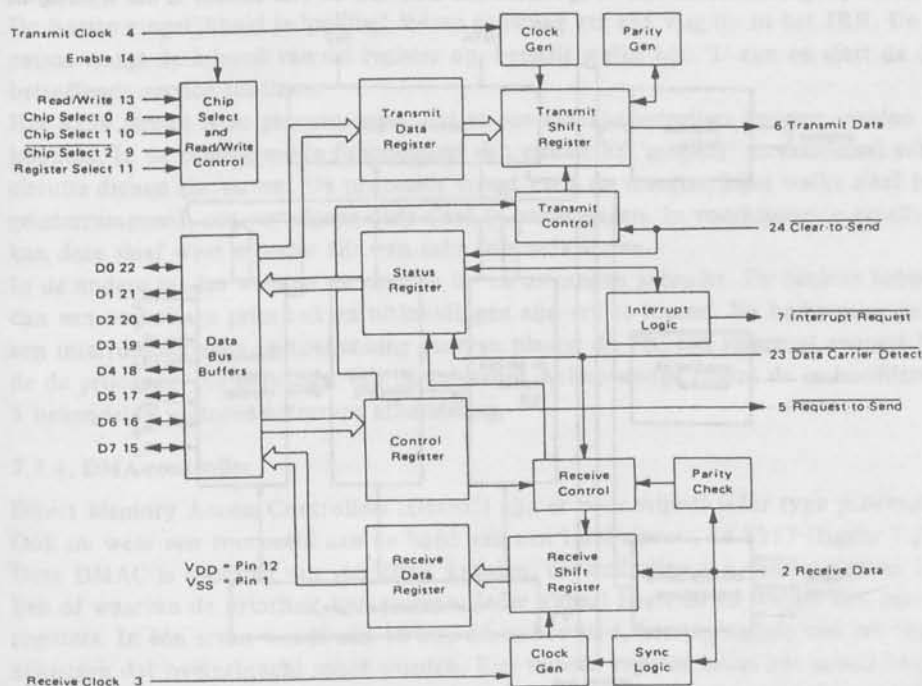
Figuur 7.18. Blokschema van een UART.

Een voorbeeld van een UART (Universal Asynchronous Receiver Transmitter) is de TMS6011 van Texas Instruments (figuur 7.18). In dit circuit zijn drie hoofdblokken te onderscheiden: een ontvang-, een zend- en een besturingsgedeelte. De besturing regelt de werking van de zender en ontvanger en zorgt voor de communicatie met de

parallele data bus. De zender zet het parallel aangeboden data-woord in een frame met startbit, parity bit, stopbits en zendt dit frame serieel uit. De zender signaleert wanneer het frame geheel verzonden is en meldt dat de zenderbuffer 'leeg' is, zodat er nieuwe informatie aangeboden kan worden. In de ontvanger wordt getest op een startbit, waarna de ontvanger de data binnenhaalt. Is er een fout in de ontvangen data (te detecteren aan de hand van de parity-check), dan zal dit gesignaleerd worden, evenals een frame-error of een overflow-error. Deze laatste ontstaat wanneer de ontvangerbuffer nog niet uitgelezen is op het moment dat de nieuwe data binnenkomt. Voor alle signaleringen worden vlaggen gebruikt die op te vragen en te testen zijn. Naast de UART's zijn er ook USART's (de 'S' staat voor synchronous), die ook de synchrone communicatie mogelijk maken. In de verzameling hulpcircuits die de fabrikanten van microprocessors leveren vinden we voor de seriële I/O bij Intel de 8251, bij Motorola de 6850, bij Signetics de 2651, bij Rockwell een SDC, bij Zilog een SIO, enzovoorts. Deze circuits hebben alle gelijke basisfuncties, die terug te vinden zijn in het blokschema van de ACIA (Asynchronous Communication Interface Adapter, Motorola 6850) in figuur 7.19. De ACIA heeft een zender en ontvanger voor asynchrone communicatie. De gebruiksfilosofie is gelijk aan die van de PIA (6820). Er zijn vier interne registers: voor status, besturing, te ontvangen en voor te zenden informatie. De ACIA is toegankelijk gelijk twee geheugenplaatsen. De vier registers zijn te adresseren met behulp van de lijn RS (Register Select) en het R/W (Read/Write) signaal.

Wat zijn de mogelijkheden van deze ACIA?

- Bij de ontvanger testen op een valse start. Ieder startelement wordt onderzocht op het aanhouden van het startelement gedurende minstens de helft van de vereiste tijdsduur. Daarna wordt de binnenkomende informatie als startelement geaccepteerd, zo niet, dan zal de start als valse start worden beschouwd, hetgeen in het status-



Figuur 7.19. Blokschema van een ACIA (6850).

register met het opzetten van een vlag wordt gesignaleerd.

- Behalve een valse start kunnen in het statusregister de volgende foutmeldingen plaatsvinden:
parity fout, hetgeen duidt op een verminkt ontvangen karakter,
overrun, deze fout treedt op wanneer de ontvanger meer karakters ontvangt dan de microprocessor kan verwerken,
framing error, hetgeen een fout in de synchronisatie betekent.
- De ACIA is te gebruiken tot een transmissiesnelheid van 0,5 Mb/s (Megabytes per seconde).
- De ACIA is bovendien geschikt voor het aansturen van een modulator-demodulator (modem). Een modem zet seriële digitale informatie om in analoge signalen die geschikt zijn voor transmissie via telefoonlijnen. Immers, een telefoonlijn is bedoeld voor spraaksignalen met frequenties tussen 300Hz en 3400Hz.

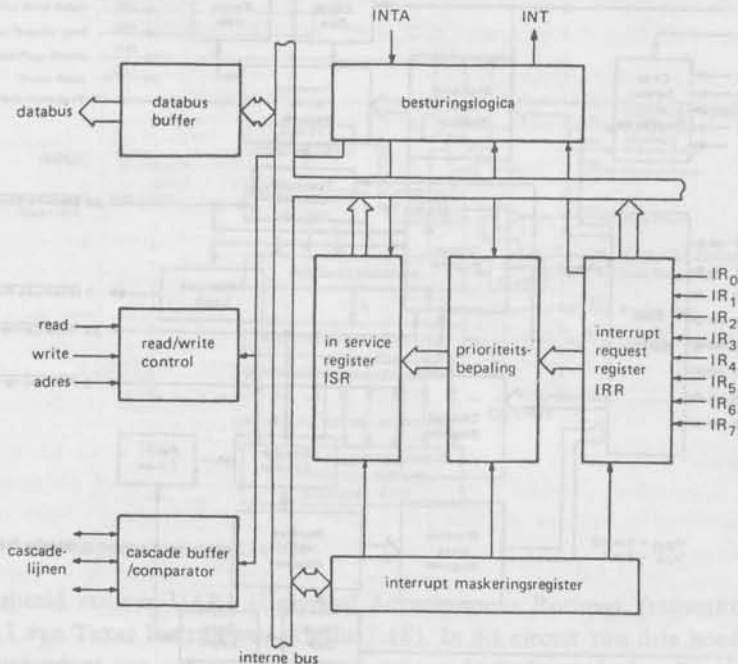
In het blokschema vindt men aan de rechterkant de signalen naar de modem/rand-apparatuur. Alle signalen voldoen aan de RS232-C standaard (zie 8.3.2).

Aan de linkerkant zijn de aansluitingen van het processorsysteem getekend en de klokkingangen voor respectievelijk de zender en de ontvanger.

7.3.3. Interrupt controller

Hoewel vrijwel iedere microprocessorfabrikant zijn eigen interrupt controller uitbrengt, zijn de functionele verschillen tussen de circuits onderling gering. Als voorbeeld van een Programmable Interrupt Controller (PIC) zal daarom de 8259 van Intel worden gebruikt.

Deze controller, waarvan het blokschema in figuur 7.20 is gegeven, heeft acht prioriteitsniveaus, dus acht interrupt ingangen. Er zijn vier modes mogelijk die met een paar instructies programmatisch zijn te specificeren. Eén van de vier modes is die waarbij de



Figuur 7.20. Blokschema van een programmeerbare interrupt controller (Intel 8259).

interrupt request ingangen IR 0–7 een onderlinge prioriteit bezitten, lopend van 0 (de hoogste) tot 7 (de laagste prioriteit). De interrupt aanvragen worden in het Interrupt Request Register (IRR) opgeslagen en in prioriteit vergeleken, waarbij de aanvraag met de hoogste prioriteit wordt doorgegeven aan het In Service Register (ISR).

Een tweede mogelijkheid is een roterende prioriteit, waarbij de prioriteit tussen de ingangen blijft bestaan, maar de aanvraag die het laatst is afgehandeld krijgt vervolgens de laagste prioriteit en alle overige ingangen schuiven een plaats in prioriteit omhoog.

	toegewezen prioriteitsniveau							
	7	6	5	4	3	2	1	0 (hoogste)
vóór eerste afhandeling	IR ₇	IR ₆	IR ₅ *	IR ₄	IR ₃	IR ₂ *	IR ₁	IR ₀
na eerste afhandeling	IR ₂	IR ₁	IR ₀	IR ₇	IR ₆	IR ₅ *	IR ₄	IR ₃
na tweede afhandeling	IR ₅	IR ₄	IR ₃	IR ₂	IR ₁	IR ₀	IR ₇	IR ₆

*) actieve interrupt request

Tabel 7.1. Prioriteitsverloop bij rotating priority.

In tabel 7.1 is het prioriteitsverloop voor drie opeenvolgende situaties aangegeven. De gedachte achter deze afhandelingsmethode is, dat voor een apparaat dat net is afgehandeld, de kans kleiner is dat het direct daarna weer interrumpeert. Door dit apparaat nu als laagste in prioriteit te plaatsen worden de andere relatief bevoorreed.

Een derde mode van werken is 'masking'. Hierbij is het mogelijk met behulp van een bitpatroon in het maskeringsregister interrupt aanvragen van een lager niveau door te laten. Onder programmabesturing wordt een '1' op de betreffende plaats in het maskeringsregister gezet, overeenkomend met het niveau van de lopende aanvraag. Dit niveau wordt dan doorbroken, zodat aanvragen met een lager niveau kunnen doorkomen.

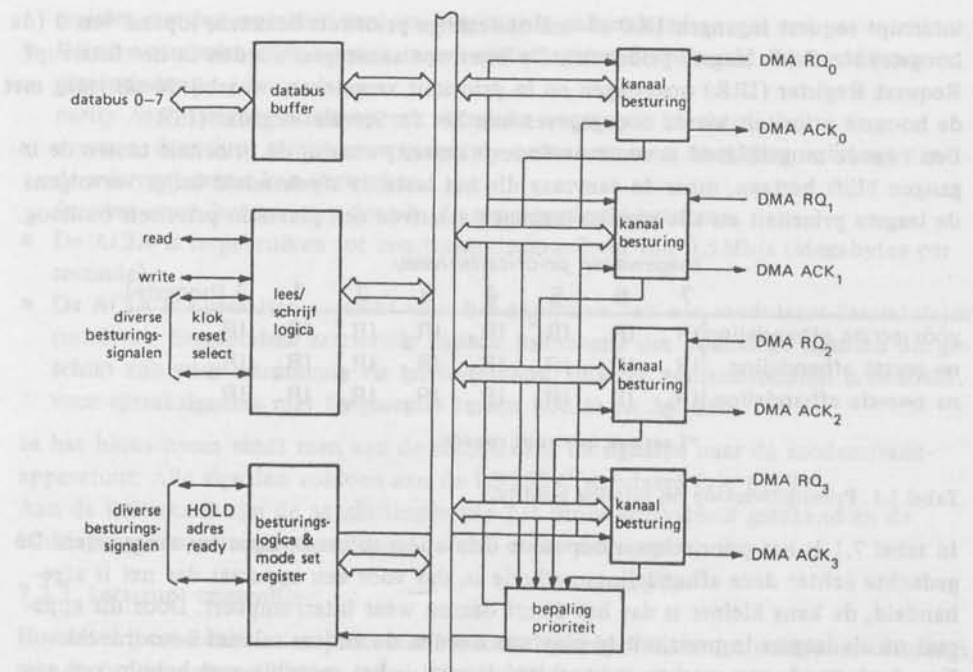
De laatste mogelijkheid is 'polling', iedere aanvraag zet een vlag op in het IRR. De processor vraagt de inhoud van dit register op, bepaalt welke bits '1' zijn en start de desbetreffende service routines.

Het 8259 circuit is zo georganiseerd, dat er een aantal controllers kunnen worden gekoppeld. In de polling mode functioneert één circuit als 'meester' en maximaal acht circuits dienen als 'slaven'. De processor vraagt eerst de meester, kijkt welke slaaf heeft geïnterrupteerd, om vervolgens deze slaaf te ondervragen. In voorkomende gevallen kan deze slaaf weer meester zijn van acht volgende slaven.

In de andere modes worden de cascade in- en uitgangen gebruikt. De circuits hebben dan een onderlinge prioriteit en uitbreidingen zijn vrij te kiezen. Na herkenning van een interruptaanvraag en toekenning daarvan plaatst de PIC een interrupt request bij de de processor. Na ontvangst van de Interrupt Acknowledge begint de in hoofdstuk 5 behandelde vectored interrupt afhandeling.

7.3.4. DMA-controller

Direct Memory Access Controllers (DMAC) zijn er voor vrijwel ieder type processor. Ook nu weer een voorbeeld aan de hand van een Intel circuit, de 8257 (figuur 7.21). Deze DMAC is voorzien van vier DMA kanalen, die onderling een vaste prioriteit hebben of waarvan de prioriteit kan roteren. Ieder kanaal heeft in de DMAC een paar registers. In één ervan wordt een 16-bits adres bewaard, het beginadres van het blok geheugen dat overgebracht moet worden. Een tweede register bevat het aantal over te brengen woorden en de richting van het transport: naar of van het geheugen. Voor ieder DMA-transport moeten de inhoud van deze registers programmatisch worden gevuld met de juiste gegevens.



Figuur 7.21. Blokschema van een DMA-controller. (Intel 8257).

Het circuit heeft vier request ingangen voor externe vragen om DMA-transport. De controller bepaalt de prioriteit en zendt een DMA-acknowledge naar de geselecteerde aanvrager. DMA-transport kan nu op twee manieren plaatsvinden: byte-voor-byte mode of in de 'burst'-mode. Bij de eerste methode wordt voor ieder byte een request geplaatst (relatief langzaam data transport); in het tweede geval wordt de request net zo lang geactiveerd tot alle te transporteren bytes zijn overgebracht. Na het verzenden van het aantal woorden dat overeenkomt met de inhoud van het 'word counter register', wordt naar processor en randapparaat signaleerd dat het transport klaar is.

8. Enkele standaards voor interfacing

8.1. Inleiding

Zodra er ergens een aantal eenheden aan elkaar wordt gekoppeld, is daarbij een vorm van communicatie nodig om de samenwerking van die eenheden ook zinvol te laten verlopen. In hoofdstuk 5 is behandeld hoe modules van een microcomputersysteem zijn te koppelen en welke vorm van communicatie daarbij wordt gehanteerd. De nadruk kwam in dat hoofdstuk te liggen op koppelingen van eenheden als een geheugen, een in-/uitvoereenheid, enzovoorts. Het type signalen dat daarbij gebruikt werd, was onder te brengen in de categorieën: data, adres of besturing. Het was dan ook de bedoeling om met een aantal kaarten een systeem op te bouwen dat in staat was de functie te verrichten die we wilden hebben. Het resultaat daarvan is, in eerste instantie, een systeem dat als 'centrale' dienst gaat doen van een groter geheel. Maar een computersysteem is pas nuttig als er ook iets, liefst zinvol, mee wordt gedaan! (Desnoods spelletjes.)

Om een zinvolle werking te verkrijgen zou het nuttig kunnen zijn het systeem te voorzien van zaken als: een toetsenbord, een display, achtergrondgeheugen, diverse perifere apparaten, een ander systeem, enzovoorts. Als we deze gaan koppelen aan het ontworpen systeem, dan zal vaak blijken dat ieder stuk apparatuur weer zijn eigen manier van aansluiten verlangt met steeds een eigen signaaldefinitie, zowel statisch als dynamisch. Kortom, een dagtaak voor een hardware-ontwerper.

Het zal iedereen duidelijk zijn, dat op dit gebied van de interface een standaardisatie nodig was (en nog steeds is). Een aantal van die standaards komen in dit hoofdstuk aan de orde. Deze interface standaards betreffen dan zowel de systeem-systeem verbindingen, als de apparaat-systeem verbindingen, met zowel parallelle als seriële data-overdracht.

Een parallel verbinding wordt in het algemeen gebruikt voor snelle, systeem-systeem communicatie, terwijl seriële verbindingen voornamelijk in gebruik zijn voor koppeling van randapparatuur met een systeem. Apparatuur als een beeldscherm, toetsenbord, printer, enzovoorts werkt meestal met een vorm van bit-seriële communicatie.

In dit hoofdstuk komen achtereenvolgens aan de orde: de beschrijving van de IEEE 488-1975 (Institute of Electrical and Electronics Engineers) als parallel bus en van de RS 232, 422 en 423 (respectievelijk: V24, V11 en V10) als seriële busverbindingen. Bij deze laatste komt ook de standaard voor karaktercodering volgens ASCII en EBCDIC ter sprake.

8.2. De IEEE 488 Bus (HP-bus)

8.2.1. Ontstaan van de bus

Vrijwel ieder elektronisch apparaat heeft wel een aansluiting om extern meetwaarden af te geven en/of besturingssignalen of andere informatie op te nemen. Grotere fabrikanten met een uitgebreid scala aan leverbare meet- en testapparaten, zochten al vroegtijdig naar een standaard voor aansluiting van al hun apparaten, zodat de klant niet werd opgezeald met steeds een andere interface-definitie voor een ander apparaat. Zo ontstond bijvoorbeeld bij FACIT de SP-1 interface, welke de betekenis en de timing van een aantal data- en besturingslijnen specificieert voor een 8-bits parallelle overdracht met asynchroon transport. Alle apparatuur van deze firma is met een SP-1 interface uitgerust en kan zonder problemen op een, eenmaal voor deze interface ontworpen schakeling, worden aangesloten.

Een andere fabrikant die deze problemen onderkende en er ook een interface voor definieerde, is Hewlett-Packard. In 1972 introduceerde deze firma de HP-IB (Hewlett-Packard Interface Bus), gebaseerd op een 8-bits datapad met daarnaast een achttal besturingslijnen. Alle apparatuur van HP wordt sinds die tijd met zo'n interface geleverd. De busdefinitie voldeed in de praktijk dermate goed, dat in 1975 de IEEE Standards Board de standaard IEEE 488-1975 uitgaf, die vrijwel geheel overeenstemde met de HP-IB specificatie. Het ziet er naar uit dat ook de IEC (International Electrotechnical Commission) deze bus tot standaard zal verheffen.

8.2.2. Organisatie van de interface

De IEEE 488 bestaat — zoals al vermeld — uit 16 lijnen. Hiervan zijn er acht bestemd voor data en acht voor besturing. De verbindingen tussen twee (of meer) apparaten wordt gevormd door een passieve kabel met een 24-polige connector die 'stapelbaar' is, zodat op een connector zelf, nog een volgende connector kan worden aangesloten. De doorverbinding tussen apparaten geeft de mogelijkheid om data te transporteren van een willekeurig apparaat naar een willekeurig ander apparaat. De transfer rate is maximaal 2MByte/s over korte afstanden en tot 500KByte/s over de maximaal toegestane 20m lengte. Grotere afstanden kunnen worden overbrugd via een extensie-eenheid. Ieder apparaat moet kunnen functioneren als zender en/of als ontvanger en/of als controller. Een zender kan data over de bus transporteren naar ieder apparaat en een ontvanger kan via de bus data ontvangen van ieder zendend apparaat.

Sommige apparaten kunnen zowel zenden als ontvangen; zo kan bijvoorbeeld een extern instelbaar meetinstrument via de bus instelgegevens ontvangen en daarna over dezelfde bus de meetwaarden uitzenden. Een controller regelt het verkeer op de bus door aan te geven welk apparaat mag zenden of ontvangen, verder kan een controller specifieke acties in een ander apparaat opgeven. Een minimum systeem bestaat uit een zender en een ontvanger, maar zonder controller. In zo'n opzet blijft het data-transport beperkt tot een directe overdracht van zender naar ontvanger, ieder van de apparaten is dan vast ingesteld op zenden, respectievelijk ontvangen.

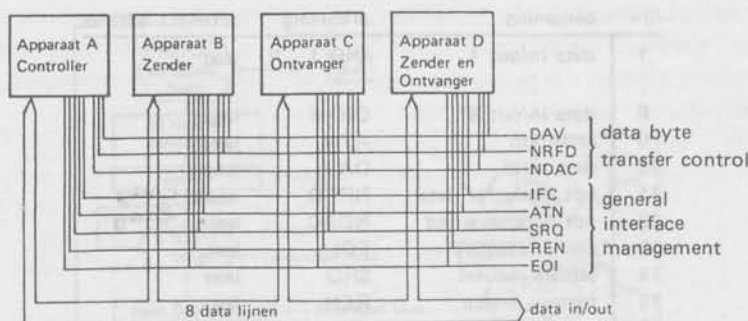
De volle verbindingsmogelijkheden van deze bus komen pas naar voren als een apparaat, dat alle drie de basisfuncties kan uitvoeren (bijvoorbeeld een computer), wordt aangesloten op de bus met daaraan een aantal andere apparaten die zenden en/of ontvangen.

8.2.3. Beschrijving van de bussignalen

De zestien bussignalen kunnen functioneel in drie groepen worden verdeeld: datalijnen, busbesturing en besturing van het datatransport. Acht van de lijnen fungeren als datalijnen, waarover een transport plaats kan vinden van één byte (8 bits) per keer. Het datatransport is bidirectioneel, dat wil zeggen dat een transport in twee richtingen mogelijk is, zowel naar een apparaat toe (instelgegevens) als van een apparaat af (meetgegevens). Een datatransport vindt asynchroon plaats, zodat een hoge mate van compatibiliteit tussen diverse apparaten haalbaar is.

De transporten tussen apparaten kunnen van 1 tot 8 bits per keer groot zijn. Alle interface boodschappen — dit zijn berichten die dienen voor het opstarten, onderhouden en afbreken van transporten — zijn in 7 bits gecodeerd, waarbij het achtste bit als pariteitbit dienst kan doen. Deze 7 bits zijn gecodeerd volgens de ASCII-code (zie hoofdstuk 8.4), een code met een zeer wijdverbreid toepassingsgebied.

In figuur 8.1 is een schematische opzet gegeven van een configuratie met vier apparaten. Vijf van deze besturingssignalen, aangeduid als *general interface management*, zorgen voor een juiste informatie-afhandeling van het bussysteem.



Figuur 8.1. Schematische opzet van een verbinding volgens de IEEE 488-1975 busdefinitie.

Eén van deze lijnen heeft als benaming *attention* (ATN). De controller regelt de functies van de overige apparaten door de ATN-lijn te activeren en daarna zend- of ontvangadressen op de datalijnen te zetten. Deze adressen zijn op ieder apparaat, hetzij extern met de hand, hetzij intern op een print, in te stellen. Als de ATN-lijn actief is, moeten alle apparaten reageren op de datalijnen. Als daarna de ATN-lijn niet meer actief is, zullen alleen die apparaten die werden geadresseerd, data zenden respectievelijk ontvangen, terwijl niet geadresseerde apparaten niet reageren.

Verschillende ontvangers kunnen tegelijkertijd actief zijn, maar per keer is er maar één zender werkzaam. In een meer complex systeem zal één van de controllers de functie van systeemcontroller krijgen. Dit is noodzakelijk om problemen van initialisatie en/of storingen te kunnen afhandelen.

Het signaal *remote enable* (REN) kan door de systeemcontroller actief gemaakt worden. Als dit gebeurt, wordt de besturing van ieder aangesloten apparaat overgenomen door de interfacebus. De apparaten reageren dan niet meer op de schakelaars en knoppen op hun frontpaneel.

Soms zal het voorkomen dat een apparaat, ingesteld als ontvanger (of een apparaat dat zelfs helemaal niet aan de communicatie meedoet), als zender wil gaan optreden. Dit kan nodig zijn als een apparaat verzamelde gegevens kwijt wil of als een foutconditie optreedt en er een alarmering moet worden gegeven. Hiertoe is het signaal *service request* (SRQ) opgenomen. Ieder apparaat kan aan de controller melden dat het service verlangt door de SRQ-lijn te activeren. De controller bepaalt dan welk apparaat service aanvraagt en welke afhandeling vereist is.

Het signaal *interface clear* (IFC) kan alleen door de systeemcontroller worden gegenereerd met als resultaat dat alle zenders en ontvangers dan niet meer geadresseerd zijn en dat alle andere controllers inactief worden. Het signaal is vergelijkbaar met een algehele reset.

De *end-or-identify* (EOI) wordt gebruikt om het eind van een datatransport aan te geven in het geval van een multiple-byte transport. Ook wordt het gebruikt bij het parallel opvragen van statusinformatie (polling) door de controller aan de overige apparaten, dit kan door zowel de ATN- als de EOI-lijn te activeren.

Binnen deze busstandaard zijn verder nog een aantal speciale acties mogelijk, zoals het adresseren van apparatuur via 2-bytes, afgesproken commandokarakters, enz. Hiervoor wordt verwezen naar de uitgebreide beschrijving van de bus door HP of naar de IEEE-standaard.

De resterende drie signalen, aangeduid als *data byte transfer control*, besturen op handshake-basis het feitelijke transport van databytes over de datalijnen; dit zijn DAV, NRFD en NDAC (zie tabel 8.1). Door middel van deze zogenaamde *3 wire*

lijn	benaming	afkorting	actieve toestand
1	data in/out 1	DIO 1	laag
:	:	:	:
8	data in/out 8	DIO 8	laag
9	attention	ATN	laag
10	data valid	DAV	laag
11	not ready for data	NRFD	hoog <i>Laag</i>
12	not data accepted	NDAC	hoog <i>Laag</i>
13	end or identify	EOI	laag
14	service request	SRQ	laag
15	remote enable	REN	laag
16	interface clear	IFC	laag

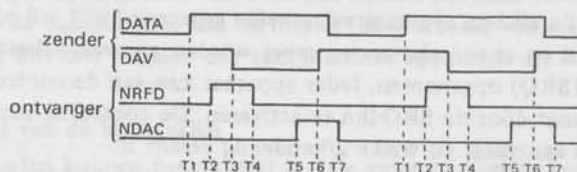
Tabel 8.1. IEEE 488-1975 bussignalen.

handshake is het transport van data, adressen, commando's, enz. over de bus relatief simpel en zeer flexibel geworden.

Door het asynchrone karakter van de bus is een grote spreiding in snelheid toegestaan, zodat apparaten van velerlei aard kunnen worden aangesloten. Ook het transport van data tussen twee apparaten met een groot verschil in mogelijke transportsnelheid is nu geregeld. Doordat de melding ~~not data accepted~~ (~~NDAC~~) als een logische EN-functie is uitgevoerd, kunnen verschillende verschillende apparaten met een onderling verschillende verwerkingssnelheid, toch tegelijkertijd informatie ontvangen omdat pas het laatste apparaat dat reageert, de NDAC-lijn weer hoog laat worden.

Om dit mogelijk te maken, is een aantal signalen als actief hoog respectievelijk als actief laag gedefinieerd. Hiervan geeft tabel 8.1 een overzicht.

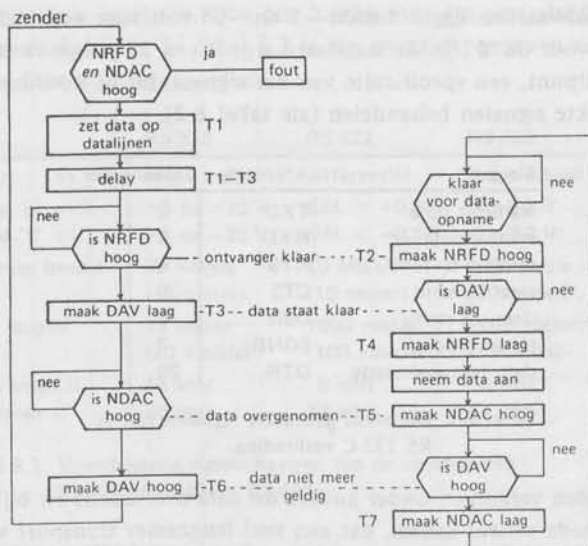
Een voorbeeld van een transport volgens handshake is gegeven in figuur 8.2. Een ontvanger geeft aan klaar te zijn voor data, door de NRFD-lijn actief te maken. Omdat hier via de logische EN-functie alle ontvangers op zijn aangesloten, wordt de lijn pas *hoog* ~~echt~~ actief als alle ontvangers klaar zijn voor data. De zender weet dan dat alle ontvangers klaar zijn en zet een databyte op de bus met daarbij DAV actief.



Figuur 8.2. Tijddiagram van de IEEE-488 handshake procedure.

Zodra een ontvanger de data ontvangen (en opgeslagen) heeft, laat deze de NDAC-lijn hoog gaan. Door de EN-werking met de andere ontvangers, zal de lijn pas hoog kunnen worden als alle ontvangers klaar zijn. NDAC blijft nu hoog totdat de zender DAV weer inactief (hoog) maakt. Daarna kunnen de toestanden weer opnieuw worden doorlopen voor een volgende overdracht.

Figuur 8.2 geeft een tijddiagram weer, corresponderend met het stroomdiagram van figuur 8.3. Hierin zijn de niveau's van de handshake-lijnen bij een datatransport aangegeven.



Figuur 8.3. Handshake procedure van de 488-bus. De aanduidingen T_n corresponderen met de T 's in het tijddiagram van figuur 8.2.

8.3. Standaards voor seriële communicatie

8.3.1. Inleiding

Bij een seriële transmissie heeft men aan twee lijnen voldoende om alle nodige signalen over te brengen tussen systemen. Om gegevens over te zenden, is dan een bit-voor-bit transmissie vereist.

De transmissie zelf kan op synchrone of asynchrone basis plaatsvinden. Onafhankelijk daarvan zal aan het uiteinde van een lijn (op een niveau hoger dan het lijnsignaal-afhandelingsprotocol), een beslissing genomen moeten worden wat op een zeker moment een bepaald bit voorstelt (nul of één), hoeveel bits samen iets voorstellen ('woorden' van 4, 5, ... of 8 bits) en wat dan precies. Dit hogere niveau komt hier niet ter sprake.

De hier behandelde standaarden gelden voor synchrone en asynchrone transmissie.

Beide methoden hebben hun voordelen. Ruwweg gezegd, passen we asynchrone transmissie toe bij een overdracht die per karakter plaatsvindt en synchrone transmissie als er per keer een blok informatie overgebracht gaat worden.

Aan de orde komen hier de standaarden RS 232, RS 422 en RS 423, en daarbij kunnen we niet om een tweetal methoden van karaktercodering heen en wel de codering volgens ASCII en EBCDIC.

8.3.2. De RS 232-C standaard

Deze standaard, uitgegeven door de EIA (Electronics Industry Association), definieert de elektrische en mechanische specificaties voor seriële transmissie van bits, oorspronkelijk bedoeld voor verbinding tussen computer of terminal en een modem. Standaardisatie was nodig op het moment dat er voor de transmissie van digitale informatie een beroep werd gedaan op de telefoonmaatschappijen. In de standaard komen ook een aantal specificaties voor ten behoeve van de aansturing van 'telefoongebonden' signalen en voor de besturing van modems. Als spanning wordt uitgegaan van ± 15 volt,

waarbij de signaalwaarden liggen tussen -5 en -25 volt voor een logische '1' en tussen $+5$ en $+25$ volt voor de '0'. In de standaard wordt een 25-polige connector gedefinieerd met, per aansluitpunt, een specificatie van het signaal. In dit hoofdstuk zullen we alleen de meest gebruikte signalen behandelen (zie tabel 8.2).

Signaal	Afkorting	Connector nr.
transmit data	TXD	2
receive data	RXD	3
request to send	RTS	4
clear to send	CTS	5
data set ready	DSR	6
signal ground	SGND	7
data terminal ready	DTR	20

Tabel 8.2. De meest gebruikte signalen van de RS 232-C verbinding.

De verdere signalen verzorgen onder andere de data-overdracht en bijbehorende besturing van een tweede serieel kanaal, dat een veel langzamer transport verzorgt dan het primaire kanaal.

De belangrijkste lijnen zijn uiteraard de data-zendlijn en de data-ontvanglijn. Deze lijnen vormen de verbinding voor het datatransport tussen twee punten.

De transmissiesnelheid kan variëren en hiervoor zijn een aantal standaard snelheden aanbevolen, te weten: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600 en 19.200 bits/sec.

Een standaard Teletype draait op 110 bits/sec, terwijl een video-terminal een veel sneller transport hanteert (vaak instelbaar van 150 tot 19.200 bits/sec).

Als een verbinding plaatsvindt via het telefoonnetwerk met behulp van modems, dan zijn de overige signalen van de RS 232 specificatie nodig om, onder andere het modem te besturen. Gaat het om een simpele directe verbinding, dan worden deze signalen doorverbonden aan de '0' of de '1', zodat alleen een verbinding van de datalijnen resteert.

De 25-polige connector, die voor de RS 232 verbindingen wordt gebruikt, is niet in de standaard beschreven, maar werd door Bell op de modems gemonteerd en is zo een 'de facto' standaard geworden. In totaal beschrijft de standaard 20 signalen, zonder de verplichting op te leggen die ook alle twintig altijd te gebruiken. Een minimale configuratie voor full-duplex verkeer kent slechts 3 signalen: *signal ground*, *transmit data* en *receive data*.

8.3.3. De RS 422 en RS 423

De RS 232-C definieert een signaaloverdracht als een *single-ended-voltage*, dat wil zeggen het al of niet aanwezig zijn van een spanning ten opzichte van aarde geeft aan of er op de lijn een '0' of een '1' staat. We kunnen echter ook werken met twee draden waartussen een verschilspanning bepaalt wat de waarde van het aangeboden bit is (dit heet dan *differential voltage*). Deze methode laat langere lijnen toe met een hogere transmissiesnelheid en maakt de verbinding ongevoeliger voor storingen.

Deze argumenten, en het feit dat steeds meer digitale systemen uitsluitend met 5 volt worden gevoed, hebben meegespeeld bij de definitie van RS 422 en RS 423 in 1977. De standaard RS 422 geeft een specificatie voor een verbinding op 'differential voltage' basis. De RS 423 is de hiermee overeenstemmende 'single-ended' versie. Helaas is de RS 232-C al zo wijdverbreid dat van toepassing van de later uitgebrachte RS 422 en

RS 423 nog maar weinig sprake is. Ook telt hierbij mee dat niet altijd de hogere transmissiesnelheid noodzakelijk is. In tabel 8.3 is een overzicht gegeven van de voornaamste verschillen tussen deze drie standaarden.

	RS 232	RS 422	RS 423
format	single ended	differential	single ended
logical '0'	+3 to +25 V	diff. > +0,3 V	> +0,3 V
logical '1'	-3 to -25 V	diff. < -0,3 V	< -0,3 V
data rate (max.)	20 Kbit/s (15 meter)	10 Mbit/s (10 meter)	300 Kbit/s (10 meter)
cable length	15 meter (20 Kbit/s)	1000 meter (100 Kbit/s)	1000 meter (1 Kbit/s)
$ V_{out} $ (max.)	15 volt	6 volt	6 volt
$ V_{in} $ (max.)	25 volt	12 volt	12 volt

Tabel 8.3. Voornaamste eigenschappen van de standaarden RS 232, RS 422 en RS 423.

8.3.4. De V24, V10 en V11 standaard

Zoals de RS 232-C een standaard is voor seriële communicatie, uitgegeven door een Amerikaans standaardbureau, zo is de V24 de Europese tegenhanger, uitgegeven door het CCITT (Comité Consultatif International de Téléphonie et Télégraphie). De V24 kent een definitie van de signalen die vergelijkbaar is met die van de RS 232-C, zelfs de benamingen zijn, waar mogelijk, gelijk gehouden. De verschillen zitten in de specificatie van de overige signalen. Zo kent de RS 232-C in totaal 20 lijnen en de V24 kent er 37.

Ook de RS 422 en RS 423 kennen een Europese tegenhanger, dit zijn respectievelijk de V11 en de V10, eveneens gespecificeerd door de CCITT. Voor een uitgebreide behandeling van de V10, V11 en V24 wordt verwezen naar *Data Transmission*, Volume VIII uit de Yellow Books van het CCITT. Hierin is tevens een beschrijving van de mechanische specificatie opgenomen.

8.4. Karaktercodering

Als we informatie overdragen, kan dit op twee manieren worden geïnterpreteerd. Ten eerste, bij een toepassing waar het gaat om de bitcombinatie op zich, zoals bij het verzenden van programmatuur voor computers in binaire vorm of van meetgegevens. Een tweede mogelijkheid is dat het transport plaatsvindt met karakters, dus leesbare informatie eventueel aangevuld met controlekarakters.

Voor deze laatste toepassing zijn er een aantal standaardcoderingen in gebruik, waarvan er hier twee worden behandeld, namelijk de ASCII (American Standard Code for Information Interchange) en de EBCDIC (Extended Binary Coded Decimal Information Code). De laatste vinden we vrijwel alleen bij IBM; de rest van de wereld gebruikt de ASCII-code.

Bij een transport van data in een willekeurig formaat, zijn er speciale karakters nodig om aan te geven waar het beginpunt van een blok data ligt en waar het eindigt, evenzo om synchronisatie te verkrijgen of om aan te geven dat het vorige karakter wordt gecorrigeerd, enz. In deze wens is voorzien door daarvoor passende coderingen op te nemen.

Tabel 8.4 geeft de codering van karakters volgens de *international reference version* zoals aanbevolen door de CCITT. Deze versie is hier opgenomen omdat ze een volle-

				b ₇	0	0	0	0	1	1	1	1	
				b ₆	0	0	1	1	0	0	1	1	
				b ₅	0	1	0	1	0	1	0	1	
					0	1	2	3	4	5	6	7	
b ₄	b ₃	b ₂	b ₁										
0	0	0	0	0	NUL	TC ₀ (DLE)	SP	0	a	P	'	p	
0	0	0	1	1	TC ₁ (SOH)	DC ₁	!	1	A	Q	a	q	
0	0	1	0	2	TC ₂ (STX)	DC ₂	"	2	B	R	b	r	
0	0	1	1	3	TC ₃ (ETX)	DC ₃	#	3	C	S	c	s	
0	1	0	0	4	TC ₄ (EOT)	DC ₄	␣	4	D	T	d	t	
0	1	0	1	5	TC ₅ (ENQ)	TC ₈ (NAX)	%	5	E	U	e	u	
0	1	1	0	6	TC ₆ (ACK)	TC ₉ (SYN)	&	6	F	V	f	v	
0	1	1	1	7	BEL	TC ₁₀ (ETB)	'	7	G	W	g	w	
1	0	0	0	8	FE ₀ (BS)	CAN	(	8	H	X	h	x	
1	0	0	1	9	FE ₁ (HT)	EM	)	9	I	Y	i	y	
1	0	1	0	10	FE ₂ (LF)	SUB	*	:	J	Z	j	z	
1	0	1	1	11	FE ₃ (VT)	ESC	+	;	K	[	k	{	
1	1	0	0	12	FE ₄ (FF)	IS ₄ (FS)	/	<	L	\	l		
1	1	0	1	13	FE ₅ (CR)	IS ₃ (GS)	-	=	M	]	m	}	
1	1	1	0	14	SO	IS ₂ (RS)	.	>	N	^	n	-	
1	1	1	1	15	SI	IS ₁ (US)	/	?	O	_	o	DEL	

Tabel 8.4. International reference version.

dige specificatie geeft van alle bitcombinaties waarbij de codering volgens ASCII is omvat. De codering van 7 bits wordt gespecificeerd; als we een achtste bit gebruiken, dan kan dit als (on)even pariteitsbit worden toegepast.

In de tabel zijn, in de linker twee kolommen, speciale karakters opgenomen, welke dienstdoen als controlekarakter bij een datatransport. TC1 t/m TC10 zijn voor *transmission control*, DC1 t/m DC8 voor *device control* en FE0 t/m FE5 zijn *format effectuators* (bijvoorbeeld: terugloop wagen, nieuwe regel, etc.). De overige zes kolommen beschrijven de 'gewone' karakters, inclusief de tekens die we ook op schrijfmachines en dergelijke tegenkomen.

Tabel 8.5 geeft de EBCDIC-codering, in gebruik bij IBM. Zoals blijkt, wijkt deze op vrijwel alle plaatsen af van de ASCII-code. Dankzij de voortgeschreden integratietechnieken is het vrij simpel om van de ene codering over te gaan naar de andere. De 8-bits EBCDIC-code wordt bijvoorbeeld als adres aangeboden aan een (P)ROM, waarbij

Bit Positions 4, 5, 6, 7

Second Hexadecimal Digit

	00				01				10				11			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	(1) NUL	(2) DLE	(3) DS	(4)	(5) SP	(6) &	(7)	(8)					(9)	(10)	(11) 0
0001	1	SOH	DC1	SOS			(13)		a	i	~		A	J	(14)	1
0010	2	STX	DC2	FS	SYN				b	k	s		B	K	S	2
0011	3	ETX	TM						c	l	t		C	L	T	3
0100	4	PF	RES	BYP	PN				d	m	u		D	M	U	4
0101	5	HT	NL	LF	RS				e	n	v		E	N	V	5
0110	6	LC	BS	ETB	UC				f	o	w		F	O	W	6
0111	7	DEL	IL	ESC	EOT				g	p	x		G	P	X	7
1000	8	GE	CAN						h	q	y		H	Q	Y	8
1001	9	RLF	EM						i	r	z		I	R	Z	9
1010	A	SMM	CC	SM		e	i	(15)								
1011	B	VT	CU1	CU2	CU3	.	\$	.	#							
1100	C	FF	IFS		DC4	<	*	%	@					J		h
1101	D	CR	IGS	ENQ	NAK	(	)	-	'							
1110	E	SO	IRS	ACK		+	:	>	=					Y		
1111	F	SI	IUS	BEL	SUB	!	~	?	"							EO

Bit Positions 0,1

Bit Positions 2,3

First Hexadecimal Digit

Control Character Representation

ACK Acknowledge
BEL Bell
BS Backspace
BYP Bypass
CAN Cancel
CC Cursor Control
CR Carriage Return
CU1 Customer Use 1
CU2 Customer Use 2
CU3 Customer Use 3
DC1 Device Control 1
DC2 Device Control 2
DC4 Device Control 4
DEL Delete
DLE Data Link Escape
DS Digit Select
EM End of Medium
ENQ Enquiry
EO Eight Ones

EOT End of Transmission
ESC Escape
ETB End of Transmission Block
ETX End of Text
FF Form Feed
FS Field Separator
GE Graphic Escape
HT Horizontal Tab
IFS Interchange File Separator
IGS Interchange Group Separator
IL Idle
IRS Interchange Record Separator
IUS Interchange Unit Separator
LC Lower Case
LF Line Feed
NAK Negative Acknowledge
NL New Line
NUL Null

PF Punch Off
PN Punch On
RES Restore
RLF Reverse Line Feed
RS Reader Stop
SI Shift In
SM Set Mode
SMM Start of Manual Message
SO Shift Out
SOH Start of Heading
SOS Start of Significance
SP Space
STX Start of Text
SUB Substitute
SYN Synchronous Idle
TM Tape Mark
UC Upper Case
VT Vertical Tab

Special Graphic Characters

c Cent Sign
., Period, Decimal Point
< Less-than Sign
(Left Parenthesis
+ Plus Sign
| Logical OR
& Ampersand
! Exclamation Mark
\$ Dollar Sign
* Asterisk
) Right Parenthesis
: Colon

¬ Logical NOT
- Minus Sign, Hyphen
/ Slash
| Vertical Line
, Comma
% Percent
_ Underscore
~ Greater-than Sign
? Question Mark
` Grave Accent
; Semicolon
Number Sign

@ At Sign
' Prime, Apostrophe
= Equal Sign
" Quotation Mark
~ Tilde
{ Opening Brace
} Closing Brace
_ Hook
_ Fork
_ Closing Brace
_ Reverse Slant
_ Chair
_ Long Vertical Mark

Tabel 8.5. Coding volgens EBCDIC.

op de geadresseerde plaats het in ASCII gecodeerde karakter staat. Hiervoor is een (P)ROM van 256×7 bits voldoende.

Ook andersom is via een (P)ROM deze omcodering te realiseren. Vrijwel alle karaktergebonden apparaten kennen de ASCII codering als standaard voor de representatie van karakters.

9. Programma-ontwikkeling

9.1. Inleiding

Zoals ondertussen bekend mag worden verondersteld, kent een computer intern alleen maar nullen en enen. Om de gebruiker respectievelijk de programmeur echter te verplichten z'n programma in deze vorm te schrijven, voert iets te ver! Daarvoor staan ons gelukkig andere hulpmiddelen ter beschikking, voornamelijk programmatuur die van fabriekswege wordt (bij)geleverd. De 'grotere' computers zijn altijd voorzien van deze systeemsoftware, kleinere systemen (met name microcomputers) worden nog wel eens 'kaal' afgeleverd. Enige jaren geleden (rond 1973/74) was het niet anders dan dat men uitsluitend hardware geleverd kreeg. Het kostte in zo'n geval weken voordat men ook maar enig idee had of het geleverde systeem zelfs maar werkte. Gelukkig is één en ander verbeterd en worden nu ook kleinere systemen geleverd waarop men, eventueel een hogere programmeertaal kan draaien.

Het minimale wat we aan programmatuur willen hebben is een programma dat de opdrachten, zoals we ons die afgekort kunnen voorstellen (bijvoorbeeld ADD voor optellen), kan vertalen in de, voor de machine hanteerbare, binaire voorstelling. Daarover gaat het eerste gedeelte van dit hoofdstuk. Verdere, nog meer luxe biedende, ondersteuning komt daarna aan bod.

9.2. Assemblerprogrammering

9.2.1. Eigenschappen van assemblers

Bij het schrijven van een programma voor een computer is het prettig als we de instructies mogen weergeven in een *mnemonische* vorm, een beknopte notatie die ons 'herinnert' aan de bedoelde operatie, bijvoorbeeld ADD voor optellen (add) en JMP voor springen (jump). Niet alleen bij de instructies is het lastig de numerieke waarde te moeten onthouden, dit geldt ook voor het adres waar bijvoorbeeld een spronginstructie naar verwijst en voor een geheugenplaats die als tussentijdse opslag in gebruik is. In deze beide laatste gevallen werken we liever met een symbolische verwijzing dan met numerieke waarden. Een adres waar we de som van een optelling bewaren en dat SUM heet, is eenvoudiger te onthouden dan adres 3651. Bij gebruik van subroutines, onthouden we beter dat we naar de subroutine 'MULTIPLY' willen verwijzen, dan naar de routine die op adres 6472 begint. Verder zou die vorm van werken tevens inhouden dat een eenmaal geschreven programma maar op één plaats in het geheugen kan worden geplaatst. Iedere andere plaats vraagt anders aanpassing van de bestaande, absolute adressen. Aan al deze bezwaren komt een assembler, een stuk systeemsoftware, tegemoet.

Een assemblerprogramma is opgebouwd uit een serie assemblerstatements, ieder statement is op te delen in vier velden:

- labelveld,
- operatieveld,
- operandveld,
- commentaarveld.

Per statement is tenminste het label-, operatie- of commentaarveld gevuld met informatie. Is in het labelveld een label aanwezig, dan wordt dit label toegekend aan een geheugenlocatie waar, bij de uitvoering van het programma, de instructie of de variabele komt te staan. Een label wordt vóór een instructie geplaatst om er vanuit een ander programmadeel naar toe te kunnen springen.

In het operatieveld vinden we een symbolische voorstelling van de operatiecode, bijvoorbeeld JMP. Het operandveld bevat, als de operatiecode dat vereist, de symbolische naam van de operand(en). Het aantal operanden wordt door het type instructie bepaald en de vorm van adressering. Zie ook paragraaf 3.3.

Commentaar is wellicht het belangrijkste veld in een assemblerprogramma. Een dergelijk programma is namelijk, doordat de gebruikte instructies op machineniveau liggen, verre van 'zelf-documenterend'. Wil het enigszins begrijpelijk zijn voor een ander dan de programmeur zelf, dan is een passende hoeveelheid commentaar onontbeerlijk. Het zou trouwens niet de eerste keer zijn, dat een programmeur zijn eigen programma's niet meer kon volgen, als hij ze een half jaar later weer onder ogen kreeg. Zinvol commentaar verhoogt de leesbaarheid en maakt aanpassingen gemakkelijker.

Het programma, geschreven in een assembleertaal, heet het *bronprogramma* (source program). Het assembleerprogramma (de assembler) vertaalt het bronprogramma in een *objectprogramma* (object program), wat gelijk is aan het bronprogramma, maar nu genoteerd in machinetaal (nullen en enen). Het assembleerprogramma kan uitgevoerd worden op een computer, waarin dezelfde processor zit als waarvoor het, te assembleren programma is bedoeld; we spreken dan van een *assembler*. De assembler kan echter ook werken op een totaal andere machine, bijvoorbeeld een assembler voor een 8085 processor die op een PDP 11 draait, in dat geval spreken we van een *cross-assembler*. Een assembler genereert tegelijk een zogenaamde listing van het geassembleerde programma. Deze listing toont naast het bronprogramma ook nog de vertaalde versie ervan, meestal in hexadecimale (of octale) vorm. Ter illustratie, het volgende voorbeeld (in 8085 assembler notatie).

In het voorbeeld staan in de eerste kolom de adressen van de instructies die erna volgen of de waarde die de symbolische variabelen krijgen toegewezen, zoals op die regel gespecificeerd. Dit zijn de echte hardware adressen. De volgende kolom geeft de vertaalde instructie weer in een hexadecimale notatie, zo is DB (= 1101.1011) de voorstelling van de 'IN'-instructie en de waarde FB is het adres van de input-poort waar het testsignaal (TSTSGL) op wordt aangeboden. Bij deze assembler wordt commentaar vooraf gegaan door een puntkomma, zoals blijkt uit de aanhef, waar een korte omschrijving van deze routine wordt gegeven en uit het commentaar na de diverse instructies.

```

; FUNCTION: TESTIN
; DESCRIPTION: TESTIN WAITS UNTIL A TEST SIGNAL
;               IS VALID, THEN INPUTS A MEASURED
;               VALUE IN REGISTER A AND RETURNS
;               TO THE CALLING ROUTINE
;
01E0                                ORG    01E0H    ;BEGINADDRESS OF PRO-
                                           ;GRAM
00FB      TSTSGL EQU    0FBH    ;"TSTSGL" = FB (HEX)
0003      TSTVAL EQU    03H    ;"TSTVAL" = 03 (HEX)
00FA      MESVAL EQU    0FAH    ;"MESVAL" = FA (HEX)

01E0      TESTIN:
01E0      DBFB      IN      TSTSGL ;INPUT OF TESTSIGNAL
01E2      E603      ANI      TSTVAL ;CHECK FOR VALUE
01E4      CAE001     JZ      TESTIN ;NOT YET - AGAIN
01E7      DBFA      IN      MESVAL ;GET MEASUREMENT VALUE
01E9      C9        RET        ;RETURN
01EA      ...

```


In sommige gevallen is de listing ook nog voorzien van opvolgende regelnummers, voorafgaand aan iedere regel in de bovengetoonde uitvoer. De eerste lijn in het voorbeeld die een betekenis heeft voor het uiteindelijke programma is de regel met:

```
01E0          ORG    01E0H    ;BEGINADDRESS OF PROGRAM.
```

Dit statement valt onder de categorie *pseudo instructies*, aanwijzingen voor de assembler, welke niet worden uitgevoerd bij executie van het programma. Hier betekent het ORG-statement dat de assembler bij de vertaling van het programma moet beginnen met de eerste instructie te plaatsen op adres 01E0, een adres dat hexadecimaal is aangegeven, zoals blijkt uit de toevoeging 'H' na het adres. Zonder deze toevoeging zou de assembler de opgegeven waarde als een decimaal getal hebben geïnterpreteerd (wat dan in het voorbeeld weer tot een foutmelding zou leiden omdat het geen decimaal getal betreft). De daarna volgende regels bevatten eveneens pseudo instructies, namelijk de instructie 'EQU', een *equate*, aangevend dat de ervoor genoemde, symbolisch aangeduide variabele, bijvoorbeeld TSTSGL, bij vertaling gelijk moet worden gemaakt aan de waarde die erna volgt, in dit geval FB (hexadecimaal). Ook hier weer de toevoeging 'H' om een hexadecimale waarde aan te geven.

Tegelijk is het bij deze assembler voorschrift om hexadecimale waarden die niet met een cijfer beginnen, te laten voorafgaan door een '0'. In dit programma worden zo achtereenvolgens aan de variabelen TSTSGL, TSTVAL en MESVAL bijbehorende waarden toegekend. Daarna vinden we pas de eerste 'echte' instructie, een INput. Deze eerste instructie komt op het opgegeven beginadres te staan, 01E0. De volgende instructie, ANI TSTVAL, de logische EN-functie van het ingelezen signaal TSTSGL met een testwaarde TSTVAL, komt op de erop volgende geheugenplaats te staan. Dat is hier adres 01E2 en niet 01E1, omdat de inputinstructie in totaal 2 bytes, dus twee geheugenplaatsen, beslaat. De volgende instructie, een conditionele sprong, test op het resultaat van de logische EN en afhankelijk daarvan wordt er teruggesprongen naar het label 'TESTIN' of doorgedaan met de volgende instructie. Deze spronginstructie is er één met een absoluut adres, reden waarom er na de instructiecode nog twee bytes nodig zijn om het totale 16-bits adres weer te geven. (De 8085 kent geen relatieve adressering) Het adres is het beginadres van deze routine, 01E0. In de vertaalde code is te zien dat de 8085 eerst het laagwaardige deel van het adres moet hebben, 'E0', en daarna pas het hoogwaardige deel, '01'.

Een assembleertaal biedt meer mogelijkheden dan de hier tot nu behandelde. Zo kunnen numerieke waarden naar keuze in binaire, octale, decimale of hexadecimale vorm worden weergegeven. Operanden hoeven niet tot een variabele beperkt te blijven, maar er mag een expressie worden gebruikt, welke door de assembler wordt uitgewerkt tot een waarde. Ook kan aan de assembler worden opgegeven om een bepaalde ruimte te reserveren in het geheugen; dit kan per byte, per 2 bytes of een willekeurig groot blok.

Bij het gebruik van een assembleertaal hebben we een directe afbeelding van instructie op objectcode. Bij langere programma's kan het de overzichtelijkheid bevorderen als we de mogelijkheid zouden hebben, bepaalde stukken die meermalen voorkomen, op een verkorte manier te specificeren. Dit laat een *macro assembler* ons toe. Deze assembler kent zogenaamde macro-instructies, instructies die we zelf kunnen definiëren. Telkens als de assembler een macro-instructie tegenkomt wordt de corresponderende code gegenereerd via expansie. In het bronprogramma kunnen we een macro-instructie bijvoorbeeld als volgt definiëren:

```

SAVE  MACRO
      PUSH PSW      ;A & FLAGS ON STACK
      PUSH B        ;B & C ON STACK
      PUSH D        ;D & E ON STACK
      PUSH H        ;H & L ON STACK
      ENDM          ;END OF MACRO

```

We hebben nu een macro-instructie gedefinieerd onder de naam 'SAVE', die alle registers redt. Als in het programma de naam 'SAVE' in het instructieveld voorkomt, dan wordt deze vervangen door de hierboven gegeven reeks instructies.

Er bestaan ook constructies waarbij parameters in de macro-aanroep kunnen worden meegenomen naar de expansie van de macro-instructie. Ook conditionele macrodefinities zijn soms toegelaten waarbij, afhankelijk van een testwaarde, *of* het ene *of* het andere deel binnen de macrodefinitie wordt geëxpandeerd.

De meeste assemblers doorlopen het bronprogramma twee keer, zogenaamde *two pass assemblers*. In de eerste *pass* wordt de waarde van alle voorkomende symbolen berekend en opgenomen in een *symbol table*. Bij deze eerste *pass* kan nog geen binaire code worden gegenereerd omdat sommige symbolen kunnen worden gebruikt voordat ze gedefinieerd zijn. Bijvoorbeeld bij een spronginstructie naar een plaats verderop in het programma komt de assembler eerst het symbolische adres, een label, tegen zonder op dat moment te weten welke waarde, adres, aan dat label moet worden toegekend. Het label komt wel in de *symbol table* te staan en op het moment dat de assembler het label tegenkomt, en dan ook de bijbehorende adreswaarde kent, wordt die waarde in de *symbol table* toegekend aan het label. Bij de tweede *pass* vindt dan de eigenlijke generatie van de binaire code plaats. Bij symbolische verwijzingen wordt dan de bijbehorende waarde uit de *symbol table* gehaald. In deze tweede *pass* wordt ook een listing afgedrukt met eventuele foutmeldingen, bijvoorbeeld fouten tegen de syntaxis, referentie naar niet gedefinieerde symbolen, niet bestaande instructiecodes, etc.

9.2.2. Codegeneratie

De assembler genereert als output een serie binaire getallen die een afbeelding zijn van het ingevoerde programma. Deze code kan *absoluut* zijn, dat wil zeggen dat de code geladen moet worden op een plaats in het geheugen zoals dat door en in het bronprogramma is opgegeven. In de toepassing moet een laadprogramma (een *absolute loader*) het objectprogramma inlezen en de waarden op de opgegeven plaatsen in het geheugen zetten. Willen we ditzelfde programma op een andere plaats in het geheugen laten werken, dan moet het bronprogramma opnieuw worden geassembleerd en daarna weer geladen op de nieuwe plaats. Wat 'nettere' assemblers genereren een relatieve (*relocatable*) code. De code wordt dan gegenereerd ten opzichte van het adres nul. Een beginwaarde van de programmateller hoeft niet in het bronprogramma te worden vermeld. De assembler produceert naast de eigenlijke code nog een lijst, de *relocation directory*, met alle argumenten die bij verplaatsing in het geheugen moeten worden gewijzigd. Het is mogelijk om in een programma te verwijzen naar symbolen in andere programma's en om symbolen van buiten het locale programma toegankelijk te maken, zogenaamde *global symbols*. Deze symbolen worden slechts in één programma gedefinieerd maar zijn vanuit andere programma's adresseerbaar. De lijst van deze symbolen heet de *global symbol directory*. Hebben we nu een relocateerbaar objectprogramma, dan is dit vergezeld van twee tabellen: de *relocation directory* en de *global symbol directory*. Een aantal van dergelijke programma's kan met elkaar worden verbonden

met een laadprogramma, de *linker*. Dit laadprogramma berekent de feitelijke waarden van de plaatsafhankelijke en globale symbolen en genereert een absoluut programma waarin alle waarden, adressen, etc. zijn ingevuld. De linker moet wel de plaats in het geheugen kennen waar het gecombineerde programma terecht moet komen. Als we nu een programma willen verplaatsen in het geheugen is een tweede assemblage niet nodig, maar kan de linker zelf een nieuwe versie genereren.

Het voordeel van een linker blijkt bij het ontwikkelen van programmatuur. Een programma kan worden opgedeeld in een aantal modules. Ieder van deze modules kan, onafhankelijk van de anderen, ontwikkeld en uitgetest worden. Door de beperkte grootte is de overzichtelijkheid een stuk beter. Achteraf combineert de linker de aparte programmamodules tot een samenwerkend geheel. Blijkt nu dat er een wijziging in het programma moet worden aangebracht, dan is wijziging van één van de modules voldoende, alleen die module behoeft opnieuw assemblage, de linker verzorgt daarna een aangepaste versie van het programma. Wel moet er op worden gelet dat de specificatie van de interface, het raakvlak van de modules, juist en volledig is, ook bij en na wijzigingen.

9.3. Compilers

Een assembleertaal en de bijbehorende assembler maken het leven van een programmeur al wat dragelijker. Ondanks dat blijven er nog een aantal moeilijkheden over. Een assembleertaal is gebonden aan een bepaald type processor zodat een eenmaal ontwikkeld programma voor processor A niet zonder meer te gebruiken is voor processor B. In dat geval is het nodig dat de programmeur een nieuwe assembleertaal leert met al de bijbehorende specialiteiten en het totale programma opnieuw gaat schrijven.

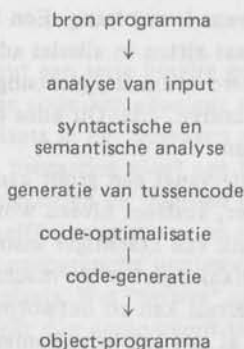
Verder is een assembleertaal sterk gericht op de structuur van de processor. De instructies zijn vrij elementair en algoritmen moeten vrij precies opgedeeld en vertaald worden in kleine, voor de processor geschikte stappen. De controle op en beheersbaarheid van het totaal aan structuur en gegevens is vrij lastig. Een belangrijk deel van de tijd en inspanning van de programmeur gaat zitten in allerlei administratief werk, zoals berekenen van datastructuren, parameteroverdracht, plaatsing van data in het geheugen, adresberekening binnen datastructuren, enz. Dit alles bevordert het maken van fouten en bemoeilijkt het opsporen ervan.

Een hogere programmeertaal ondervangt een groot aantal van deze bezwaren. Een gegeven probleem kan op een hoger, abstract niveau worden opgelost, los van boekhoudkundige rompslomp. Door gebruik van krachtiger instructies is het programma beperkter, beter overzichtelijk en onafhankelijk van de machine waar het op moet gaan draaien. Een hogere programmeertaal kan zo ontworpen zijn dat heel wat fouten bij de vertaling van het programma al aan het licht komen. De vertaling gebeurt weer door een computerprogramma, nu aangeduid met de naam *compiler*. Als de compiler op een andere processor draait dan het type waarvoor het, te compileren programma is bedoeld, dan spreken we van een *cross-compiler* in analogie met de cross-assembler. Door de aard van de compiler zal de code die wordt gegenereerd, wat minder efficiënt zijn dan die van een assembler. Afhankelijk van de soort compiler kan een toename van de code beperkt blijven tot 15 à 20%. In doorsnee zal dat eerder 50 à 100% zijn. Het zal van de toepassing afhangen of dit verlies aan efficiency (en toegenomen geheugengebruik) opweegt tegen de winst die wordt gehaald aan ontwikkeltijd en leesbaarheid van de programmatuur.

In een aantal gevallen heeft het vakgebied waarin de computer is toegepast, de structuur van de daar gebruikte programmeertaal bepaald. Als voorbeelden mogen worden genoemd:

- **BASIC**: (Beginners All-purpose Symbolic Instruction Code) een eenvoudige, makkelijk te leren taal met vrij veel rekenfaciliteiten. Oorspronkelijk op het Dartmouth College (USA) ontwikkeld voor onderwijsdoeleinden. Veel toegepast bij microcomputerssystemen. Er zijn van BASIC veel versies in omloop, die meestal niet onderling uitwisselbaar zijn.
- **FORTRAN**: (FORMula TRANslator) speciaal ontwikkeld voor numerieke toepassingen, zoals uit de naam blijkt. Krachtiger dan BASIC.
- **COBOL**: (COMmon Business Oriented Language) een programmeertaal met (Engelse) spreektaal-gerichte instructies, vooral gericht op de administratieve toepassingen. De meest gebruikte taal op dit moment.
- **ALGOL 60**: (ALGOrithmic Language) de eerste, wat de syntax betreft, formeel gedefinieerde taal. De voorloper van de 'moderne' talen met een gestructureerde controle, parameter transportmechanisme, blokstructuur, etc.
- **PASCAL**: een afstammeling van ALGOL, met een door de definitie gedwongen, nette structuur. Uitgebreide mogelijkheden voor type-definitie. Populair, ook als hogere taal voor microcomputers.
- **ADA**: een, boven PASCAL uitgaand, gedefinieerde taal met uitgebreide toepassingsmogelijkheden. Ontwikkeld als *de* programmeertaal voor iedereen. Oorspronkelijk opgezet door initiatief van het Amerikaanse Ministerie van Defensie (D.o.D.).

De invoer van een compiler is het programma geschreven in de hogere programmeertaal. De compiler levert als output een object-programma, corresponderend met de invoer. Net als bij een assembler zal het object-programma in een vorm zijn die door een linker kan worden verwerkt. De eisen die aan een compiler gesteld worden zijn vrij hoog, reden waarom deze vaak in een aantal lagen is opgebouwd, bijvoorbeeld:



In de eerste stap wordt het bronprogramma karakter voor karakter ingelezen en bekeken, statements, variabelen, getallen, enzovoorts worden omgezet in logische eenheden. De daarop volgende syntactische en semantische analyse controleert of de ingevoerde serie symbolen en tekens beantwoordt aan de structuur van de taal. De dan ontstane structuur wordt gebruikt door de tussencode-generator om een serie eenvoudige instructies per statement te genereren. Hiervoor zijn verschillende uitvoeringsvormen mogelijk. Vaak worden instructies met één functie en drie of minder operanden als tussencode gegenereerd. Deze tussencode kan worden gezien als de machinecode van een hypothetische machine (zie ook: 9.4).

Daarna vindt veelal een optimalisatie plaats van de gegenereerde code. Hierbij wordt geprobeerd het programma te verbeteren in aantal en – eventueel – soort instructies. Zo kan in een aantal gevallen een constante vooraf worden berekend, instructies met steeds dezelfde argumenten kunnen uit een lus gehaald worden en de berekening van een al bekend adres kan achterwege blijven. In de laatste stap, code-generatie, kan dan de binaire code van willekeurig welke machine worden geproduceerd. Dit betekent dat een compiler voor bijvoorbeeld 6800-code door herschrijven, respectievelijk aanpassen van de laatste stap, code-generatie, kan worden omgezet in een compiler voor 2650-code. De eenmaal geleverde grote inspanning van het opzetten van een compiler kan worden omgeslagen over meer toepassingen als er code gegenereerd dient te worden voor verschillende typen processors.

9.4. Interpretators

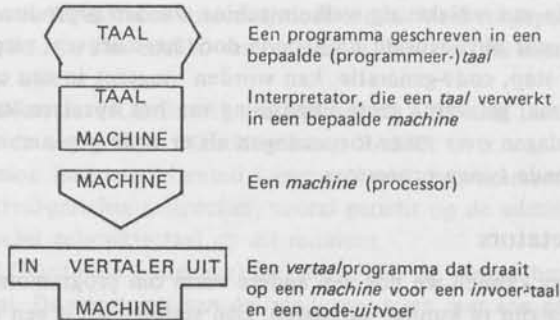
Naast compilers kennen we nog een andere vorm om programma's die in een hogere taal geschreven zijn te kunnen uitvoeren. Een statement uit een taal, bijvoorbeeld uit BASIC, kan worden 'vertolkt' door een interpretator. Die interpretator is weer een programma, nu *wel* geschreven voor de machine waarop dat BASIC programma moet draaien, dat iedere instructie apart uitvoert door via een subroutinesprong een serie machine-instructies uit te voeren die samen hetzelfde resultaat hebben als de BASIC instructie alleen. Hoewel het lijkt op een compiler, zijn er een paar levensgrote verschillen. Een interpretator moet aanwezig zijn in het geheugen van de machine waarin ook het applicatie-programma zit. Iedere keer dat een instructie wordt uitgevoerd, wordt de sprong naar de subroutine gemaakt en wordt opnieuw een heel programma-deel doorlopen. Een compiler daarentegen gebruiken we maar één keer, deze kan op een willekeurige machine draaien en een compiler genereert direct executable machine code. Vooral het verschil in executietijd tussen een gecompileerd en een geïnterpreteerd programma kan dramatische vormen aannemen.

In veel gevallen is voor PASCAL (maar dat geldt net zo goed voor andere talen) gebruik gemaakt van een compiler die, zoals vermeld in 9.3, een tussencode genereert welke min of meer standaard is en die wordt aangeduid als P-code. Omdat deze P-code te zien is als de machinecode van een hypothetische machine, was er al snel een fabrikant die een computer op de markt bracht, de *PASCAL MicroEngine* van Western Digital, die direct deze P-code executeert. Een andere ontwikkeling was het ontstaan van interpretators voor P-code. Deze leverden het voordeel op dat er éénmaal een compiler van PASCAL naar P-code nodig was, iedere gebruiker kan dan in PASCAL programmeren, en afhankelijk van de applicatie-processor, werd daarvoor dan een interpretator geleverd. Deze is namelijk relatief simpel te schrijven en vraagt niet al te veel geheugenruimte. Veelal levert het *voordeel* van het gebruik van PASCAL méér op dan het *nadeel* aan meer benodigd geheugen en langere executietijden. Door gebruik van de hogere taal kan in veel gevallen het programma kleiner zijn en levert dat weer een winst op in geheugenbezetting ten opzichte van een vergelijkbaar programma in assembleertaal geschreven.

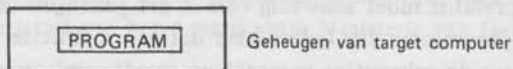
Bij personal computers wordt vrijwel altijd BASIC geleverd als hogere programmeertaal, die door een interpretator wordt verwerkt. Nieuwere ontwikkelingen tonen een groeiende belangstelling voor PASCAL, waarbij de vorm van compilatie naar P-code en daarna interpreteren favoriet is.

9.5. Schematische voorstelling van vertalingen/executie

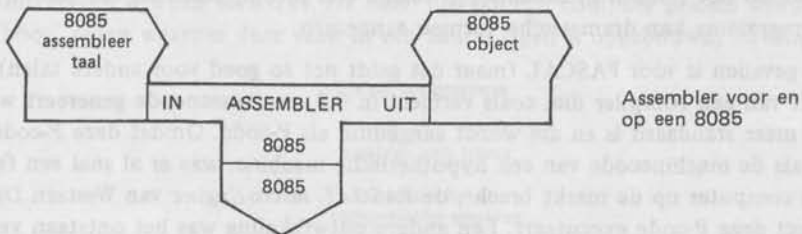
Met behulp van een aantal symbolen is het mogelijk de verschillende vormen van programma-vertaling en -executie aan te geven. We maken gebruik van de volgende vier symbolen:



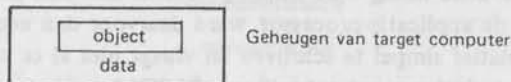
Door een aantal van deze symbolen te combineren kan de werking van (cross)assemblers en -compilers verduidelijkt worden. Naast ieder van de combinaties kan ook worden aangegeven wat er in het geheugen van de applicatie-computer (target computer) aan programmatuur komt te staan en wel op de volgende manier:



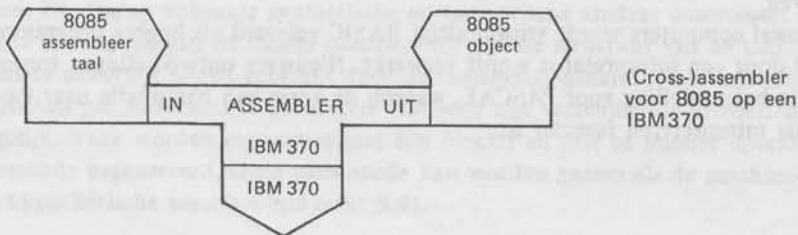
Het assembleerproces is nu als volgt voor te stellen:



In het geheugen van de 8085 (de target computer) vinden we:

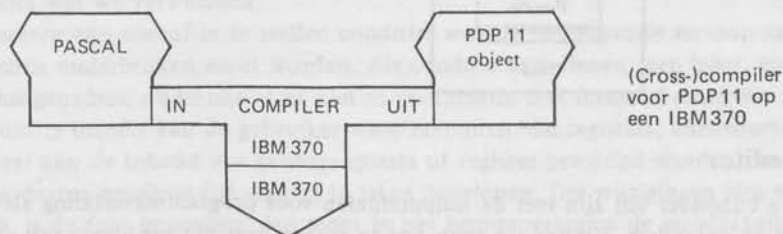


Als de vertaler op een andere machine zou draaien dan degene waar de objectcode voor bestemd is (we spreken dan over cross-assemblers) dan zou de figuur er als volgt uit zien:



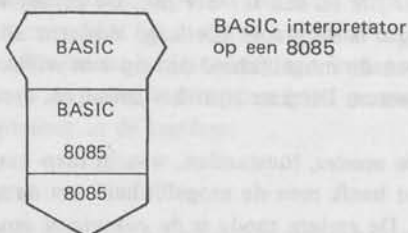
In het geheugen van de target computer vinden we dezelfde gegevens als in het 'gewone' geval van assembleren.

Hebben we te doen met een programma, geschreven in bijvoorbeeld PASCAL, dat we willen compileren, dan levert dat:

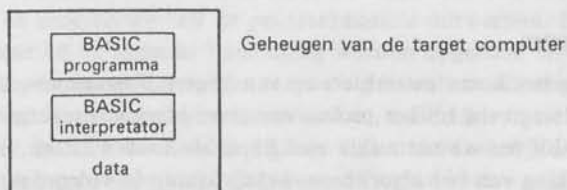


Dit schema geldt voor een crosscompiler, omdat de machine waarop de compiler draait (IBM 370), een andere is dan de machine waarvoor de code wordt gegenereerd (PDP 11). Zouden we gebruik maken van een compiler, werkend op de PDP 11 zelf, dan is er sprake van 'gewoon' compileren. In dat geval was in de bovenstaande tekening de aanduiding 'IBM 370' vervangen door 'PDP 11'. In het geheugen van de target computer is dezelfde informatie te vinden als na het assembleren, voor deze computer maakt het geen verschil waar de objectcode vandaan komt.

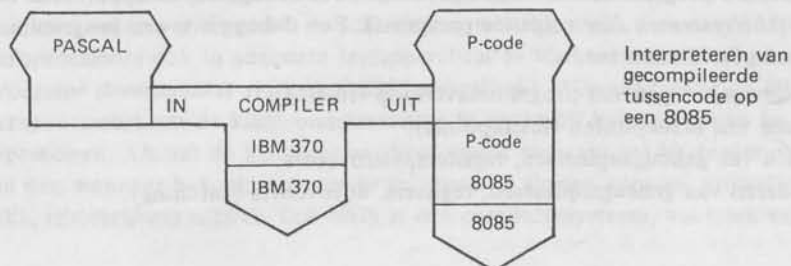
Vervolgens het proces bij interpretatie:



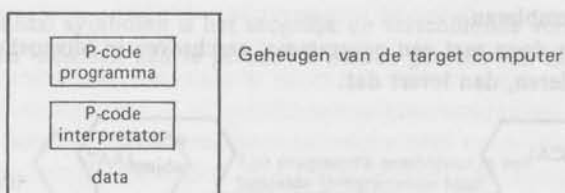
Een programma, in BASIC geschreven, wordt door een BASIC interpretator verwerkt. Het BASIC programma zelf en de interpretator bevinden zich in het geheugen van de target computer:



Tenslotte nog het geval dat we te maken hebben met een compilatie naar een tussencode, bijvoorbeeld de P-code. De situatie is dan als volgt:



De geheugenbezetting van de target computer die bij bovenstaande opzet hoort, is:



9.6. De editor

Hoewel we bijzonder blij zijn met de hulpmiddelen voor programmavertaling als assemblers en compilers, hebben we toch nog te maken met het probleem van de invoer van het bronprogramma. Bij invoer van wat voor programma dan ook, moet er een hoeveelheid 'type-werk' worden verricht. De meeste programmeurs kunnen we niet rekenen tot de klasse van de ervaren typisten/typistes. Daarom zal het vaak voorkomen dat er een toets wordt bediend, anders dan degene die was bedoeld. Ieder die wel eens een schrijfmachine heeft bediend, kent de daaropvolgende ergernis. Om bij het intypen van programma's, en de daarbijgaande correcties, de gang van zaken te vergemakkelijken is er een hulpprogramma, de *editor*. Hiermee kan tekst (dus niet alleen maar een bronprogramma) worden ingevoerd en gewijzigd. In het algemeen werkt men met een *invoer file* en een *uitvoer file*. De invoer wordt bewerkt en levert een uitvoer op. In het begin is de invoer file leeg. Moderne editors zijn afgestemd op een beeldscherm en hebben de mogelijkheid om op een willekeurige plaats op het scherm een actie uit te voeren. Dit kan zijn: het invoeren, verwijderen of veranderen van karakters.

Een editor kent vaak twee *modes*, toestanden, waarin men kan werken. Allereerst de *edit mode*, in die toestand heeft men de mogelijkheid om de tekst op het scherm naar believen te wijzigen. De andere mode is de *command mode*, waarin commando's gegeven kunnen worden, die betrekking hebben op operaties met de gehele ingevoerde file. Dit kan zijn: het wegschrijven, afdrukken, verwijderen, toevoegen of opzoeken van delen van een tekst. Verder zal er, afhankelijk van de uitvoering van de editor, nog een aanvullende hoeveelheid mogelijkheden zijn.

9.7. Debuggers

Dankzij het gebruik van assemblers en van hogere programmeertalen, worden heel wat fouten al gedetecteerd bij het proces van assemblering respectievelijk compilatie. Ondanks dat blijven we natuurlijk met bepaalde fouten zitten, hetzij fouten gemaakt bij de uitwerking van het algoritme, hetzij fouten in volgordes, overdracht van parameters, enzovoorts. Dat zijn de fouten die er pas bij de feitelijke executie van het programma uit komen. Er is hulpprogrammatuur en -apparatuur voor het opsporen van die fouten. De programmatuur wordt gevormd door *debuggers*, de apparatuur bestaat uit *ontwikkelssystemen* (zie volgende paragraaf). Een debugger is een programma dat de volgende faciliteiten biedt:

- (stapsgewijs) volgen van programmaverloop (tracing);
- plaatsen van breekpunten (breakpoints);
- uitlezen van geheugenplaatsen, registers, enzovoorts;
- veranderen van geheugenplaatsen, registers, enzovoorts (patching).

Bij het *tracen* van een programma zorgt de debugger ervoor dat het gebruikersprogramma stap voor stap wordt uitgevoerd en na elke instructie kan de waarde van de registers, van de programmacounter, van geheugenplaatsen, enzovoorts op een beeldscherm of printer worden weergegeven. Het is zo na te gaan op welk punt een programma niet meer doet wat we verwachten.

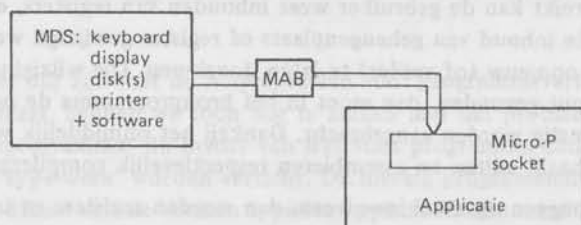
Breekpunten zijn vooraf in te stellen condities waarop het normale verloop van het programma onderbroken moet worden. Als conditie kan dienen: een label, een instructie, een geheugenadres, de laatste al of niet in combinatie met lezen of schrijven. Als zo'n breekpunt is bereikt kan de gebruiker weer inhouden van registers, enzovoorts bekijken. Eventueel kan de inhoud van geheugenplaats of register gewijzigd worden om het programma daarna opnieuw (of verder) te laten doorlopen. Die wijzigingen zijn echter tijdelijk, is de fout gevonden, dan moet in het bronprogramma de noodzakelijke wijziging overeenkomstig worden aangebracht. Dankzij het onmiddellijk wijzigen is in de testfase het herhaald editen en assembleren respectievelijk compileren overbodig.

Gebeurt het debuggen op machine-niveau, dan worden registers en instructies aangeduid met hun adres of hexadecimale waarde. Is de debugger wat netter uitgevoerd, dan zal ook symbolische adressering van registers zijn toegelaten. Is er bij het schrijven van het programma gebruik gemaakt van een hogere programmeertaal, dan wordt debuggen lastiger, omdat het directe verband tussen de instructie in het bronprogramma en de objectcode niet geheel bekend is. In dat geval (en trouwens ook bij gebruik van een assembler) is men aangewezen op testroutines. Die moet men dan in het bronprogramma opnemen op kritieke plaatsen, zodat bij executie nagegaan kan worden of de werking nog zodanig is, als men zich voorstelde. Het extra aan inspanning dat toevoeging van testroutines vraagt, wordt in de praktijk meer dan goed gemaakt door de verkorting en vereenvoudiging die optreedt in de testfase.

9.8. Ontwikkelssystemen

Alle ondersteuning die door software geleverd kan worden bij de ontwikkeling van programmatuur voor microprocessors is uitermate welkom. Een aantal mogelijkheden hiervan is genoemd in de voorgaande paragrafen. Ondanks dat, moeten we ons realiseren dat de toepassingen van microprocessors een andere categorie vormen dan toepassing van een computer of van digitale logica. Bij computers krijgen we een werkend hardware systeem geleverd en moeten we zelf de programmatuur ontwerpen. Bij digitale logica moeten het schema en de logische samenhang worden opgesteld en daarna het systeem gebouwd. Microprocessors liggen wat toepassing betreft daar tussen in. Enerzijds is software nodig om de toepassing te laten werken, anderzijds ligt de toepassing zo dicht bij de hardware dat ook kennis van digitale logica noodzakelijk is. Als een eenmaal gerealiseerde microprocessor-toepassing voor de eerste keer wordt ingeschakeld zal deze niet werken, in ieder geval niet zoals de bedoeling was. Waar moet dan de fout gezocht worden, in de logica (hardware) of in de programmatuur (software)? Als deze door verschillende personen (of groepen) zijn ontwikkeld, is het zeker dat ze naar elkaar zullen verwijzen als de schuldvraag wordt gesteld. Om voor toepassingen van microprocessors ook in adequate testapparatuur te voorzien zijn de zogenaamde *microprocessor development systems* (MDS) ontwikkeld. Allereerst door de fabrikant van microprocessors om de klant ondersteuning te geven bij het ontwerpen en uittesten van toepassingen. Als het de klant eenvoudiger wordt gemaakt zal hij eerder en meer afnemen dan wanneer het microprocessor-gebeuren er als een enorme, vrijwel onoverkomelijke, rijstebrijberg uit ziet. Een MDS is een computersysteem, voorzien van pro-

grammatuur en van hardware-aanpassingen, alles gericht op ondersteuning van de gebruiker bij ontwikkeling van zijn applicatie. De programmatuur bestaat tenminste uit een editor en een assembler (eventueel een compiler). Het grote verschil met een gewoon computersysteem zit in de debug-faciliteiten. Een MDS is voorzien van een aansluiting, een kabel met een plug, die zonder meer in de plaats van de microprocessor zelf kan worden gestoken. De totale MDS kan nu *als microprocessor* gaan optreden. Een dergelijke werkwijze wordt algemeen aangeduid met *in-circuit-emulation* of ICE. De configuratie ziet er uit als in de onderstaande figuur is aangegeven:



De aansluiting van MDS aan applicatie gebeurt via een aanpassingscircuit, hier MAB (microprocessor adaptation box) genoemd. Hierin zit onder andere een microprocessor van hetzelfde type als in de applicatie was toegepast en een hoeveelheid snelle logica. Omdat de MDS als CPU voor de applicatie dienstdoet, hebben we, als gebruiker van de MDS:

- toegang tot alle interne registers in de CPU;
- controle over alle signalen die de CPU ingaan en uitgaan;
- volledige controle over het uit te voeren programma; waar het zich bevindt en hoe het uitgevoerd moet worden;
- door de fysische verbinding met de applicatie: toegang tot de geheugenplaatsen en tot de in- en uitvoerpoorten die in de applicatie voorkomen.

De ontwikkeling van een bepaalde toepassing is nu op te delen in een aantal stappen. De eerste stap is de *programma-ontwikkeling*. Men schrijft een programma, met behulp van de editor. Deze eerste versie kan summier worden gecontroleerd door, na een assembleerfase, de MDS een simulatie van het programma te laten uitvoeren. Simulatie van een programma gaat onder (software-)besturing van een simulator, welke de executie van instructies uitvoert en in- en uitvoerinstructies simuleert door acties naar geheugenplaatsen in de MDS en/of het toetsenbord en beeldscherm. De fouten die men tegenkomt kunnen worden verwijderd (editor) en een herziene versie kan, na assemblage, worden getest. Parallel hieraan kan ook al een eerste test op het functioneren van geheugen en I/O van de applicatie worden uitgevoerd. Door de controle over de adres- en datalijnen kan de MDS waarden schrijven in het geheugen en deze teruglezen. Ook kan een waarde naar een outputpoort worden gestuurd en kan nagegaan worden of het systeem daar juist op reageert; aan inputpoorten aangeboden waarden kunnen in de MDS worden gelezen en gecontroleerd.

Een volgende stap kan zijn dat het *programma in de applicatie* wordt *ondergebracht*, via een (E)PROM of direct vanuit de MDS via de MAB in een RAM (als de applicatie daarmee is uitgerust). Onder besturing van de MDS kan het programma worden uitgevoerd, waarbij de MDS de microprocessor en eventueel ook nog de I/O emuleert. In het laatste geval kan weer worden gekozen of I/O in het geheugen van de MDS wordt gesimuleerd of dat inputwaarden van het toetsenbord komen en outputgegevens naar het beeldscherm gaan. Bij die keuze is het zelfs nog vrij om geheugen en I/O voor een deel in de MDS te situeren en een ander deel in de applicatie.

Werkt alles naar behoren, dan kan het *programma in de applicatie* met het 'echte' geheugen en de 'echte' I/O werken onder toezicht van de MDS. In deze laatste stap hebben we nog steeds de mogelijkheid om condities op te geven waarop de normale loop van het programma moet worden onderbroken (vergelijk: breekpunten bij debuggen). In de MDS kan een geheugen aan gebracht zijn, het *trace-geheugen*, waarin de signaalwaarden van de diverse lijnen worden opgeslagen. Doet zich nu een conditie voor, als gespecificeerd, dan kan het lopende programma worden onderbroken en kan via het toetsenbord de inhoud van het trace-geheugen zichtbaar worden gemaakt om een indruk te krijgen van het programmaverloop. De inhoud van het trace-geheugen kan, net als bij het statisch onderzoeken van de inhoud van het geheugen, in binaire, octale of hexadecimale vorm worden zichtbaar gemaakt. Tevens kan de inhoud worden terugvertaald naar de mnemonische voorstelling van de instructies, wat stukken simpeler voor de gebruiker is om te interpreteren.

Als de gebruiker een MDS aanschaft van dezelfde firma als waarvan hij zijn circuits betreft, dan is zo'n MDS alleen geschikt voor de ontwikkeling van dat type (of die typen) microprocessor(s). Een gebruiker die verwacht in de toekomst ook andere merken toe te gaan passen, zal tegen deze beperking de nodige bezwaren kunnen opmerken. Bepaalde fabrikanten onderkennen dit bezwaar en brachten universele MDS-en uit, ontwikkelsystemen die ondersteuning leveren voor een heel scala microprocessoren. De software is daarbij vrij simpel te leveren en voor de feitelijke aanpassing aan een bepaald type wordt de MAB in een uitgebreide vorm geleverd, met voor ieder type microprocessor een eigen MAB. Fabrikanten die een dergelijke benadering volgen, zijn onder andere: Philips, Hewlett-Packard en Tektronix.

9.9. Assembleertaal of HLL

De keuze van een programmeertaal hangt van diverse factoren af. Een aantal zal hier behandeld worden, vooral in relatie tot het gebruik van *of* een hogere programmeertaal *of* van assembleertaal. Algemeen gesproken kan men zeggen dat bij voorkeur gebruik gemaakt moet worden van een hogere programmeertaal. Men moet gegronde redenen hebben om van dat standpunt af te wijken, bijvoorbeeld: als men geen compiler ter beschikking heeft op het systeem waarop programma-ontwikkeling gedaan gaat worden, of als aan het programma bepaalde, strenge, tijdsafhankelijke eisen worden gesteld, zodat men na compilatie met geen mogelijkheid hierover een uitspraak kan doen. Ook een reden kan zijn dat er veel met I/O-gebonden signalen wordt gewerkt, wat toch weer neerkomt op I/O-instructies in combinatie met veel bit-test en bit-manipuleer instructies.

Gebruik van assembler-programmering kent de volgende nadelen:

- Het is processor-afhankelijk; een eenmaal ontwikkeld programma kan alleen maar op dat ene type microprocessor draaien. Willen we eenzelfde programma voor een ander type processor hebben, dan betekent dit het compleet herschrijven van de software.
- Het is lastig in assembleertaal een programma te schrijven. Het is nodig om in zo'n geval zowel het probleem als de machine goed te kennen, iets dat lang niet altijd opgaat.
- Men maakt gemakkelijk fouten. Alle verschrijvingen zijn in principe toegestaan en leveren later de nodige foutmeldingen op bij het assembleren. Steeds is het zaak goed na te gaan wat een instructie precies voor (bij-)werking heeft om bij het vervolg van het programma daarop te anticiperen.

- Door de niet-doorzichtige structuur van een assembleertaal is een eenmaal geschreven programma moeilijk te corrigeren en/of uit te breiden.
- Een programma is moeilijk te controleren op juiste werking. Wordt het programma opgesplitst in verschillende delen, dan zal het lastig zijn om een goede interface definitie te vinden.

In een beperkt aantal gevallen is assembler-programmering wel toegelaten en zelfs aan te bevelen, bijvoorbeeld in de al eerder genoemde omstandigheden dat veel operaties nodig zijn, dicht tegen de I/O aan. Een andere omstandigheid is de ontwikkeling van een product dat in grote hoeveelheden gefabriceerd gaat worden, bijvoorbeeld videogames. Voor zo'n toepassing kan de extra ontwikkeltijd opwegen tegen minimale productiekosten als het programma net in een handzame hoeveelheid geheugen 'geprakt' kan worden, resulterend in minder circuits.

10. Commercieel verkrijgbare microprocessors

10.1. Inleiding

Na de introductie van de 4004, welke algemeen als de eerste microprocessor wordt beschouwd, door Intel in 1971 zijn er nog talloze andere typen op de markt gebracht. Niet alleen verschenen er nieuwe typen, in de loop van enkele jaren verdwenen er ook de nodige. De oorzaak van succes of mislukking lijkt veelal te liggen in het feit of een fabrikant, bij of na de introductie van een type, goede en voldoende hardware en software ondersteuning kan leveren. Ook zeer belangrijk is het moment van introductie ten opzichte van de al verkrijgbare typen, met andere woorden: is het een 'stap voorwaarts' of levert de concurrentie al vergelijkbare producten. Wat het laatste effect betreft, heeft vooral Intel daar veel voordeel van gehad. Deze firma introduceerde als eerste de microprocessor (4004), als eerste de 8-bits versie (eerst 8008, een jaar later de 8080), en ze leverde als eerste een 16-bits processor (8086). Een firma die snel na een eerste introductie ook een vergelijkbaar aanbod heeft pakt nog wel een aardig marktaandeel, maar daarna loopt de kans van slagen hard terug. Zo hebben Motorola (6800), Rockwell (6502) en Zilog (Z-80) hun succes te danken aan een alerte reactie op de marktsituatie en blijven firma's als Signetics en Fairchild wat achter, ook door een mindere ondersteuning naar de afnemers toe.

We zien dat de situatie nu een zekere stabiliteit vertoont met een beperkt aantal microprocessor-families, die samen het overgrote deel van de microprocessor-markt voor hun rekening nemen. Deze stabiliteit is een gevolg van de enorme investeringen die afnemers hebben gedaan in de ontwikkeling van software, in de aanschaf van hardware en van ontwikkelsystemen. Dan wordt er niet meer zo snel overgeschakeld van het ene naar het andere type microprocessor. Dit heeft weer tot gevolg dat fabrikanten een grote nadruk leggen op de 'opwaartse compatibiliteit' van nieuwe typen. In dit hoofdstuk zullen een aantal microprocessors worden behandeld die een duidelijke plaats op de markt veroverd hebben. Die behandeling zal niet tot in alle details zijn, daarvoor wordt verwezen naar de handboeken van de diverse fabrikanten. Per type zal duidelijk worden gemaakt wat de kenmerkende eigenschappen zijn van de architectuur en van daaruit moet worden afgeleid wat de gevolgen zijn voor de instructieset. Bij ieder type zal ook een opsomming worden gegeven van de beschikbare soorten circuits uit dezelfde familie.

10.2. 8-Bits microprocessors

10.2.1. Algemeen

Microprocessors met een 8 bit breed datapad hebben als kenmerk dat ze algemeen bruikbaar zijn. We zien ze toegepast als processor in randapparatuur, instrumenten, tekstverwerkers, communicatiesystemen, in multi-processorsystemen, enzovoorts. Er bestaan complete computersystemen voor algemene toepassing, opgebouwd rond microprocessors en er zijn systemen voor een bepaalde toepassing, die dan uit niet meer dan twee of drie chips bestaan.

De grote vlucht die microprocessors hebben genomen is begonnen bij de 8 bits typen. Enkele bekende fabrikanten en typen zijn: Intel (8008, 8080, 8085, 8088), Motorola (6800, 6809), Rockwell en MOS Technology (6502), National Semiconductor (8060), Fairchild (F8), RCA (1802), Signetics (2650) en Zilog (Z-80).

Uiteraard is het ondoenlijk deze microprocessors allemaal te behandelen, vandaar de beperking tot een viertal. Als eerste de 8080/8085 omdat dit de basis is geweest van

de ontwikkeling op het microprocessorgebied en omdat dit type zeer vaak wordt toegepast. Dan de 6502, met name omdat we dit type zo vaak tegenkomen in personal computers. De 2650, aangezien dit de processor is met de fraaiste en meest uitgebreide adressering. Tenslotte de 6809 als exponent van de 6800-familie, die als een trendsetter te zien is voor een toekomstige ontwikkeling van 8-bits processors, die gebruik maken van de ondersteunings-IC's met 8-bits hardware en die intern werken met 16-bits data. Bij de behandeling van de typen zal blijken dat het succes voor een groot deel te danken is aan het uitbrengen van de ondersteunings-IC's en het voorzien in software.

10.2.2. De 8085

De 8085 is een verbeterde versie van de 8080, die al in 1973 werd uitgebracht. De 8085 is uitgevoerd in NMOS technologie met als veranderingen ten opzichte van de 8080: enkele +5V voeding, seriële in- en uitgang, uitgebreide interruptsignalen en een interne klok. De belangrijkste kenmerken zijn:

Technologie: NMOS, CMOS (van RCA)

Woordlengte: 8 bits

Adres: 16 bits

Instructielengte: 1 tot 3 bytes

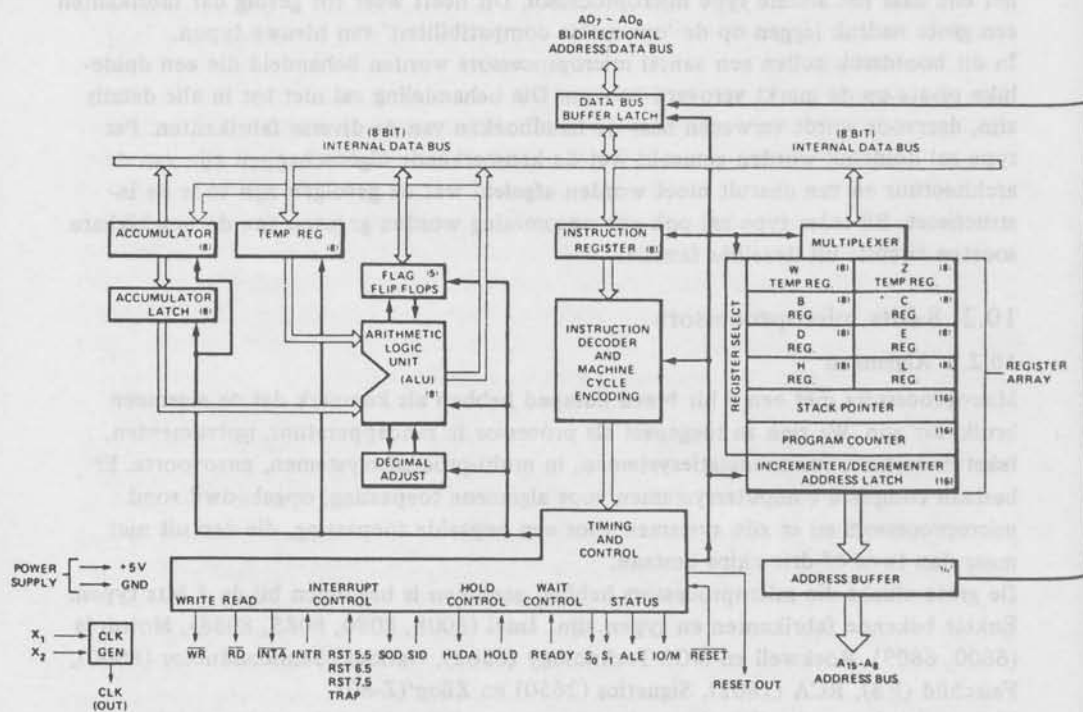
Aantal instructies: 78

Instructietijd: 4 tot 18 klokcycli

Klokfrequentie: (maximaal) 6MHz.

Second source: AMD, Siemens, Oki, RCA.

Figuur 10.1 geeft het blokschema van de 8085, zoals dat in de Intel documentatie is opgenomen. Als we deze structuur vergelijken met de algemene opzet van een CPU zoals die in hoofdstuk 1 is gegeven, dan zijn er een aantal elementen zonder meer



Figuur 10.1. Blokschema van de 8085.

vergelijkbaar. Allereerst de adres- en databuffers, respectievelijk rechts onder en midden boven in het schema; met als enig verschil dat bij de 8085 de databuffer ook een latch (onthoud-)functie heeft. In het midden zien we van boven naar beneden achtereenvolgens: 'Instruction Register', 'Instruction Decoder' en 'Timing and Control', ook delen die bekend zijn van de opbouw in hoofdstuk 1. Links in het schema zien we de ALU; dat is op zich niet nieuw, wel dat er rond het rekenorgaan naast 'Accumulator' en 'Flag Flip Flops' (= conditie flip-flops) nog een tweetal registers zit, 'Temp. Reg.' en 'Accumulator Latch'. Deze twee registers zijn toegevoegd om aan interne timing-eisen te voldoen en hebben voor de werking op zich geen significante betekenis. Onder de ALU staat in een blokje 'Decimal Adjust', een functie die programmatisch de accumulator via de ALU een speciale, decimale instelling geeft. Tenslotte rechts een blok met registers, waaronder een aantal bekende, namelijk: de 'Program Counter' en de 'Stack Pointer'. De andere registers zijn alle als 8-bits registers te beschouwen, met als extra mogelijkheid om voor bepaalde opdrachten twee naast elkaar liggende registers te koppelen tot een 16-bits register. We kennen de zes registers: B & C, D & E en H & L, die steeds per twee stuks als een dubbellengte register te gebruiken zijn. H & L hebben nog de speciale betekenis dat bij alle referenties naar een geheugenplaats door een instructie de inhoud van de combinatie HL als adreswijzer dienstdoet. Ook de accumulator is in bepaalde gevallen gecombineerd tot een 16 bits 'register', en wel met de conditie flip-flops.

De aansluitingen van de 8085 zijn te verdelen in: adres-, data- en controlesignalen. De 8085 heeft als bijzonderheid dat de datasignalen op dezelfde pennen naar buiten komen als de adreslijnen 0 t/m 7; in de tijd gezien, na elkaar. Deze manier van multiplexen houdt in dat de gebruiker, extern, een register moet aanbrengen dat de adreslijnen 0-7 opslaat omdat even later dezelfde lijnen data-informatie transporteren. Dit is één van de verschillen met de 8080, die data en adres volledig gescheiden had. Intel heeft dit gedaan omdat men de chip met dezelfde hoeveelheid aansluitpennen wilde uitrusten bij uitbreiding van de interrupt faciliteiten. De controlesignalen vertonen nu vijf verschillende interrupts:

- De INTR (interrupt request) werkt net als bij de 8080. De processor genereert een $\overline{\text{INTA}}$ als teken dat de INTR gehonoreerd is en verwacht dat extern een code voor een subroutine aanroep wordt aangeboden. De INTR kan via software worden toegelaten of geblokkeerd; blokkering vindt automatisch plaats na honorering van een INTR.
- De drie interrupts RST 5.5, RST 6.5 en RST 7.5. Deze zijn vergelijkbaar met INTR met het verschil dat bij deze drie interrupts er intern een RESTART-instructie wordt gegenereerd naar een adres in het geheugen. Ze hebben onderling een prioriteit van 7.5 voor de hoogste en 5.5 voor de laagste (het niveau van INTR is nog lager).
- De vijfde interrupt is de TRAP, dit is een niet-maskeerbare interrupt, dat wil zeggen dat ze niet te blokkeren is (ook niet via de 'Disable Interrupt' instructie). Deze heeft de hoogste prioriteit van allemaal.

De uitgangen $\overline{\text{RD}}$ en $\overline{\text{WR}}$ geven aan dat de data op de databus gelezen respectievelijk geschreven moet worden. De READY ingang dient voor synchronisatie van datatransport, zolang READY laag is blijft de CPU wachten met de lees- of schrijfcyclus. De HOLD input geeft aan dat een andere processor controle over de adres- en databus wil hebben. Zodra de processor de lopende machine-cyclus heeft afgemaakt, worden de adres- en datalijnen met de bijbehorende besturingssignalen vrijgegeven en kan een ander ze besturen. Als teken voor deze toestand wordt het signaal HLDA (HLD Acknowledge) uit-

gezonden, zodat ook extern te zien is dat de processor in de 'hold' toestand is. De signalen SID (serial input data) en SOD (serial output data) zijn voor seriële data in en uit. Met de instructie SIM en RIM zijn de waarden van deze lijnen in te stellen respectievelijk in te lezen via bit 7 van de accumulator.

Het signaal $\text{IO}/\overline{\text{M}}$ geeft aan of de data op de databus met input/output te maken heeft of met het geheugen. ALE (address latch enable) is de puls waarmee de externe adreslatch bestuurd kan worden. Als deze puls verschijnt, staan op de lijnen AD0-AD7 de adressen 0-7, welke we extern moeten opslaan. Het signaal $\overline{\text{RESET}}$ wordt gebruikt bij inschakelen of bij een algehele reset van het systeem. Het signaal heeft tot gevolg dat de PC, de Interrupt Enable en de HLDA-flip-flop op '0' komen te staan. Synchroon met de processorklok wordt een signaal $\overline{\text{RESET OUT}}$ afgegeven dat weer extern bruikbaar is voor de reset van de rest van het systeem. Tussen X_1 en X_2 kan een kristal of een RC-netwerk worden aangesloten waarvan intern de klokfrequentie wordt afgeleid. Eventueel kan X_1 ook als klokingang dienen, in het geval er een extern klok-sig-naal in gebruik is.

Tenslotte de signalen S_0 en S_1 . Deze signalen geven informatie over de status van de processor en wel als volgt:

	S_1	S_0
HALT	0	0
WRITE	0	1
READ	1	0
FETCH	1	1

Tabel 10.1 geeft een overzicht van de soort instructies die de 8085 kent.

1. Data manipulatie

- Rekenkundig en logisch.
- BCD rekenen.
- Dubbellengte operaties, waarbij een instructie twee data bytes als één 16-bit woord behandelt.

2. Data verplaats

De 8085 kan drie paar 'general purpose registers' gebruiken als pointers naar een plaats binnen het 64K geheugenbereik. Hiermee is meervoudige indexering te plegen, hoewel dat meer stappen vraagt dan de 'standaard' index-methode. Er zijn twee extra instructies voor seriële communicatie.

3. Programma manipulatie

De stackpointer wordt gebruikt om een 'last in - first out' stapel te creëren in het werk-geheugen. Alle registers kunnen worden geincrementeerd en gedecrementeerd. Er is voorzien in een meervoudige interrupt. De bus laat extern toegevoegde DMA-afhandeling toe.

4. Programma status

Er zijn instructies die toegang geven tot het statusregister.

Tabel 10.1. Overzicht 8085-instructies.

De 8085 familie is een zeer uitgebreide. Alle bestaande support circuits van de 8080 zijn bruikbaar bij de 8085 en verder heeft Intel voor de 8085 ook nog een eigen serie circuits uitgebracht. Dat was nodig omdat de 8085 met een gemultiplexte data- en adresbus werkt. Een 8085-systeem moet zodoende altijd voorzien worden van logica om de laagste adreslijnen uit te coderen en te onthouden. Om aan dit bezwaar tege-moet te komen heeft Intel daartoe een serie combinatie-circuits ontwikkeld. Deze circuits bevatten een combinatie van geheugen en I/O-faciliteiten en kenmerken zich verder door het feit dat ze intern de demultiplexing verzorgen van adres- en datalijnen. Als voorbeelden noemen we hier:

8155, een circuit met $\frac{1}{4}\text{K} \times 8$ bit RAM, twee programmeerbare 8-bits poorten, één programmeerbare 6-bits poort en één programmeerbare 14-bits teller.

8355, bevattende: $2K \times 8$ bit ROM met twee programmeerbare 8-bits poorten.

8755, met $2K \times 8$ bit EPROM en twee programmeerbare 8-bits poorten.

Met deze circuits is het mogelijk om een systeem te bouwen rond een 8085 met slechts een twee- of drietal chips, wat toch voldoende biedt aan geheugen en vooral aan I/O-lijnen om de toepassing te realiseren.

Zoals al vermeld zijn ook de andere circuits toepasbaar uit de 80 reeks, waaronder seriële en parallelle interface circuits, timers, DMA controller, interrupt controller, floppy disk controller, CRT en keyboard controller, floating point processing unit, bubble memory controller, enzovoorts; zie ook hoofdstuk 7.

10.2.3. De 6502

De 6502 is er één uit een serie microprocessors die allen met de aanduiding 65## door het leven gaan. De meeste typenummers horen bij processors die gekort zijn in hun mogelijkheden, bijvoorbeeld door minder adreslijnen naar buiten te voeren. De bedoeling is steeds om een chip te leveren met minder aansluitpennen, behalve de 6502 en de 6512 hebben de anderen namelijk 28 pennen. De 6502 en 6512 hebben ieder 40 pennen én de volledige 16-bits adresbus. Het belangrijkste verschil tussen deze twee is dat de 6502 een klok op de chip zelf heeft en de 6512 niet. De 6502 kan worden gezien als een verbeterde 6800. Wat de architectuur betreft zijn er zeer veel overeenkomsten, overigens zonder dat ze qua instructieset of busstructuur compatibel zijn. De 65 serie is van de Commodore Semiconductor Group (voorheen: MOS Technology Inc.).

De bekendheid van de 6502 is vooral te danken aan de successen van de personal computers die van dit type gebruik maken, met name de Apple en de PET (CBM). Het doel van de ontwerpers was om zoveel mogelijk PDP11-achtige adresseringsfaciliteiten te verkrijgen.

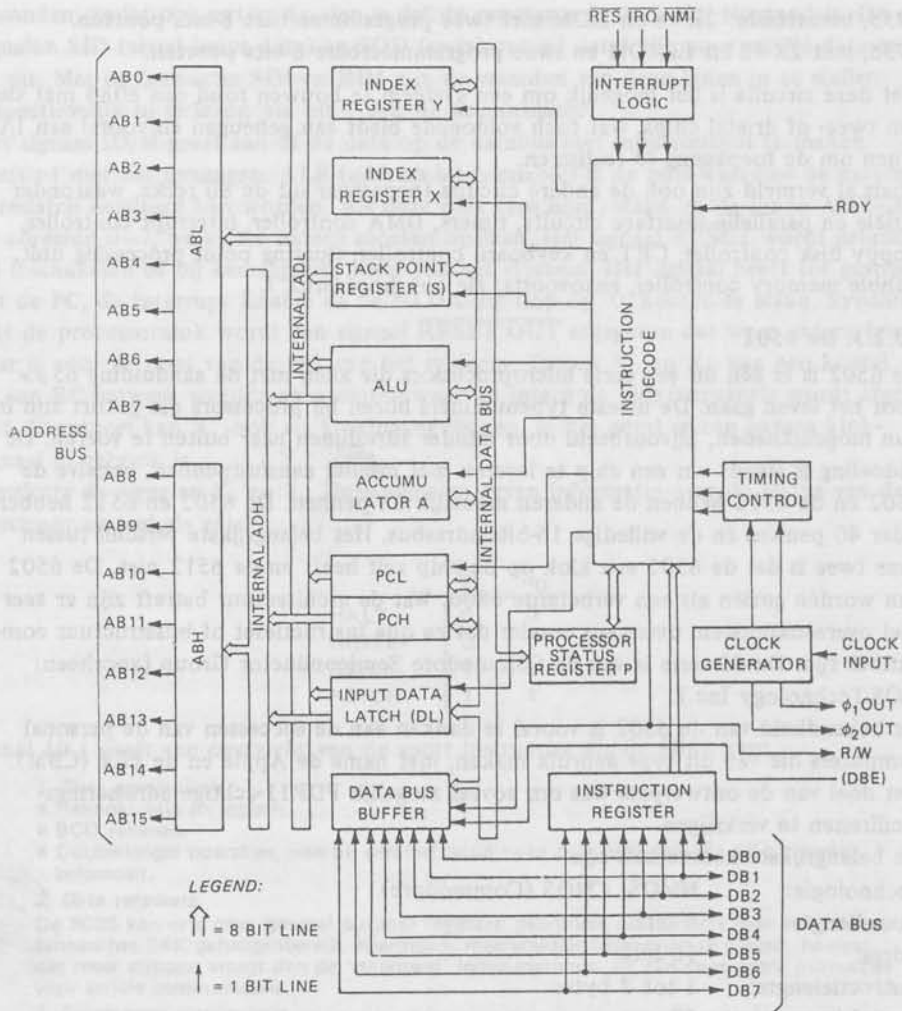
De belangrijkste kenmerken zijn:

Technologie:	NMOS, CMOS (Commodore)
Woordlengte:	8 bits
Adres:	16 bits
Instructielengte:	1 tot 3 bytes
Aantal instructies:	57
Instructietijd:	2 tot 7 klokcycli
Klokfrequentie:	(maximaal) 2 MHz
Second source:	Rockwell, Synertek.

Figuur 10.2 geeft het blokschema van de 6502 microprocessor. In dit blokschema valt ons allereerst op dat hierin centraal de 'instruction decode' staat aangegeven. Deze vormt, samen met de erboven aangegeven 'interrupt logic', (rechts) 'timing control' en 'clock generator' het interne besturingsgedeelte. Vandaar de verbindingen naar de links gelegen registers. Dit geheel komt overeen met respectievelijk de instructie decoder en de timing & control uit het blokschema van figuur 1.2.

Onderaan de figuur zien we de databus, met daaraan verbonden zowel een databus buffer als het instructieregister. Bij deze CPU kan een instructie dus direct naar het IR, zonder via de databus buffer te lopen. Links zien we de adresbus, gesplitst in een hoog (ABH) en een laag (ABL) gedeelte.

Opvallend is, dat ook alle interne registers zijn weergegeven in een 8-bits vorm. We zien achtereenvolgens, van boven naar beneden: een indexregister Y, een indexregister X, de stack pointer S (8-bits!), de ALU, de accumulator A, de programmateller (gesplitst in PCL en PCH), een data latch (DL) en de databus buffer.



Figuur 10.2. Blokschema van de 6502.

Als laatste zien we nog een processor status register P, gekoppeld aan de interne bus en aan de 'instruction decode'. In dit register vinden we onder andere de conditie flip-flops. De 6502 kent een BREAK instructie, te vergelijken met een software interrupt, waarvan het optreden ook in register P wordt opgenomen om onderscheid tussen hard- en software interrupts te kunnen maken. De conditie flip-flops die voorkomen, zijn: sign, zero, carry en parity.

Omdat de stack pointer maar 8 bits kent en stack operaties op het geheugen plaatsvinden, moet er één of andere afspraak bestaan over de waarde van de acht hoogstwaardige bits. In de 6502 is gekozen voor de waarde '1', dat wil zeggen dat bij stack operaties het adres aan de bovenkant wordt aangevuld met 0000 0001 en dat dus alle adressen liggen tussen 100 en 1FF (hexadecimaal).

De aansluitingen van de 6502 zijn te verdelen in drie groepen: adres-, data- en controle-signalen. De adressignalen, de adresbus, bestaat uit 16 lijnen, AB0 t/m AB15. De databus is 8 bits breed, bidirectioneel en bestaat uit de lijnen DB0 t/m DB7. Over de databus vindt transport plaats van instructies en gegevens. De besturingssignalen moeten

apart vermeld en behandeld worden, als eerste de interrupts. De 6502 kent twee interrupts: de $\overline{\text{IRQ}}$ en de $\overline{\text{NMI}}$. De $\overline{\text{NMI}}$ genereert altijd een interrupt, deze is non-maskable; de $\overline{\text{IRQ}}$ kan programmatisch worden geblokkeerd met een 'disable interrupt' instructie. De $\overline{\text{RES}}$, reset, initialiseert de processor vanaf 'niets', bijvoorbeeld bij inschakelen van de voedingsspanning. De PC wordt geladen met de inhoud van geheugenplaatsen FFFC (PCL) en FFFD (PCH), beide plaatsen hexadecimaal aangeduid.

De RDY input verzorgt de synchronisatie tussen processor en geheugen bij een leesoperatie, zolang RDY laag is wordt executie van de lopende actie uitgesteld. Bij schrijven heeft de RDY geen effect.

De kloksignalen ϕ_1 en ϕ_2 zijn extern te gebruiken en leveren een synchrone werking met de processor op. Op CLOCK INPUT is extern een kristal of een RC-netwerk aan te sluiten om de klok op te wekken.

De lijn R/W indiceert of de data op de databus de processor in moet (R) of juist er uit komt (W).

Het signaal SYNC levert een aanduiding dat de processor bezig is met een 'fetch' cyclus en onderscheidt zodoende deze cyclus van andere leescycli.

Tenslotte de aansluiting S/O, deze ingang zet het overflow bit op '1' als de signaalwaarde verandert van '1' naar '0'. Deze ingang is bedoeld voor samenwerking met perifere circuits, maar als in het programma rekening wordt gehouden met een mogelijke beïnvloeding van de 'overflow' door extreme gebeurtenissen, is gebruik ervan toegelaten.

De 6502 was één van de eerste processors met een uitgebreide adressering, mede door de twee indexregisters. De soort instructies van de 6502 geeft tabel 10.2.

1. Data manipulatie

- Rekenkundig en logisch.
- BCD rekenen via een control bit in het status register.
- Operaties op plaatsen in de adresruimte, hetzij geheugen of een I/O-poort.

2. Data verplaats

De 6502 kent een echte indexering via twee indexregisters, respectievelijk X en Y. De index-offset is beperkt doordat de registers 8 bits breed zijn. Er is een verkorte adresseringsvorm naar pagina '0'. Voor tabel manipulatie zijn er twee indirect-geïndexeerde en geïndexeerd indirecte instructies.

3. Programma-manipulatie

De stack pointer wordt gebruikt om een 'last in — first out' stapel te creëren in het werkgeheugen van maximaal 256 plaatsen.

De conditionele sprongen hebben een +/- relatieve offset.

Er is voorzien in een maskeerbare en een niet-maskeerbare interrupt.

4. Programmastatus

Het status register kan van en naar de stapel worden gebracht. Een aantal bits kan direct geset of gecleard worden. Eén bit wordt naar een aansluitpen gevoerd.

Tabel 10.2. Overzicht soorten 6502-instructies.

Tot de familie van de 6502 behoren de circuits 6522, 6525, 6530 en 6532. Hiervan is de 6522, de Versatile Interface Adaptor of VIA, een echt interface circuit; de drie andere zijn combinatie-circuits.

De 6522 heeft intern twee 8-bits poorten, voorzien van logica om data transport te begeleiden op 'handshake' basis. Verder zijn twee counter/timers toegevoegd en kent het circuit elementaire logica voor seriële communicatie.

De 6525 is een combinatie van drie 8-bits poorten en een geheugen van 64K ROM.

De 6530 is een combinatie van 1K×8 bits ROM, 64 bytes RAM, twee parallelle I/O-poorten, een programmeerbare interval timer en interrupt logica. De 6532 verschilt van de 6530 alleen in geheugen: de 6532 heeft 128 bytes RAM en geen ROM.

10.2.4. De 2650

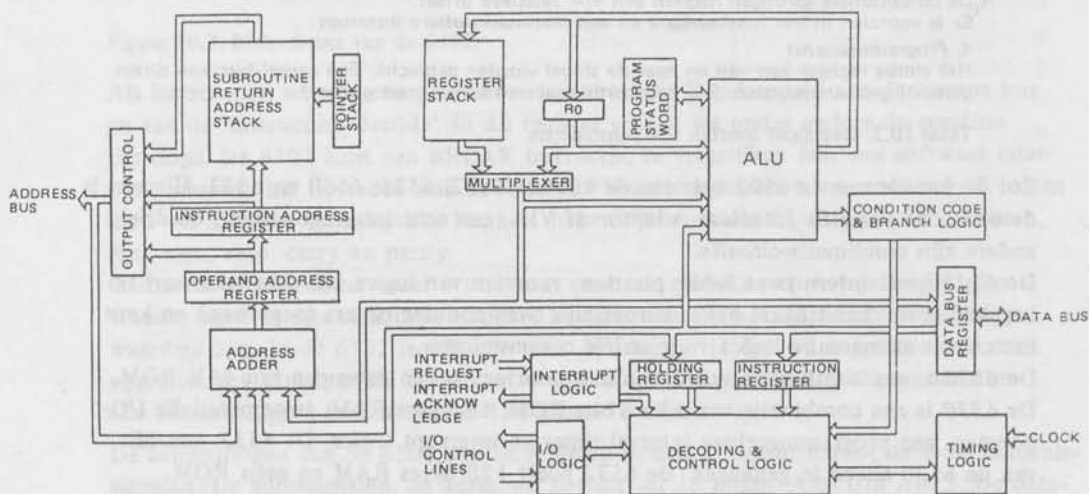
De microprocessor 2650 is een ten onrechte vrij onbekende processor. De chip heeft een aantal eigenschappen, die zeer aantrekkelijk zijn voor gebruikers, zowel in de software als in de hardware. Toch is de grote doorbraak voor dit type nooit gekomen, waarschijnlijk omdat de fabrikant (eerst Signetics, later Philips/Signetics) onvoldoende ondersteuning gaf in software en in systeemontwikkeling. Ook het vrij laat uitbrengen van enige perifere circuits zal meegespeeld hebben. Toen in 1973 de 2650 werd ontworpen, was deze zijn tijd ver vooruit, maar helaas is ook nu nog de ondersteuning veel minder dan bij de concurrerende merken. Echt nieuwe ontwerpen van ondersteuningscircuits zijn niet te verwachten.

De ontwerpers van de 2650 hadden tot doel een processor te maken zoals een programmeur die zou verwachten, iets waar ze redelijk in zijn geslaagd. De belangrijkste kenmerken zijn:

Technologie:	NMOS
Woordlengte:	8 bits
Adres:	15 bits
Instructielengte:	1 tot 3 bytes
Aantal instructies:	75
Instructie-tijd:	3 tot 18 klokcycli
Klokfrequentie:	(maximaal) 2 MHz
Second source:	National Semiconductor.

Een blokschema van de 2650 geeft figuur 10.3. Wat niet uit dit schema blijkt, is, dat de 2650 een processor is met een statische klok. De klokfrequentie kan tot aan DC worden teruggebracht, bij debuggen een handig hulpmiddel. Andere processoren zijn dynamisch en eisen daardoor altijd een minimale klokfrequentie omdat ze anders interne data verliezen.

In dit schema valt allereerst een gedeelte op aan de linkerkant, waar de adresbus naar buiten komt. We zien van onder naar boven: een 'address adder', 'operand address register' en 'instruction address register'. Deze drie delen zorgen ervoor dat de 2650 uitgebreide adresseringsfaciliteiten kent, omdat het adresmanipulaties toestaat. Erboven



Figuur 10.3. Blokschema van de 2650.

is in een apart blok een 'subroutine return address stack' aangegeven met de bijbehorende stack pointer. De 2650 heeft een zeven lagen diepe adresstapel op de chip zelf. Dit beperkt het aantal te bewaren terugkeeradressen, maar de toegang ertoe is snel.

Rechts beneden is de 'timing logic' met, ernaast, de 'timing & control logic', 'I/O logic' en 'interrupt logic'. Dit geheel komt overeen met het 'timing & control' deel uit figuur 1.2. Boven dit blok zien we een 'instruction register', een bekend onderdeel, en een 'holding register', waarin dat deel van een multiple byte instructie wordt bewaard, dat bij de executie van de instructie later nog nodig is.

Rechts zien we, in het midden, de databus buffer met schuin erboven de ALU en de conditiecode bits. Tenslotte, in het midden, een 'register stack', bevattende zes algemeen toepasbare registers en een register 'R0', de accumulator. Via een multiplexer kan een register worden geselecteerd naar een interne bus. Naast R0 een blokje 'program status word', een 16-bits register, waarin de waarden van de conditie flip-flops staan, één bit voor serieel in, één voor serieel uit, één bit wat aangeeft of de interrupt aan staat of niet, alsook de waarde van de stack pointer.

De 2650 chip heeft de bekende signalen, 15 adreslijnen en 8 datalijnen, verder een 'reset', 'clock', 'interrupt request', 'interrupt acknowledge', 'M/I/O', 'R/W', 'pause' (= wait) en 'WRP' (write pulse).

Afwijkende signalen zijn: ten eerste de OPREQ en de OPACK. Deze signalen begeleiden alle in- en uitgaande transport, pas als OPREQ actief is, hebben de andere controlesignalen (zoals M/I/O, R/W, enzovoorts) betekenis. Alle signalen behouden hun waarde totdat, naar de processor toe, de melding is gegeven dat de gevraagde actie is verricht door middel van het actief maken van OPACK. Dit levert een compleet asynchroon mechanisme op voor communicatie van de chip met de buitenwereld. De 2650 kent een 'sense' input en een 'flag' output, beiden kunnen dienen voor seriële communicatie, vergelijk de 8085.

Speciaal ten behoeve van een viertal één-byte I/O-instructies zijn de signalen 'D/C' (data/control) en 'E/NE' (extended/non extended).

In tabel 10.3 zijn de typen instructies opgenomen die de 2650 kent. Hierbij moet worden opgemerkt dat deze processor de volgende adresseringen kent:

- onmiddellijke adressering,
- relatieve adressering, al of niet indirect,
- absolute adressering, al of niet indirect,
- absolute adressering, al of niet met indexering,
- absolute adressering, indirect en met indexering.

1. Data manipulatie

- Rekenkundig en logisch.
- Een keuze uit acht adresseringsmodes, zoals hierboven al genoemd.

2. Data verplaats

'Load' en 'store' instructies in combinatie met alle adresseringen. Ieder van de zeven general purpose registers kan als indexregister dienst doen. De bus kan in een tri-state toestand worden gebracht, voor DMA.

3. Programma manipulatie

Conditionele sprongen, idem naar subroutine, met alle relatieve en absolute adresseringsmodes.

Speciale alles-in-één instructies voor 'loops'.

Enkel-niveau interrupt met extern aan te bieden vector.

4. Programma status

Een 16 bits breed status register met eigen instructies voor manipulatie van de bits.

Tabel 10.3. Overzicht soorten 2650-instructies.

Bij gebruik van een register als indexregister is het mogelijk om in de operatiecode mee te geven of het zonder meer een indexering is, indexering met auto-increment, of indexering met auto-decrement.

Bij relatieve en absolute adressering is het meestwaardige bit in gebruik om aan te geven of de adressering al of niet indirect is, reden waarom de 2650 maar 15 adreslijnen kent.

Tot de familie circuits rond de 2650 behoren de 2651, een programmeerbaar circuit voor seriële communicatie (inmiddels opgevolgd door een verbeterde versie met type-nummer 2661), de 2655, idem voor parallelle aansluiting en de 2652, een multi-protocol communicatie circuit.

De 2651 kan een serieel datatransport verzorgen in een synchrone of asynchrone mode met een programmeerbare Baud rate van DC tot 1 Mbit/s. Het circuit is programmeerbaar vanuit bijvoorbeeld een 2650 en kan (bij asynchroon transport) op 5 tot 8 bits woorden, type pariteit, aantal stop bits, enzovoorts worden geprogrammeerd. Bij synchroon transport is ook nog op te geven welk karakter als synchronisatie karakter en welk als 'delete' in gebruik is.

De 2655 kent drie programmeerbare 8-bits poorten en heeft bovendien nog een ingebouwde, ook programmeerbare, 16-bits timer. Poorten zijn afzonderlijk programmeerbaar in een 'strobed' input, een 'strobed' output, serieel in/uit, en één poort zelfs in een bidirectionele mode.

De 2652 ondersteunt communicatie volgens bepaalde standaard communicatie protocollen, zowel bit- als byte-georiënteerd. Het circuit verzorgt de formatting, het zenden en ontvangen, de controle op fouten zowel als de daarop te nemen maatregelen.

In de 267# serie levert Philips/Signetics ook nog een aantal circuits die niet direct als algemeen ondersteuningscircuit zijn te beschouwen, maar die zijn gericht op samenwerking met video en dan ook gericht zijn op de spelletjesmarkt.

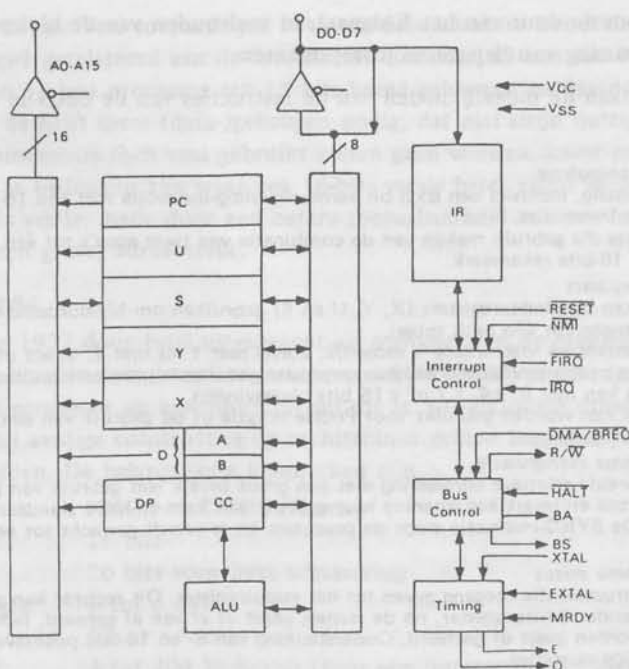
10.2.5. De 6809

De 6809 is te beschouwen als een sterk verbeterde 6800. Motorola heeft in deze processor uitgebreide 16-bit adresseringsfaciliteiten voorzien, door het aanbrengen van twee 16-bits indexregisters en een (tweede) 16-bits stack pointer voor de gebruiker, die ook als derde indexregister te gebruiken is. In de praktijk blijkt de 6809 een factor 2,5 à 3 sneller te zijn dan een 6800 met ongeveer 40% minder instructies, voor een vergelijkbaar programma.

De belangrijkste kenmerken zijn:

Technologie:	NMOS, CMOS (van Hitachi)
Woordlengte:	8 bits
Adres:	16 bits
Instructielengte:	1 tot 5 bytes
Aantal instructies:	59
Instructietijd:	2 tot 20 klokcycli
Klokfrequentie:	(maximaal) 2 MHz (busfrequentie)
Second source:	AMI, Fairchild, Hitachi.

Figuur 10.4. geeft het blokschema van de 6809. Door het uitbrengen van een processor die naar buiten een 8 bits breed uiterlijk heeft en intern (soms) met 16 bits werkt, heeft Motorola bereikt dat bij de systeembouw nog steeds alle 8-bits ondersteuningscircuits uit de 6800-reeks kunnen worden toegepast. In het blokschema zien we, van boven naar beneden in het midden: de programmateller (PC), de twee stapelwijzers (user stack U en hardware stack S), de indexregisters (X en Y), de accu-



Figuur 10.4. Blokschema van de 6809.

mulators (A en B) die in sommige instructies tot één 16-bits register (D) worden gecombineerd, een pagina register (8-bits, DP), de conditiecodes (CC) en tenslotte de ALU.

Bij bepaalde indexeringsmodes is een combinatie met auto-increment of -decrement toegelaten. Als de accumulators A en B tot één 16-bits register worden gecombineerd, fungeert accumulator A als hoogstwaardige deel.

De besturingssignalen bestaan uit interruptsignalen en busbesturing. De signalen $\overline{\text{NMI}}$, $\overline{\text{IRQ}}$ en $\overline{\text{FIQ}}$ vormen respectievelijk: een niet-maskeerbare interrupt, een 'gewone' interrupt en een snelle interrupt. Bij de snelle interrupt wordt alleen de conditiecode en de programmateller gered, in tegenstelling tot de IRQ en NMI waarbij de totale processorinhoud wordt gered op de stapel.

$\overline{\text{DMA/BREQ}}$ laat gebruik van de processorbus toe van buiten de chip, zoals voor DMA-toepassing, waarbij BA meldt dat de bus vrij is. Met $\overline{\text{HLT}}$ wordt de processor stilgezet na afhandeling van de lopende instructie, ook dan meldt BA de bus vrij. $\text{R}/\overline{\text{W}}$ is het lees/schrijfsignaal.

BS, bus status, heeft samen met BA, bus available, de volgende betekenis:

BA	BS	toestand
0	0	normaal
0	1	interrupt acknowledge
1	0	SYNC acknowledge
1	1	HALT of bus grant

XTAL en EXTAL dienen om de interne oscillator van de 6809 aan te sluiten op een extern kristal. De pen EXTAL kan ook worden aangestuurd met een externe klok op TTL-niveau, waarbij XTAL aan aarde wordt gelegd. De kristalfrequentie is viermaal die van de bus. E en Q zijn kloksignalen die naar buiten komen om externe aansluitingen te synchroniseren met het interne gebeuren.

MRDY dient om de duur van het E-sigitaal, in veelvoud van de klokperiode, te verlengen bij toepassing van 'langzame' componenten.

In tabel 10.4 staan de mogelijkheden van de instructies van de 6809 in de verschillende categoriën.

1. Data manipulatie

- Rekenkundig, inclusief een 8x8 bit vermenigvuldig-instructie met een 16-bit resultaat.
- Logische instructies.
- Instructies die gebruik maken van de combinatie van twee accu's tot één 16-bits register.
- Beperkt 16-bits rekenwerk.

2. Data verplaats

De 6809 kan vier indexregisters (X, Y, U en S) gebruiken om bijvoorbeeld drie blokken data te mengen tot één blok totaal.

Auto-increment en -decrement is mogelijk, zowel met 1 als met 2, direct of indirect.

Pagina 0 kan programmatisch worden verplaatst gedurende programma-uitvoer.

Indexering kan met 0, ± 4 , ± 7 of ± 15 bits plaatsvinden.

De U-stack kan worden gebruikt voor Poolse notatie of bij gebruik van een hogere programmeertaal.

3. Programma manipulatie

Een uitgebreide relatieve adressering met een groot bereik laat gebruik van plaats-onafhankelijke code toe en levert een opening naar gebruik van kant-en-klare standaard programmatuur in ROM. De SYNC-instructie stopt de processor en er wordt gewacht tot een interrupt optreedt.

4. Programma status

Er zijn instructies die toegang geven tot het statusregister. Dit register kan gewisseld worden met ieder ander 8-bits register, op de stapels gezet of er van af gehaald. Ieder aantal vlag-bits kan worden geset of gecleard. Ondersteuning van 8- en 16-bits positieve en negatieve rekenkundige operaties.

Tabel 10.4. Overzicht soorten 6809 instructies.

Bij de 6809 kunnen alle ondersteuningscircuits van de 6800 worden gebruikt. De bekendste zijn:

6820, een peripheral interface adapter (PIA), welke twee 8-bits I/O-poorten kent.

Iedere pen van een poort kan als in- of uitvoer worden geprogrammeerd. Bij iedere poort is een tweetal signalen voor handshake.

6850 en **6852**, respectievelijk: de asynchronous communications interface adapter (ACIA) en de synchronous serial data adapter (SSDA), welke ondersteuning leveren voor asynchrone, respectievelijk synchrone communicatie. (6850 en 6852 samen zijn vergelijkbaar met de 8251 uit de 8085-familie.)

De belangrijkste circuits uit de verdere reeks zijn: de 6828, priority interrupt controller; 6840, programmable counter/timer; 6844, DMA controller en tenslotte de 6848, een combinatie van $2K \times 8$ ROM, een 8-bit I/O-poort en een counter/timer.

10.3. 16-bits microprocessors

10.3.1. Algemeen

Nadat we in de ontwikkeling van de microprocessors de stap van 4-bit naar 8-bit typen hebben meegemaakt, was het niet meer dan een kwestie van tijd voordat de stap van 8- naar 16-bit typen werd gedaan. In 1977 bracht Intel als één der eersten een 16-bits processor op de markt, de 8086, spoedig gevolgd door andere fabrikanten. De ontwikkeling van 8 naar 16 bit is een vrij logische als we bedenken dat de verbetering van integratietechnieken steeds meer componenten op een chip toeliet. De ontwikkeling was dan ook tweeledig, enerzijds een fabricage van snellere processors door kleinere afmetingen van de chip en anderzijds een toename van het aantal onderdelen op de chip en ook daarmee een toename van de capaciteit.

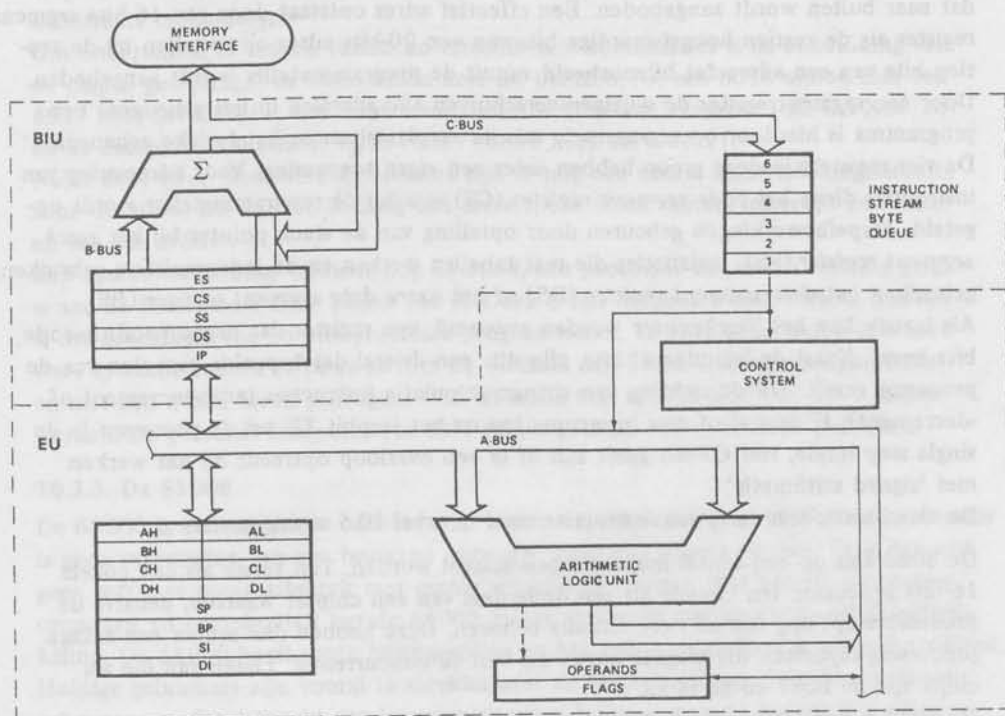
Het is niet te verwachten dat de 16-bits processors de 8-bits versies zullen verdringen. Een 8-bits processor is namelijk in veel gevallen beter en efficiënter toe te passen dan

zijn 16-bits collega. Veel toepassingen bevatten data/karakter bewerkingen en zijn daardoor 1-op-1 gerelateerd aan de breedte van het datapad van een 8-bits processor. Ook heeft een 16-bits processor een 16 bits breed geheugen en daardoor voor 8 bits toepassingen de helft meer (data-)geheugen nodig, dat niet altijd nuttig wordt gebruikt. Dat 16-bits processors toch veel gebruikt zullen gaan worden, komt omdat er talloze toepassingen te bedenken zijn waar een 16-bits versie beter en/of sneller zal werken dan een 8-bits versie, mede door een betere toepasbaarheid van een hogere programmeertaal en een groter adresbereik.

10.3.2. De 8086

De 8086 is in 1977 door Intel uitgebracht als opvolger van de 8080/8085 serie. Hierbij is rekening gehouden met de software voor een 8080/8085, in die zin, dat programma's voor de 8-bit processor op assembler niveau op de 8086 kunnen draaien. In de 8086 zijn een aantal aardige constructies op architectuur gebied toegepast, die verder nog genoemd worden. De belangrijkste kenmerken zijn:

Technologie:	NMOS en CMOS (Harris)
Woordlengte:	16 bits
Adres:	20 bits voor byte adressering
Instructielengte:	1 tot 6 bytes
Aantal instructies:	97
Instructietijd:	4 tot 204 klokcycli (voor een 'integer divide' met extended address adresssing)
Klokfrequentie:	(maximaal) 10 MHz
Second source:	AMD, Fujitsu, Harris, Mitsubishi, NEC, Siemens.



Figuur 10.5. Blokschema van de 8086.

De 8086 is, wat opbouw betreft, in twee delen te scheiden, een bus interface unit (BIU) en een execution unit (EU) die erg nauw samenwerken, maar voor een belangrijk deel onafhankelijk van elkaar bezig zijn. In figuur 10.5 staat het blokschema van de 8086.

De EU voert de elementaire processing functions uit, zij bevat de data registers en de ALU. De EU haalt instructies van de BIU en stuurt niet-gerelocateerde operand-adressen terug. Dan ontvangt de EU operands uit het geheugen via de BIU, verwerkt ze, en stuurt resultaten terug naar de BIU voor opslag.

De BIU is toegevoegd om een maximaal profijt van de busbandbreedte te halen, aangezien dat de belangrijkste beperkende factor van de processor-verwerkingssnelheid is.

De BIU haalt instructies van tevoren, al voordat de EU deze nodig heeft. De instructies worden gebufferd in een stapel van maximaal zes bytes, waar ze worden opgeslagen in afwachting van decodering en executie. Als gevolg van deze organisatie hoeft de EU niet te wachten als zij een volgende instructie nodig heeft. De BIU voorziet in de functies die nodig zijn voor de 'operand fetch' en 'store', adres relocatie en busbesturing, dit alles parallel aan de werking van de EU.

In de 8086 onderscheidt men drie sets van ieder vier 16-bits registers, te weten: de algemene registers, de pointer & index registers en de segment registers.

De *algemene registers* zijn voor algemeen gebruik met als bijzonderheid dat ze ook per hoogst- of laagstwaardige byte zijn te adresseren, overeenkomstig de configuratie van de 8085.

Binnen de *pointer & index register* groep heeft ieder register een eigen betekenis, namelijk: een stack pointer, een base pointer, een indexregister voor de bron-data en een indexregister voor de bestemmingsdata.

De *segment registers* gebruikt de 8086 bij berekening van het adres van 20 bit breed, dat naar buiten wordt aangeboden. Een effectief adres ontstaat door een 16-bits segment register als de zestien hoogstwaardige bits van een 20-bits adres op te tellen bij de zestien bits van een adres dat bijvoorbeeld vanuit de programmateller wordt aangeboden. Door een segmentregister de wijzigen verschuiven alle adressen in het geheugen. Een programma is hierdoor op een simpele manier verplaatsbaar in het fysieke geheugen. De vier registers in deze groep hebben ieder een eigen toepassing. Voor adressering van instructies dient het *code segment register* (CS) waarbij de programmateller wordt opgeteld. Stapelbewerkingen gebeuren door optelling van de stack pointer bij het *stack segment register* (SS). Instructies die met tabellen werken en de indexregisters gebruiken, gebruiken het *data segment register* (DS) of het *extra data segment register* (ES).

Als laatste kan het 'flag' register worden genoemd, een register dat negen conditie-code bits bevat. Naast de 'standaard' bits, zijn dit: een drietal dat bepaalde aspecten van de processor regelt: DF de richting van stringmanipulatie instructies (auto-increment of -decrement); IF laat al of niet interrupts toe en het trapbit TF zet de processor in de single step mode. Het OF-bit geeft aan of er een overloop optreedt bij het werken met 'signed arithmetic'.

De classificatie van de typen instructies staat in tabel 10.5 weergegeven.

De 8086 kan op een aantal manieren beschouwd worden. Ten eerste als een enkele 16-bits processor, ten tweede als een onderdeel van een chipset waartoe, behalve de processorchip, nog één of twee circuits behoren. Deze hebben dan samen een zekere processing-capaciteit die uitgaat boven die van de concurrentie. Leden van die set chips zijn de 8087 en de 8089.

De 8087 is een numerieke processor die puur gericht is op rekenkundige functies. De chip doet dit ongeveer 100× sneller dan dat met de 8086 mogelijk zou zijn. Toevoe-

1. Data manipulatie

- 8- en 16-bits rekenkundige instructies met en zonder teken, zowel binair als decimaal.
- Logische instructies.
- Vermenigvuldig- en deelinstructies.
- Bit-, byte- en blok-operaties.

2. Data verplaats

De 8086 kent een uitgebreide reeks adresseringsmodes, met relatieve adressering ten opzichte van registers of segmenten.

Met segmentregisters kan de gebruiker door het geheugen heen, vier gebieden opgeven: voor programma, voor stack en twee voor data. Deze gebieden zijn willekeurig groot en mogen elkaar overlappen.

3. Programma manipulatie

De processor kent sprongen, subroutine aanroepen en terugkeer uit subroutines binnen segmenten en ook naar andere segmenten toe. Conditionele sprongen op een logische toestand kan relatief binnen ± 128 bytes vanaf de PC.

Speciale mogelijkheden voor iteraties binnen 'loops' en een prefix voor snelle iteraties in herhaalde string manipulaties.

4. Programma status

Er zijn instructies die toegang geven tot het status register. De conditiebits zijn een uitbreiding van die van de 8085.

Tabel 10.5. Overzicht soorten instructies van de 8086.

ging van de chip aan de 8086 is transparant voor de programmeur, het lijkt of de instructieset van de 8086 is uitgebreid. De 8087 heeft intern een 84 bits breed datapad en doet een bewerking op 32-bits floating point getallen — bijvoorbeeld optellen/af-trekken — in $17\mu s$, een vermenigvuldiging in $19\mu s$, worteltrekken in $36\mu s$ en een goniometrische functie binnen $90\mu s$. Dit alles bij een, op 5 MHz werkende, 8086. Alle getalrepresentaties volgens de IEEE floating point standaard.

De 8089 is een I/O processor (IOP), een processor die intern is toegespitst op het ver-richten van I/O operaties, zoals data van en naar de bus transporteren. De opzet is vergelijkbaar met I/O kanaal processors in de vorm waarin ze voorkomen bij mini- en mainframe computers.

Om onderscheid te maken tussen de verschillende combinaties is de aanduiding van de chip(s) gewijzigd; de 8086-alleen heet nu iAPX86/10, een 8086 samen met een 8089 heet iAPX86/11, een 8086 samen met 8087 wordt aangeduid als iAPX86/20 en de complete combinatie 8086+8087+8089 heet iAPX86/21.

Naast deze twee 'dedicated' processors zijn er nog een aantal ondersteuningscircuits rond de 8086. Dit zijn de latches, bus drivers, een clock driver, interrupt controller en een bus arbiter.

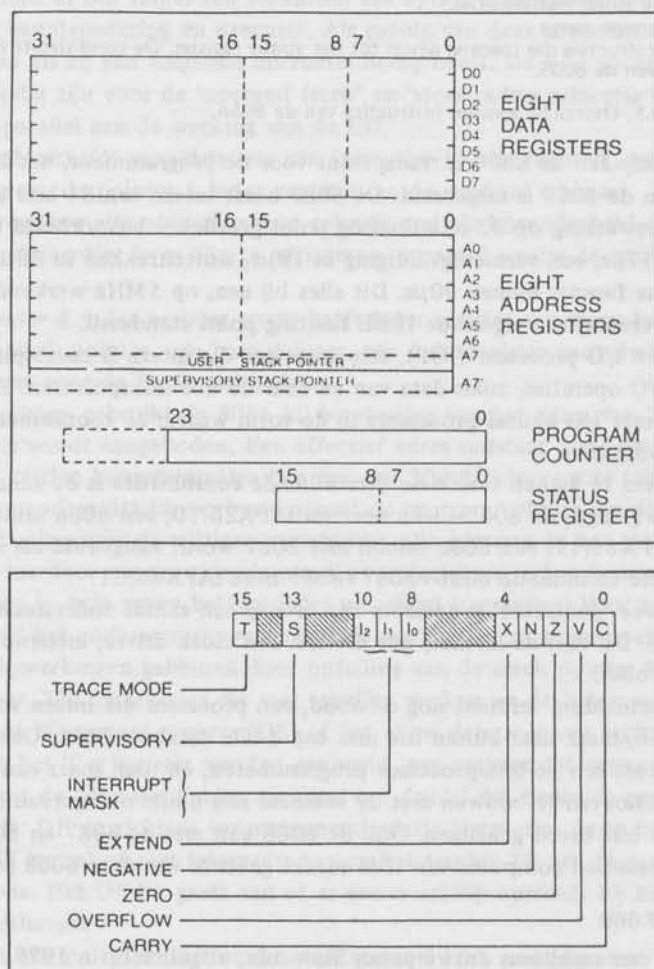
Een aparte vermelding verdient nog de 8088, een processor die intern volledig gelijk is aan de 8086, maar naar buiten toe met een 8-bits databus werkt. Gevolg hiervan is dat gebruikers een 16-bits processor programmeren, en toch maar een 8-bits hardware systeem hoeven te bouwen met de veelheid aan 8-bits ondersteuningscircuits en met een 8 bits breed geheugen. Ook de 8088 kan met de 8087 en 8089 samenwerken. De personal computer van IBM maakt gebruik van deze 8088 processor.

10.3.3. De 68000

De 68000 is een ambitieus ontwerp van Motorola, uitgebracht in 1979. Deze processor is geen verbetering van een bestaand ontwerp, maar een geheel nieuwe. Er is dan ook geen software compatibiliteit met eerder uitgebrachte types. Wat betreft processing capaciteit en complexiteit ligt de 68000 in het verlengde van de 6800-6809 ontwikkeling. De 68000 heeft grote belangstelling en het aantal gebruikers is nu snel groeiend. Huidige gebruikers zijn vooral te spreken over de manier waarop grote hoeveelheden geheugen worden bewerkt zonder segmentering. De voornaamste kenmerken zijn:

Technologie: NMOS
 Wordlengte: 16 bits
 Adres: 23 bits voor woord-adressering
 Instructielengte: 1 tot 3 woorden (van 16 bits)
 Aantal instructies: 61
 Instructietijd: 4 tot 190 klokeycli (voor een 'signed divide' met extended addressing)
 Klokfrequentie: (maximaal) 12,5 MHz
 Second source: Hitachi, Mostek, Rockwell, Signetics/Philips, Thompson/CSF.

Het schema van de 68000, zoals de programmeur dat ziet, geeft figuur 10.6. Het statusregister is nog eens apart weergegeven.



Figuur 10.6. Registeropbouw (boven) en statusregister (onder) van de 68000.

De 68000 is een 16-bits processor, die intern een groot aantal bewerkingen kan uitvoeren op 32-bits woorden. De registerset is namelijk opgebouwd uit acht dataregisters en acht adresregisters die ieder 32 bits breed zijn. Eén van deze registers (A7) is voorbestemd om als gebruikers stack pointer te fungeren, daarnaast komt nog een systeem stack pointer voor.

De programmateller is 24 bits breed, hiervan komen de 23 meestwaardige bits naar buiten op de adresbus en uit de chipsignalen UDS en LDS (upper- en lower data strobe) is af te leiden of het om een hoog of laag byte adres gaat. In het 16-bits statusregister vinden we de conditiecodes, waaronder ook: of de processor in de 'supervisory' of in de 'trace' mode werkt, alsmede een 3-bits interrupt masker.

Behalve de ALU, die 16 bits breed is, kent de processor nog twee 16 bits brede AU's (rekenkundige eenheden, geen logische), die tegelijkertijd berekeningen kunnen uitvoeren op de twee 16-bits helften van de adresregisters.

In tabel 10.6. is aangegeven hoe de typen instructies zijn te classificeren:

1. Data manipulatie

- Rekenkundige instructies met en zonder teken.
- Logische instructies.
- Vermenigvuldig- (16x16) en deel instructies (32:16) met en zonder tekenbit.
- Bit, BCD-nibbles (4 bits), byte, short (16 bits) en long (32 bits) woorden.

2. Data verplaats

De 68000 kent een uitgebreide reeks adresseringsmodes waaraan kan worden toegevoegd: post-incrementing, pre-decrementing, een offset en indexing.

De acht 32-bits registers kunnen als indexregister dienstdoen, maar ook als stackpointer.

De acht 32-bits data registers kunnen ook als indexregister worden gebruikt.

3. Programma manipulatie

De processor kent (conditionele) sprongen en subroutine aanroepen. Link en unlink instructies vormen een basis voor het opzetten van lokale gebieden ten behoeve van gestructureerde programmeren.

Zeven interruptniveaus, inclusief een niet-maskeerbare, met maximaal 256 interrupt vectors.

4. Programma status

Er zijn instructies die toegang geven tot het status register. Een 'trap' instructie vereenvoudigt het testen van programmatuur. Een mogelijkheid tot het volgen van het programma-verloop in een 'trace mode', waarin na iedere instructie de besturing wordt overgegeven aan een monitor programma.

Tabel 10.6. Overzicht soorten instructies van de 68000.

De 68000 kent twee werkingsniveaus, te weten: het supervisor of systeemniveau en het user of gebruikersniveau.

Op *systeemniveau* zal meestal het operating system draaien, terwijl de gebruikersprogramma's op het *user-niveau* thuishoren. Er zijn instructies die alleen op systeemniveau kunnen worden uitgevoerd. Ieder niveau heeft zijn eigen stapelwijzer, die in beide gevallen wordt geadresseerd als A7. Onderscheid tussen deze twee is mogelijk door gebruik van het statusbit (bit 13) in het statusregister, dat aangeeft of de processor al dan niet in de 'supervisory' mode werkt.

De 68000 heeft een aantal instructies die het werken met een hogere programmeertaal vereenvoudigen, zoals de MOVEM, welke alle registers op de stapel zet en de LINK/UNLINK instructies die werkruimte reserveren.

Door een serie 'TEST en SET' instructies, die in één cyclus een bit kunnen testen en veranderen, kunnen met behulp van semaforen multiprocessor systemen worden gebouwd.

Als ondersteuningscircuits kent de 68000 er een viertal:

68451, een memory management unit;

68120, intelligent peripheral controller;

68230, parallel interface;

en drie versies van de Signetics reeks communicatie-chips: 2652, 2653 en 2661.

10.4. Enkelchip computers

10.4.1. Inleiding

Bij de inleiding van paragraaf 10.3 is al gewezen op de invloed die de verbeterde integratietechnieken hadden op de ontwikkeling van microprocessors. Dat dit leidde tot

de stap van 8- naar 16-bits versies, was niet het enige gevolg. Ook werd er in een bestaand circuit gekeken naar wat er aan zou kunnen worden toegevoegd om de gebruiksmogelijkheden te vergroten. Er verschenen circuits met een combinatie van functies, bijvoorbeeld: RAM + ROM + I/O.

Maar er was ook een trend naar alleenstaande computerchips, dus een processor waaraan wat geheugen en wat I/O-faciliteiten zijn toegevoegd. Eerst alleen voor 4-bits processors, later ook voor 8-bit versies. Bij deze circuits moeten we goed bedenken dat een dergelijke computer altijd fysieke grenzen kent aan het aantal pennen dat aan het circuit zit, dus de hoeveelheid I/O-lijnen is beperkt. Toch zijn er talloze, beperkte toepassingen waar een enkelchips microcomputer voldoende biedt om de applicatie te realiseren. Zo zijn er vooral in de wereld van de consumentengoederen, met een sterke gerichtheid op calculator-achtige toepassingen, hele series van mogelijkheden voor 4-bits processors. Bijvoorbeeld: rekenmachines, wasmachines, weegschalen, drankenautomaten, kasregisters, enzovoorts.

Vooral in de 4-bits markt zien we chips met ver doorgevoerde in- en uitgangsmogelijkheden voor aansturing van LCD-displays, A/D-omzetters, serie interfaces, enzovoorts. Onder de 4-bits vallen onder andere de TMS1000 serie van Texas Instruments (veruit de meest verkochte), de μ COM van NEC, de COP400 van National Semiconductor en de PPS4 familie van Rockwell.

De bekendsten onder de 8-bit typen zijn onder andere Intel's 8048 (en 8051) reeks, de PIC1650 van General Instruments en de 3870 van Mostek Corp.

Van ieder van deze soorten wordt er één behandeld in de volgende paragrafen, en wel de TMS1000 en de 8048 reeks.

10.4.2. De TMS1000

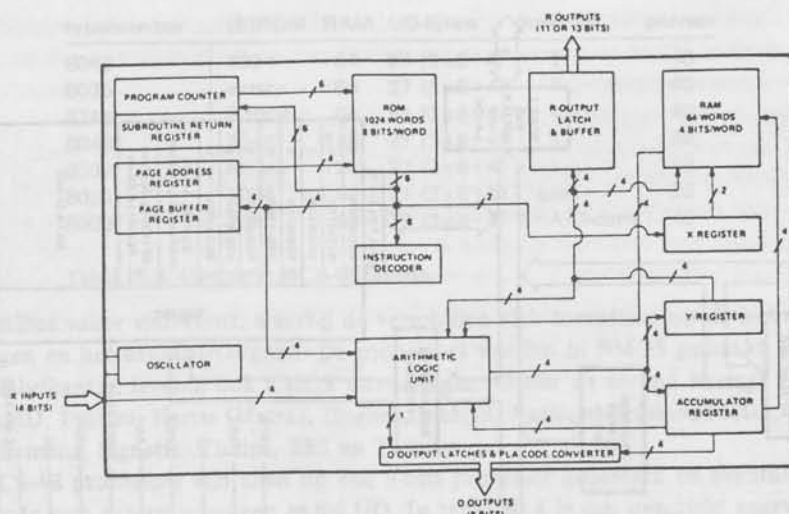
Sinds de introductie van de TMS1000 in 1974 heeft deze reeks veruit de hoogste verkoop aantallen bereikt van alle typen. De verdere uitbouw van deze reeks richt zich op twee gebieden, ten eerste een toenemend gebruik van CMOS en/of low-power NMOS en ten tweede een grote mate van 'klanten-versies', speciaal voor één afnemer gemaakte uitvoeringen van de basis TMS1000 versie. Dit laatste voor grote aantallen, waarbij 'groot' steeds kleiner wordt, in absolute zin.

Van een aantal versies maakt Motorola een kopie in CMOS. Texas Instruments maakt zelf ook CMOS, maar dan wordt tegelijk een uitbreiding van de functies gerealiseerd.

Van de standaardversies bestaan er een zevental, die verschillen in: hoeveelheid interne ROM, hoeveelheid RAM en aantallen I/O-lijnen. In tabel 10.7 staat een overzicht van deze serie.

typenummer	techniek	klok kHz	geheugen		I/O pennen						
			ROM	RAM	K/LK	L	O	R	pennen		
TMS1700	PMOS	300	512	32	—	4	—	8	9	28	
TMS1000	PMOS	300	1024	64	—	4	—	8	11	28	
TMS1200	PMOS	300	1024	64	—	4	—	8	13	40	
TMS1100	PMOS	300	2048	128	—	4	—	8	13	28	
TMS1300	PMOS	300	2048	64	—	4	—	8	13	40	
TMS1400	PMOS	500	4096	128	—	4	—	8	10	28	
TMS1600	PMOS	500	4096	128	1	4	4	8	16	40	

Tabel 10.7. Overzicht TMS1000-serie. De hoeveelheden geheugen zijn uitgedrukt in bytes.



Figuur 10.7. Blokschema van de TMS1000.

De I/O-lijnen vragen nog een nadere toelichting. De 'K' inputs zijn algemeen toepasbare inputs, hoewel ze in het begin geacht werden van een toetsenbord te komen (vandaar de K van 'keyboard'). De 'O' outputs zijn gewone outputs, altijd acht, oorspronkelijk voor aansturing van LED displays, en dergelijke. De 'R' outputs zijn besturingssignalen die bijvoorbeeld kunnen aangeven welk display of welk segment van een display wordt geactiveerd.

De 'O' outputs worden gedecodeerd uit een vijftal datalatches in de chip, omdat deze decodering met een PLA (programmable logic array) gebeurt, heeft zelfs de afnemer nog de keuze om deze decodering op te geven, samen met de gewenste ROM-inhoud. Zelfs specificatie van een eigen microprogramma is nog mogelijk, de fabrikant doet dit in dezelfde processlag als waarin ROM- en PLA-programmering gebeurt.

In het blokschema van de TMS1000, dat in figuur 10.7 is weergegeven, zien we bekende onderdelen als: ALU, accumulator, instructie decoder, ROM en RAM. Daarnaast de output latches en de PLA decodering. Ook de 'R' outputs hebben een latch + buffer in de chip. De programmateller is 6-bits, welke samen met de inhoud van een pagina adres register de totale 10 bits oplevert, nodig om het 1024-bits ROM te adresseren (het blokschema geldt voor de TMS1000). Deze ROM is 8 bits breed, de instructies bestaan dan ook uit 8-bits woorden.

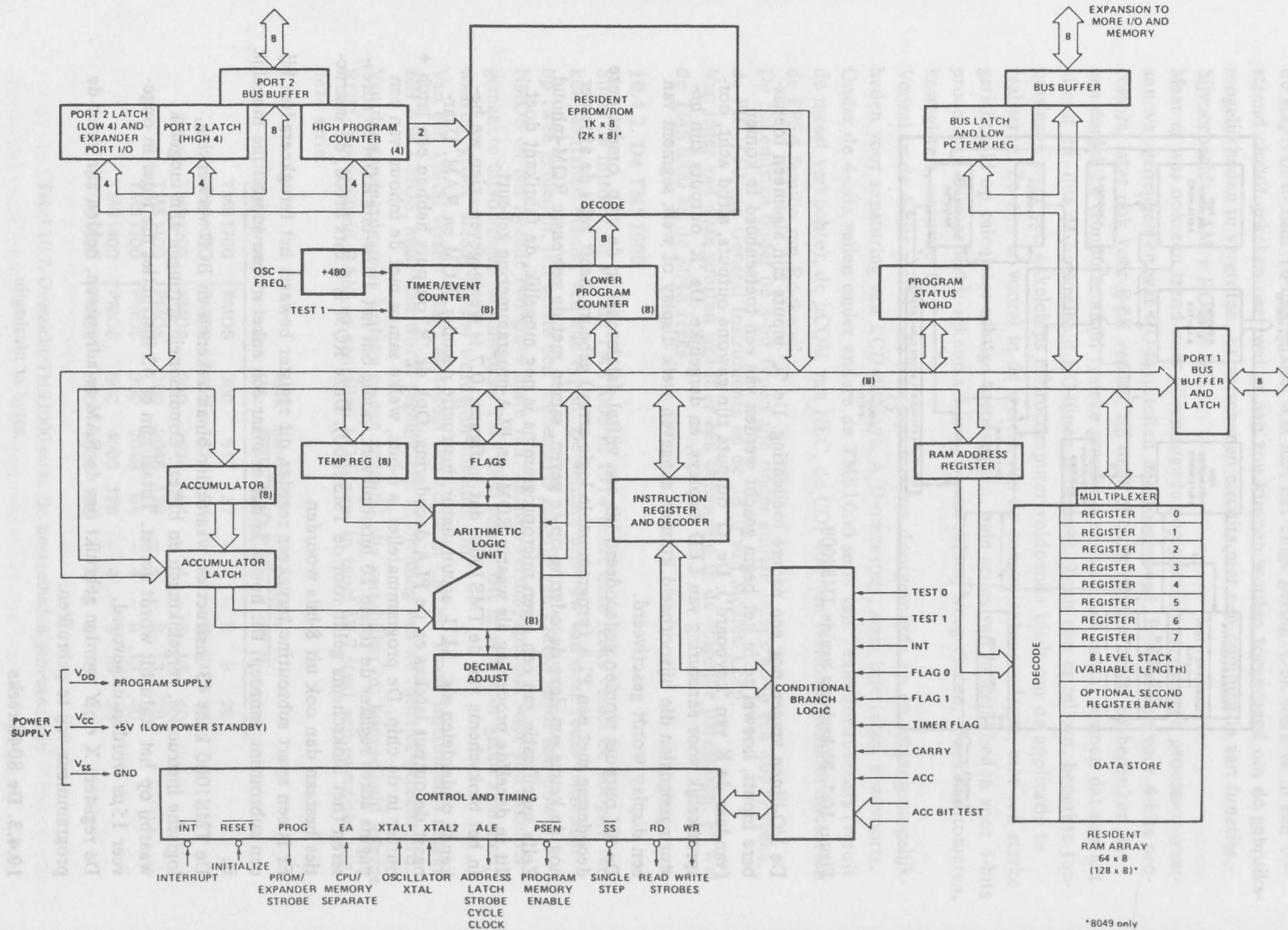
Er is een apart subroutine terugkeer register, dit register bewaart het terugkeeradres bij een subroutine aanroep. Dit houdt in dat er maar één enkel niveau subroutine mogelijk is.

De TMS1000 kent 43 instructies, waaronder binair rekenen en BCD verwerking, logische instructies, vergelijkingen en bittest. Conditionele sprongen zijn mogelijk, waarbij op het statusbit wordt getest. Totaal zijn er 43 instructies die ieder in ongeveer 15 μ s worden uitgevoerd.

De registers X en Y worden gebruikt om de RAM te adresseren, beiden zijn door de programmeur vrij te gebruiken.

10.4.3. De 8048-reeks

Evenals de vorige besproken processor, is ook de 8048 er één uit een reeks van processors, de MCS-48 familie. Deze vormen samen een op elkaar lijkend geheel, zoals



*8049 only

Figuur 10.8. Blokschema van de 8048.

typenummer	(E)ROM	RAM	I/O-lijnen	interrupt	pennen
8048	1024	64	27 (3x8+4)	1	40
8035	extern	64	27 (3x8+4)	1	40
8748	E1024	64	27 (3x8+4)	1	40
8049	2048	128	27 (3x8+4)	1	40
8039	extern	128	27 (3x8+4)	1	40
8021	1024	64	21 (2x8+5)	geen	28
8022	2048	64	28 (3x8+4)	2+A/D-conv.	40

Tabel 10.8. Overzicht MCS-48 familie.

bij families vaker voorkomt, waarbij de verschillen zich toespitsen op de hoeveelheid geheugen en het aantal I/O-lijnen. De processors worden in NMOS gemaakt en sommige fabrikanten leveren ook CMOS uitvoeringen. Onder de second sources vinden we: AMD, Fujitsu, Harris (Matra), Hughes, Intersil, National Semiconductor, NEC, Oki, Siemens, Signetics/Philips, SSS en Toshiba.

De MCS-48 processors zijn allen op een 8-bits processor gebaseerd en eventueel uitbreidbaar met extern geheugen en/of I/O. In tabel 10.8 is een overzicht gegeven van de verschijningsvormen van de MCS-48 processors met de bijbehorende kenmerken. De 8035 en de 8039 zijn gelijk aan respectievelijk de 8048 en de 8049, met als enig verschil dat de eersten geen ROM op de chip hebben. De gebruiker kan dan net zoveel extern geheugen aansluiten als hij wil en ook de keuze van de soort geheugen is dan vrij.

De 8748 is gelijk aan de 8048, maar heeft EPROM in plaats van ROM, wat een uiterst handzaam circuit oplevert voor prototype ontwikkeling.

De 8021 is een gestripte versie van de 8048, met minder aansluitingen, waardoor de uitvoering wat voordeliger is; belangrijk bij zeer grote aantallen. De instructieset is een subset van die van de 8048.

De 8022 is een computer, die, uitgaand van de 8021, toegevoegde I/O, geheugen en een A/D-converter heeft, met de daarbij behorende instructies.

Alle circuits hebben een ingebouwde timer/counter en zijn voorzien van een aansluiting om op simpele wijze de hoeveelheid I/O-lijnen uit te breiden. Hiertoe is er een apart circuit, de 8743, dat 16 extra lijnen levert.

Ook de ondersteuningscircuits uit de 8085 reeks (zie 10.2.2) zijn zonder meer op de MCS-48 aan te sluiten.

De MCS-48 reeks kent instructies welke 1 of 2 bytes beslaan en die executietijden hebben tussen de 1 en 10 μ s, mede afhankelijk van de uitvoering van het circuit.

Figuur 10.8 geeft het blokschema van de 8048.

In het schema zijn de ROM en RAM te onderscheiden, RAM is onderverdeeld in een aantal plaatsen, die als registers, stack of anderszins worden gebruikt. Bepaalde instructies houden daar rekening mee. De in-/uitgangspoorten 'port 1' en 'port 2' geven zestien I/O-lijnen, uitbreidbaar met de acht van 'port 0', die ook gebruikt worden als een uitbreiding van het aantal I/O-lijnen nodig is. Bij de 'control & timing' zien we een aantal signalen, die speciaal voor uitbreiding zijn aangebracht, onder andere: prom expander strobe, address latch enable en de read en write strobes. Verder zien we een aantal onderdelen die ook in een 'gewone' microprocessor voorkomen: ALU, accumulator, een 12-bits programmateller en een program status word.

De componenten uit de MCS-48 reeks leveren in de praktijk een oplossing die in prestatie vergelijkbaar is met een systeem opgebouwd uit losse microprocessor-circuits, de prijs is echter beduidend lager, ook al omdat minder circuits ook minder administratieve handelingen, opslag, montagetijd en -ruimte vergen dan een meer-circuits op-

lossing. Deze laatste argumenten blijken in de praktijk bij kleinere series al een voldoende positief effect te sorteren om toepassing van de 8748 waar te maken, ondanks de hogere componentenprijs.

Als 'opgevoerde' versie van de MCS-48 serie heeft Intel inmiddels een 8051 (met 8031 en 8751 als modellen met variaties in het geheugen) uitgebracht. Deze heeft ten opzichte van de 8048: 4K ROM, 128 byte RAM, extern uitbreidbaar tot 128K. Verder vier 8-bits poorten, een on-chip UART voor seriële communicatie tot 31Kbaud, twee 16-bits timer/counters met interrupt en twee externe, vectored interrupts. De instructieset kent ook een vermenigvuldig- en een deel instructie. De architectuur is duidelijk verbeterd met als extra effect dat de instructieverwerkingstijd rond de 1 μ s (vermenigvuldiging in 4) ligt. Een CMOS-versie met 8K aan ROM is aangekondigd.

11. Bit-slice processorbouwstenen

11.1. Inleiding

Hoewel in de afgelopen jaren de capaciteit van de microprocessors (en het gebruik ervan) een grote vlucht heeft genomen, zijn er ondanks al deze verbeteringen toch toepassingsgebieden, waar dit type microprocessors niet te gebruiken is. De belangrijkste kenmerken immers van de microprocessor zijn:

- beperkte woordlengte,
- vaste instructieset,
- MOS-snelheid.

Hieruit valt af te leiden dat, met name in die gebieden waar het aankomt op een hoge dataverwerkingssnelheid, of waar een speciale organisatie of datapadbreedte vereist is, een type microprocessor met bovengenoemde eigenschappen niet gebruikt kan worden. Voor wat betreft de snelheidseisen lijken de snellere technologieën (TTL, Schottky-TTL, ECL) een oplossing te bieden, maar door lagere integratiedichtheid en vooral hogere dissipatie is dit geen reëel alternatief.

De oplossing is dan het splitsen van de microprocessor in functionele delen. Wat dan ontstaat is de bit-slice microprocessor, te beschouwen als een n-bits centraal processorgedeelte met aritmetische en logische faciliteiten en één of meer registers. De instructiedecodering, besturing van geheugen en I/O, enz., moet dan apart en met zogenaamde *wilde logica* gerealiseerd worden.

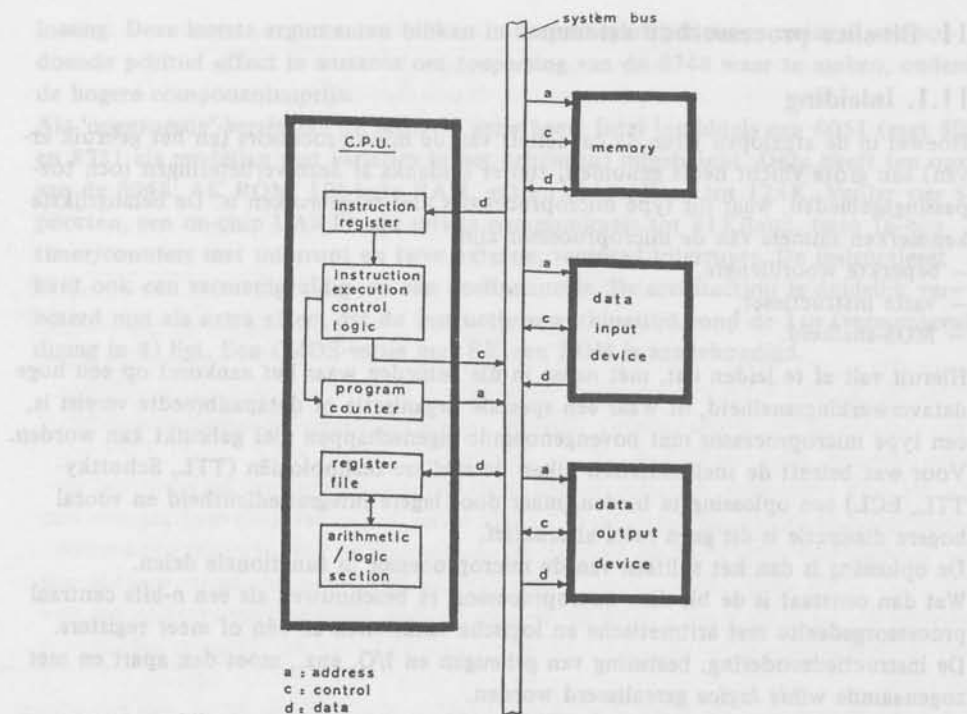
Tegenover de toegenomen ontwerp-problematiek staat nu een aanzienlijke winst in verwerkingssnelheid; één cycle time van bijvoorbeeld een Signetics 2650 is 2,4 *micro*-seconde, met een bit-slice microprocessor is een cycle time van 45 *nanoseconde* haalbaar.

11.2. De centrale processorbouwsteen

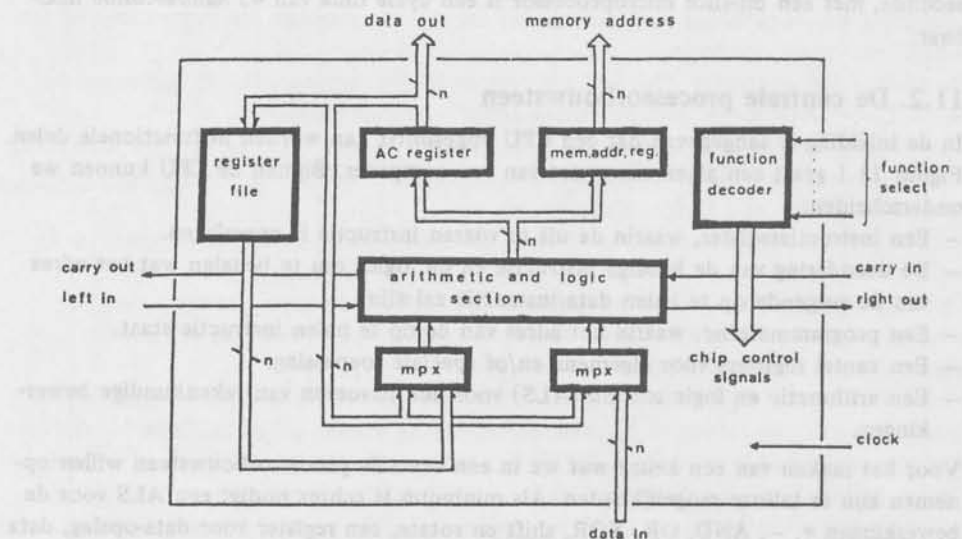
In de inleiding is aangegeven dat een CPU opgesplitst kan worden in functionele delen. Figuur 11.1 geeft een algemene opzet van een computer. Binnen de CPU kunnen we onderscheiden:

- Een instructieregister, waarin de uit te voeren instructie is opgeslagen.
- De decodering van de huidige instructie en de logica om te bepalen wat het adres van de volgende op te halen data/instructie zal zijn.
- Een programmateller, waarin het adres van de op te halen instructie staat.
- Een aantal registers voor algemene en/of speciale toepassing.
- Een arithmetic en logic section (ALS) voor het uitvoeren van rekenkundige bewerkingen.

Voor het maken van een keuze wat we in een centrale processorbouwsteen willen opnemen zijn er talloze mogelijkheden. Als minimum is echter nodig: een ALS voor de bewerkingen +, -, AND, OR, XOR, shift en rotate, een register voor data-opslag, data in- en uitvoerfaciliteiten, signalen voor koppeling met naastgelegen bouwstenen en een functiedecoderingsschakeling. Een mogelijke opbouw geeft figuur 11.2. In deze opzet is een *register file* opgenomen, onder andere voor opslag van tussenresultaten en/of data. Een verder voordeel van een register file kan gehaald worden uit het toekennen van een speciale functie aan een register. Zo kan één van de registers gebruikt worden als base- of indexregister voor gebruik bij de overeenkomstige adresseringsmodes; ook kan aan een register de functie van (micro)programmateller of stack pointer worden toegekend.



Figuur 11.1. Algemene opzet van een computer.



Figuur 11.2. Blokschema van een n -bits processorbouwsteen.

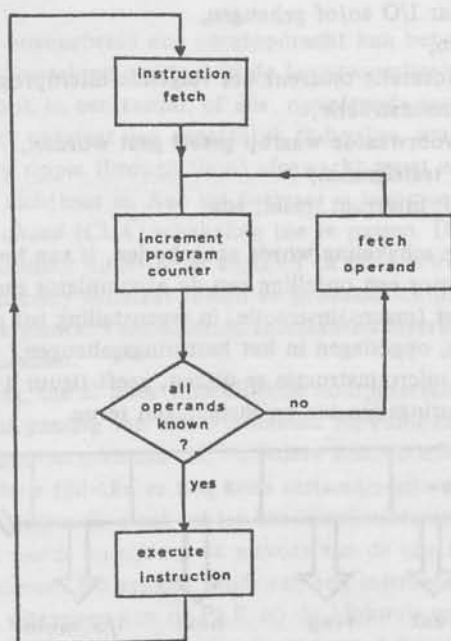
Verder is in figuur 11.2 bij ieder van de twee ingangen van de ALS een multiplexer opgenomen, welke onder besturing van de *chip control signals* de gewenste data aan de ALS toevoert. Ook is in een *memory address register* voorzien voor het adresseren van een plaats in het (hoofd)geheugen. Dit register, en het apart aangegeven AC-(reken)register, kan ook opgenomen gedacht worden in de register file. Een verdere uitbreiding van de bouwsteen kan nog zijn het naar buiten brengen van

signalen, die direct informatie leveren over de inhoud van het AC-register, zoals: inhoud $AC = 0$, waarde van het hoogstwaardige (teken)bit, enz.

De *function select* signalen op de ingangen van de bouwsteen in figuur 11.2 worden in de *function decoder* geklokt. Deze signalen worden gedecodeerd en er ontstaan besturingssignalen binnen de chip, die de functie van de ALS instellen, de multiplexers in de gewenste stand zetten, het juiste register in de file aanwijzen, de doorverbindingen maken, enz. Na enige (korte) tijd zijn alle signalen weer tot rust gekomen, overeenkomstig de nieuwe toestand. Nu kan in een volgende klokfase deze nieuwe toestand worden gefixeerd, de informatie extern verwerkt en (eventueel afhankelijk van het resultaat van de vorige bewerkingen) de volgende function select signalen worden ingesteld.

11.3. Overige bouwstenen

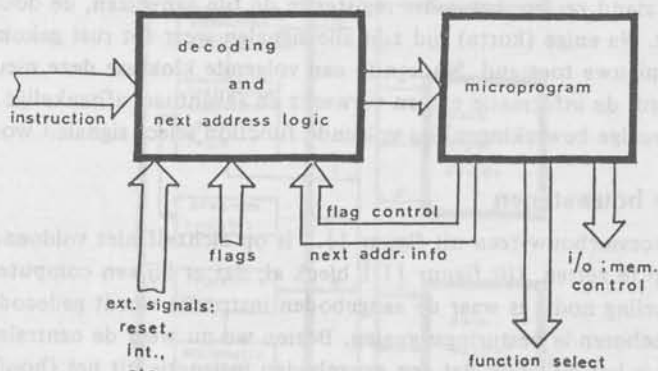
De centrale processorbouwsteen uit figuur 11.2 is op zichzelf niet voldoende om een systeem mee op te zetten. Uit figuur 11.1 bleek al, dat er bij een computer in ieder geval een schakeling nodig is waar de aangeboden instructie wordt gedecodeerd en omgezet in de bijbehorende besturingssignalen. Bezien we nu weer de centrale processorbouwsteen dan is het duidelijk dat een aangeboden instructie uit het (hoofd)geheugen van een computer niet door middel van een eenmalige instelling van de function select signalen te verwezenlijken is, immers zelfs een simpele instructieverwerking ziet er al uit als in figuur 11.3.



Figuur 11.3. Voorbeeld van een (simpele) instructieverwerking.

Dit vraagt al om meer dan één enkele toestand. Een systeem opgebouwd met centrale processorbouwstenen moet een organisatie hebben, die er voor zorgt dat er per aangeboden instructie een aantal opdrachten uitgevoerd wordt, die bij elkaar het gewenste effect sorteren. Dit aantal uit te voeren opdrachten (instructies) slaan we op in een geheugen. Er ontstaat dus binnen de computer een tweede besturingsniveau, zowel voor de hardware als voor de software. Om onderscheid te maken tussen beide niveaus noemen we het programma, dat de interne werking van de CPU bestuurt, het *microprogramma*.

De opdrachten, die samen het microprogramma vormen, heten *micro-instructies*, dit zijn bitpatronen die de werking van digitale elementen in een schakeling specificeren. Om de micro-instructies in de juiste volgorde uit te voeren hebben we een besturings-schakeling nodig. Tevens dient in deze schakeling, waarvan figuur 11.4 een voorbeeld is, de decodering van de aangeboden instructie uit het (hoofd)geheugen te gebeuren.



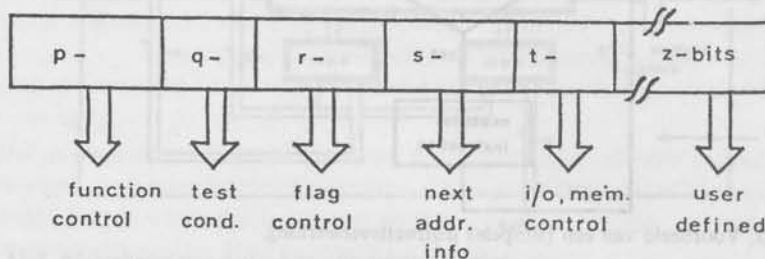
Figuur 11.4. Voorbeeld van een besturingsschakeling voor het uitvoeren van micro-instructies.

Verder is in figuur 11.4 nog voorzien in de volgende mogelijkheden:

- besturingssignalen naar I/O en/of geheugen,
- het zetten van vlaggen,
- het meegeven van informatie omtrent het volgende microprogramma-adres vanuit de geadresseerde micro-instructie,
- het instellen van de voorwaarde waarop getest gaat worden,
- aparte ingangen voor testsignalen,
- ingrepen van buitenaf; interrupt, reset, enz.

De instructie, die aan de schakeling wordt aangeboden, is van het 'gangbare' type (bijvoorbeeld $\text{ADD } A, R_p$ voor een optelling van de accumulator met register (p)) en wordt wel aangeduid met (macro)instructie, in tegenstelling tot de micro-instructie uit het microprogramma, opgeslagen in het besturingsgeheugen.

Een voorbeeld, hoe een micro-instructie er uitziet, geeft figuur 11.5. Hierin vinden we de bovengenoemde besturingssignalen en instellingen terug.



Figuur 11.5. Voorbeeld van een micro-instructie.

De realisatie van de hardware van een besturingsschakeling kan gebeuren met *wilde logica*, met behulp van PLA's of met speciaal ontwikkelde *control units* of *bit-slice* processor bouwstenen. De uitvoering met TTL-circuits voert in te veel details om hier te behandelen.

Een programmable logic array (PLA) is een circuit waarmee een aantal combinaties van ingangsvariabelen uitgecodeerd kunnen worden. Dit gaat door middel van een in

de chip opgenomen matrix, welke eenmalig programmeerbaar, verbindingen legt tussen ingangen van de chip en een aantal poortcircuits in de PLA. Een PLA kan circa 20 TTL-chips vervangen en leidt tot een aanzienlijke vereenvoudiging van de hardware opzet.

Enkele fabrikanten van bit-slice microprocessors leveren ook een, aan de processor-chip aangepaste, microprogramma-besturingseenheid of *sequencer*.

Hierin zijn latches, multiplexers, microprogrammateller en verdere, passende, combinatorische en sequentiële logica opgenomen. Afhankelijk van fabrikant en van uitvoering van de schakeling kan dan een totale decodering en besturing in één of meer chips ondergebracht worden. Het microprogramma zelf kan ondergebracht worden in een ROM of in een RAM. Bij gebruik van een ROM ligt het programma vast en is direct bij inschakelen gebruiksklaar. Is een RAM toegepast, dan moet na het inschakelen dit geheugen eerst worden geladen met het gewenste microprogramma. Tegenover dit, geringe, nadeel staat echter dat er nu, voor iedere toepassing een probleemgerichte instructieset te maken is en daardoor een efficiëntere programmering.

Van zowel ROM als RAM zijn talrijke typen beschikbaar, zodat altijd wel een geschikte keuze mogelijk is, voor wat betreft capaciteit, snelheid, datapadbreedte, enz.

Naast een bit-slice microprocessorbouwsteen en een sequencer brengen fabrikanten nog andere ondersteuningscircuits uit. Deze zijn voornamelijk gericht op het onderwerpen van problemen die de ontwerper tegenkomt bij implementatie. Als voorbeeld de volgende situatie.

Bij de uitvoering van bijvoorbeeld een optelopdracht kan het voorkomen dat een carry, die als gevolg van de bewerking ontstaat in de laagstwaardige sectie van de processor, tot gevolg heeft dat ook in een aantal, of alle, opvolgende secties eveneens een carry optreedt. Bij dit effect ontstaat een aanzienlijk tijdverlies, omdat steeds per sectie de doorlooptijd (carry ripple through time) afgewacht moet worden, voordat aan de uitgang een resultaat zichtbaar is. Aan dit bezwaar is tegemoet te komen door een zogenaamde *carry-look-ahead* (CLA) schakeling toe te passen. Dit circuit bepaalt direct uit speciaal op de processor uitgevoerde signalen en uit een eventueel inkomende carry of, en zo ja waar, een carry ontstaat tussen de processor-chips en zorgt voor een carry-in signaal op de juiste plaats. Verschillende fabrikanten leveren zo'n CLA-circuit bij een bit-slice microprocessor.

Een andere verbetering, die al door verschillende computerfabrikanten is toegepast, wordt bereikt door toepassing van een zogenaamd *pipe-line-register* (PLR). Bij het doorlopen van het microprogramma zal, bij iedere instructiefetch, dit een zekere tijd beslaan. Gedurende deze tijd kan er nog geen instructie-uitvoering plaatsvinden. Bij het uitvoeren van de instructie staat echter het instructiefetch-gedeelte van de besturing stil. In een PLR wordt nu tijdens de uitvoer van de ene instructie, de volgende instructie vast opgenomen. Nu zal een fetch van een instructie niets anders zijn dan de reactietijd van de uitgangen van de PLR op de klokpuls en dus aanzienlijk minder dan zonder de PLR. Helaas geldt ook hier de wet van behoud van ellende. Het pipe-line mechanisme werkt goed als steeds de volgende instructie een geheugenplaats verder staat dan de huidige instructie.

Worden veel spronginstructies toegepast, dan zal steeds het juist in het PLR gehaalde adres vervangen dienen te worden door het sprongadres. Dit kost dan steeds circa een halve cyclus extra. Ook het gebruik van conditionele spronginstructies levert problemen op; het resultaat van een bewerking, waarop getest moet worden, komt immers één cyclus vertraagd pas beschikbaar. Zo'n conditionele spronginstructie moet dus met extra cyclustijd vertraagd worden uitgevoerd of de vertraging moet door een zeer zorgvuldige instructievolgorde opgevangen worden.

Tenslotte kan nog vermeld worden, dat er door enkele fabrikanten nog circuits worden uitgebracht, die een aparte functie vervullen, die ook vrij simpel met bestaande TTL-bouwstenen gerealiseerd kan worden. Dit zijn onder andere timing circuits voor het genereren van een klokpuls, busbuffers, prioriteits interruptafhandeling, geheugen-interfaces, enzovoorts.

Tabel 11.1 geeft een overzicht van verkrijgbare bit-slice microprocessorbouwstenen met de belangrijkste eigenschappen en ook welke verdere bijbehorende circuits geleverd worden.

Fabrikant	Typenr.	Bits	Cycle time	Techn.	Package	Chip family
AMD	2901	4	50 ns	TTL	40p-DIL	2902, carry-look-ahead
	2903	4	50 ns	TTL	48p-DIL	2909, -10, -11 } microprogram
	29203	4	50 ns	TTL	48p-DIL	2930, -31, -32 } sequencer diverse andere circuits
Intel	3002	2	70 ns	TTL	28p-DIL	3001, sequencer 3003, carry-look-ahead diverse andere circuits
Fairchild	9405	4	100 ns	TTL	24p-DIL	9400-reeks: databuffers, stack, FIFO, etc.
Motorola	10800	4	55 ns	ECL	48p-QUL	10801, microprogram control
	10900	8	70 ns	ECL	64p-Q	10904, microcode sequencer
Texas Instr.	SBP0400	4	1000 ns	I ² L	40p-DIL	Tcy bij 150 mA injector stroom
	74S481	4	100 ns	TTL	48p-QIL	74S482, controller element
MMI	6701/02	4	200 ns	TTL	40p-DIL	6710, control unit diverse andere circuits

Tabel 11.1. Overzicht van verkrijgbare bit-slice microprocessorbouwstenen.

11.4. Ontwerp van een processoreenheid

De eerste stap die gedaan moet worden bij het ontwerp van een processor is het vastleggen van de gewenste specificaties, wat zijn de eisen ten aanzien van:

- datapadbreedte,
- gewenste aantal en soort registers,
- de interruptafhandeling,
- gewenste snelheid,
- power-on-restart faciliteit,
- soort en aantal te bedienen I/O-apparaten,
- enzovoorts.

Liggen al deze ontwerpsspecificaties vast dan kan, aan de hand daarvan, de hardware structuur worden vastgesteld. Welke aanpassingen gewenst zijn, welke organisatie nodig is in verband met bijvoorbeeld de snelheid, welke circuits, die de fabrikant levert, toepasbaar zijn, enzovoorts. Dit zijn allemaal factoren die bepaald en omgezet kunnen worden in een hardware realisatie.

Na vaststelling van de hardware is een volgende stap het vastleggen van de gewenste (macro)instructieset. De te ontwerpen processor is immers een uniek exemplaar, waarvan we zelf alles moeten specificeren. Zoals zal blijken, is het verstandig om de instructies per groep samen te beschouwen, zo zal er een onderscheid te maken zijn in onder andere:

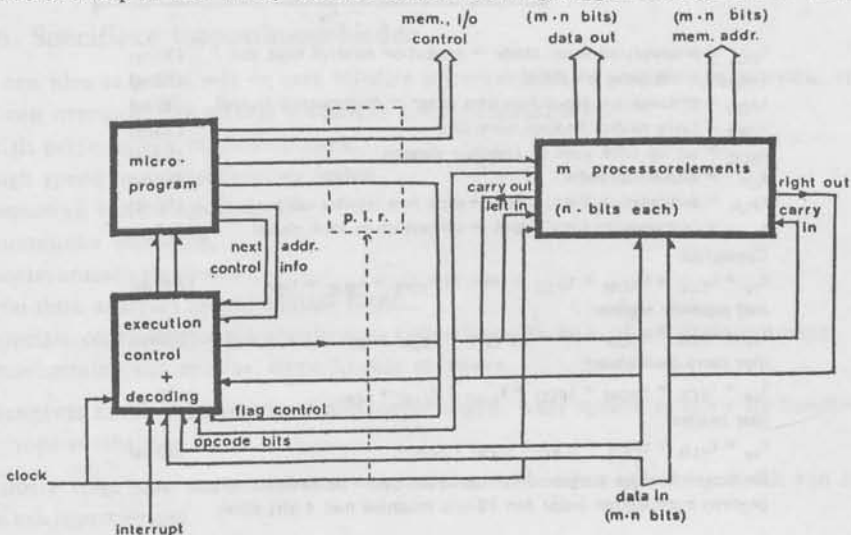
- een memory reference groep; bijvoorbeeld laden van data in een register,
- een immediate groep; bijvoorbeeld laden met data uit een (volgende) adres,
- een jump groep; alle (conditionele) spronginstructies,
- een input/output groep,
- een subroutine groep,
- enzovoorts.

Door de macro-instructies per groep bij elkaar te houden en er bij het toekennen van een operatiecode aan een instructie voor te zorgen, dat instructies uit een groep een gemeenschappelijk operatiecode-deel hebben, is tegelijkertijd een mogelijkheid ontstaan om in het microprogramma, per groep, een startadres van de, bij de groep behorende, afhandelingsroutine te hebben. In deze afhandelingsroutine kan dan verder bepaald worden, welke van de groep van instructies wordt bedoeld. Zo zal, na detectie van een spronginstructie, in de afhandelingsroutine bepaald kunnen worden of het een voorwaardelijke spronginstructie betreft en zo ja, op welke vlag getest moet worden. Bij instructies, die eenzelfde ALS-operatie tot doel hebben, zoals een ADA (add accu) en een ADI (add immediate) kan, na het ophalen van de operanden een gemeenschappelijke routine voor de optelling ontstaan.

Deze laatste afwegingen begeven zich al op het gebied van het implementeren van het microprogramma. Dit moet de laatste stap zijn in het ontwerp. Eén van de belangrijkste overwegingen betreft de afweging software/hardware. Hoewel in de tweede stap de hardware gedefinieerd is, wil dit niet zeggen, dat latere argumenten hierin geen wijziging kunnen veroorzaken. Bij het ontwerp en bij het microprogramma kan namelijk worden uitgegaan van een opzet met een smal micro-instructiewoord, daardoor minder hardware, maar per uit te voeren macro-instructie meer microprogrammastappen en een relatief langzame schakeling. Of de keuze kan vallen op een breed micro-instructiewoord, met meer hardware, kleinere afhandelingsroutines en relatief snel.

Deze twee benaderingen worden aangeduid met respectievelijk *verticale* en *horizontale* microprogrammering.

In de laatste fase, de implementatie van het microprogramma, zou kunnen blijken dat bijvoorbeeld niet aan de snelheidseis kan worden voldaan, dan moet, na een aanpassing van het ontwerp in fase twee, weer de laatste ontwerpstappen doorgenomen worden.

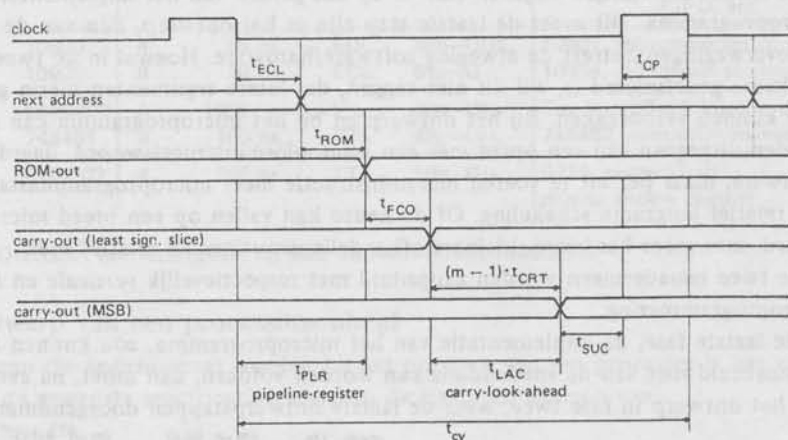


Figuur 11.6. Schematische opzet van een $m \times n$ -bits processor met pipe-line register (gestreept).

Figuur 11.6 geeft schematisch een opzet voor een $m \times n$ -bits processor met gestreept een pipe-line register. Het besturingsgedeelte is dat van figuur 11.4. Verder is een enkele interrupt opgenomen en zijn de verbindingen met het hoofdgeheugen aangegeven.

Hoewel in het bovengenoemde ontwerp uitgegaan is van een te realiseren processor, gelden dezelfde ontwerpstappen ook voor bijvoorbeeld een peripheral controller, met dien verstande dat in plaats van macro-instructies, opdrachten ontstaan die specifiek met het randapparaat te maken hebben en de nadruk zal vooral komen te liggen op I/O-operaties.

Tot slot nog een opmerking over de timing van het, met processeurelementen gerealiseerde systeem. Het tijddiagram van figuur 11.7 kan men zich denken te behoren bij het schema van figuur 11.6. Uitgaande van de klokpuls is steeds aangegeven welke reactie volgt op de voorgaande. Onderaan het tijddiagram is vermeld welk gedeelte van de timing verandert bij gebruik van respectievelijk een PLR en een CLA. Bij de, alleszins reëel, aangenomen delay-tijden, die tussen haakjes achter de diverse grootheden zijn aangegeven, resulteert dit in de eronder aangegeven cyclustijden. Dit is een micro-instructie cyclustijd, waaruit geen conclusie getrokken kan worden ten aanzien van de



t_{ECL} = propagation time: clock → execution control logic out (30 ns)

t_{ROM} = acces time van ROM (30 ns)

t_{FCO} = propagation time: function select → chip output (carry) (30 ns)

t_{CRT} = carry ripple through time/slice (30 ns)

t_{SUC} = set up time voor de conditie vlaggen (10 ns)

t_{CP} = klokpulsbreedte (15 ns)

t_{PLR} = propagation time: clock → pipe line register outputs (15 ns)

t_{LAC} = propagation time: input → output carry look ahead (20 ns)

Cyclustijd:

$t_{cy} = t_{ECL} + t_{ROM} + t_{FCO} + (m-1) \cdot t_{CRT} + t_{SUC} + t_{CP}$ (205 ns)

met pipeline register:

$t_{cy} = t_{PLR} + t_{FCO} + (m-1) \cdot t_{CRT} + t_{SUC} + t_{CP}$ (160 ns)

met carry-look-ahead:

$t_{cy} = t_{ECL} + t_{ROM} + t_{FCO} + t_{LAC} + t_{SUC} + t_{CP}$ (135 ns)

met beiden:

$t_{cy} = t_{PLR} + t_{FCO} + t_{LAC} + t_{SUC} + t_{CP}$ (90 ns)

De tussen haakjes aangegeven delay-tijden geven de overeenkomstig aangegeven cyclustijden (voor een 16-bits machine met 4-bits slice).

Figuur 11.7. Tijddiagram behorend bij figuur 11.6.

macro-instructie verwerkingstijd. Hebben we voor een macro-instructie bijvoorbeeld zes microcycli van 100ns elk nodig en daarbij nog tweemaal een leescyclus van het (hoofd-) geheugen met een toegangstijd van 400ns, dan levert dit een instructie-verwerkingstijd van totaal 1,4 μ s op.

11.5. Hulpmiddelen bij het ontwerp

Net zoals bij de microprocessors een zekere hoeveelheid ondersteuningsprogrammatuur geleverd wordt, is dit ook het geval bij de bit-slice microprocessors. De mogelijkheden voor deze programmatuur zijn echter beperkter, omdat er slechts uitgegaan kan worden van die elementen, welke bij iedere gebruiker dezelfde functie hebben. Dit zijn dus de processorelementen en de microprogramma-besturingschip. Van beide elementen zijn aan de uit te voeren functies, bijbehorende mnemonische codes toegevoegd, die met een micro-assembler te vertalen zijn in micro-instructies. Tevens geeft zo'n programma extra mogelijkheden, zoals gebruik van symbolische adressen, definiëring van speciale velden en een listing van een microprogramma met daarbij binaire code en foutmelding.

Een ontwikkeling, die al bekend is van de 'gewone' microprocessors, is ook voor deze circuits de *in-circuit-emulator*. Hierbij gaat men uit van een processorsysteem, dat via een speciale kabel, waarin een aanpassingsschakeling, aangesloten wordt op de plaats, waar in de te testen schakeling de microprogramma-besturingschip zit. Het processor-systeem neemt dan de functies van deze chip over en met speciaal bijgeleverde software geeft dit faciliteiten als:

- onderbreking van het microprogramma; opvragen en eventueel wijzigen van registers en status,
- aangeven van startadres voor programma-executie,
- instellen van breakpoints,
- volgen en uitprinten van de doorlopen adressen,
- testen op logische niveaus (TTL),
- single step,
- wijzigen en uitvoeren van microprogramma's die in het microprocessorsysteem gesimuleerd worden.

11.6. Specifieke toepassingsgebieden

Om een idee te geven wat er met bit-slice microprocessors gedaan kan worden, volgt hier een overzicht van enkele specifieke toepassingsgebieden:

- high performance minicomputers,
- high speed instrumentatie en testen,
- emuleren van computers,
- numerieke besturing,
- communicatieprocessors,
- real time analyse; radar, digitale filters,
- speciale centrale processoreenheden, respectievelijk I/O- of periferiebesturing,
- bescherming van speciaal ontwikkelde software.

Samengevat komt het neer op al die toepassingen, waar sprake is van data handling met hoge snelheid.

Tenslotte volgt hier een aantal voor- en nadelen verbonden aan het gebruik van bit-slice microprocessors.

■ Voordelen

- De opzet wordt optimaal, omdat aan specifieke eisen (voor wat betreft de software, snelheid, datapad, organisatie) tegemoet is gekomen.
- Door wijziging van het microprogramma is ook andere software te verwerken (emulatie).
- Het ontwikkelde systeem is relatief simpel uit te breiden of aan te passen aan specifieke gebruikersbehoefte.
- De hardware opzet vereist minder chips dan met zogenaamde wilde logica het geval zou zijn, daardoor ook minder vermogen.

■ Nadelen

- Het is een uniek systeem; zelf moet voor documentatie gezorgd worden.
- Er is geen direct hardware vergelijk met een tweede systeem mogelijk; vooral in de uittestfase zal dit van belang zijn.
- Alle software zal zelf ontwikkeld moeten worden; ook assemblers, monitors, editors, enzovoorts.
- Het vereist veel ontwerptijd; de cyclus specificatie—hardware—microprogramma zal meermalen doorlopen worden.

12. Parallel processing

12.1. Inleiding

Hoewel het de bedoeling is, in dit hoofdstuk in te gaan op aspecten van parallel processing met microcomputers, zijn veel van die aspecten evenzo geldig voor mini- of maxicomputers. Immers, parallel processing is een functionele aanduiding voor een systeemopzet en is niet direct component-gebonden.

Mede daarom zullen eerst een aantal algemene aspecten aan de orde komen en kan daarna wat meer aandacht besteed worden aan de microcomputer.

12.2. Redenen voor parallel processing

Er zijn een aantal gevallen aan te geven, waarin parallel processing duidelijk voordelen biedt of zelfs de enige, praktische realisatie oplevert. Als voorbeeld de volgende gevallen.

- Er blijven altijd problemen, die een dusdanig grote hoeveelheid *processing power* vergen, dat geen bestaande, enkele, processor die problemen aan kan. Veelal wordt dan van nog snellere circuits gebruik gemaakt, maar ook hieraan komt eens een eind.
Een andere oplossing kan zijn het toepassen van speciale organisaties, zoals parallel processing.
- Bepaalde processen bezitten een soort intrinsiek parallellisme. In het proces of de bewerking zelf zijn verschillende, parallel lopende delen te onderscheiden, welke daardoor uitermate geschikt zijn voor parallel processing. Te denken is hierbij aan onder andere matrixmanipulaties, beeld- en andere signaalverwerking.
- Vooral bij in-line computertoepassingen worden hoge eisen gesteld aan bedrijfszekerheid. Een opzet met een aantal, parallel werkende processors, waarbij in geval van defect raken van één exemplaar, de overigen de taak overnemen resulteert in dat geval in een vermindering van de totale capaciteit, maar niet in uitval van de gehele installatie.
- Ook economische redenen kunnen een zeer belangrijke rol spelen. Voor een bepaald systeem zullen de kosten van de software vaak veel hoger uitvallen dan de kosten voor de hardware. Dan is het duidelijk dat gezocht moet worden naar middelen om die software te vervangen door hardware. Als voorbeeld bezien we de PDP-11 serie van Digital Equipment Corporation. Deze serie loopt van het type PDP-11/03 tot de PDP-11/70, met toenemende capaciteit. Ieder type uit deze serie heeft zijn eigen, vaak specifieke, hardware opzet, dus aparte ontwikkeling en productie. Het zou veel voordeliger zijn als een systeem gerealiseerd kan worden op basis van een kwantitatieve modulariteit, zodat de computercapaciteit voor een bepaalde toepassing te bereiken is door een aantal identieke eenheden, zonder de technologie te veranderen. Dit levert voordeel op bij de fabricage, in de voorraadvorming (ook voor reserve-onderdelen) en het personeel in onderhouds- en reparatie-afdelingen zal op een standaard type processor sneller ingewerkt zijn.

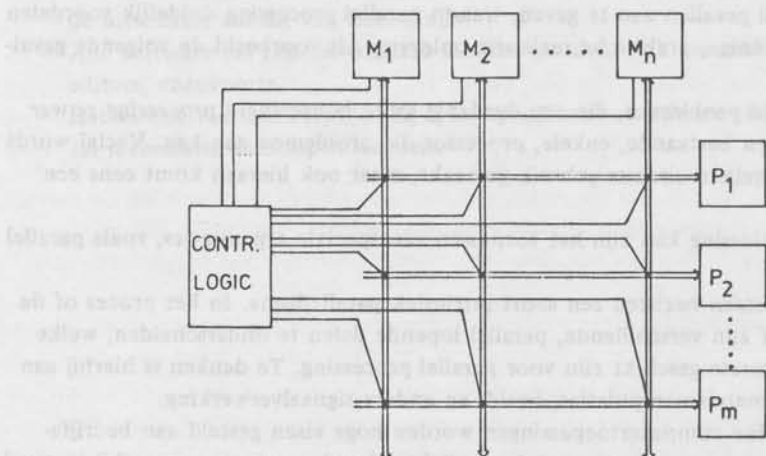
In alle bovengenoemde omstandigheden zou een opzet van een systeem met parallel werkende processors technisch en/of economisch voordeel opleveren ten opzichte van een enkel processorsysteem.

12.3. Hardware structuren voor parallel processing

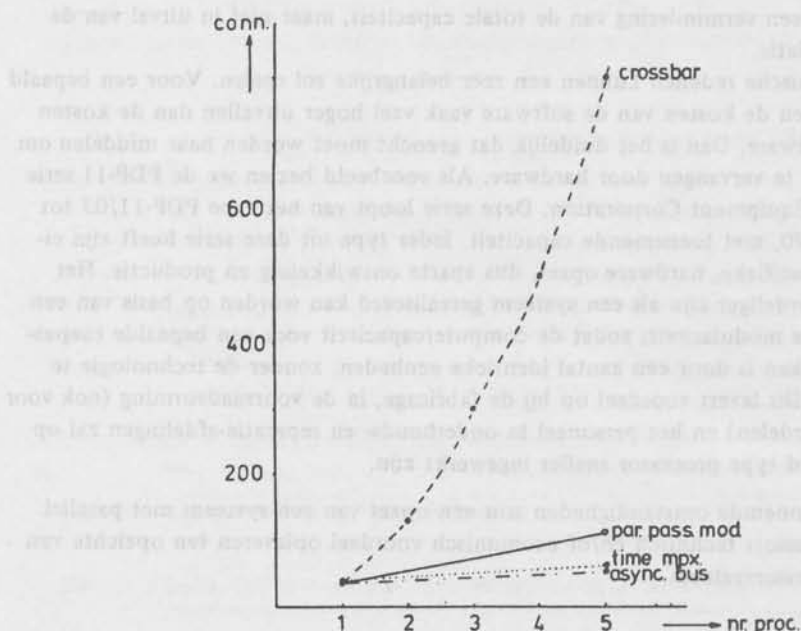
In theorie zijn er net zoveel interessante hardware structuren voor parallel processing te bedenken als iemands fantasie toelaat. Of ieder van die structuren dan ook praktisch bruikbaar is, is een tweede. Hier zullen we ons beperken tot een aantal structuren en verbindingstechnieken die realiseerbaar gebleken zijn. De structuren zijn te verdelen in twee gebieden, te weten:

- de direct gekoppelde systemen én
- netwerken.

Direct gekoppelde systemen zijn weer onder te verdelen, namelijk ten eerste in de systemen waarin doorverbindingen gemaakt worden met een zogenaamde *crossbar*-schakelaar en ten tweede de bus-georiënteerde systemen.



Figuur 12.1. Direct gekoppeld systeem met crossbar-schakelaar.

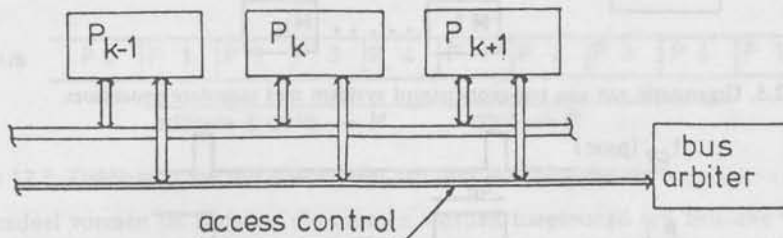


Figuur 12.2. Aantal benodigde verbindingen voor een 8-bits datasysteem.

Eerstgenoemde systemen, waarvan figuur 12.1 een voorbeeld geeft, hebben als voordeel dat een zeer flexibel en eenvoudig uitbreidbaar netwerk gerealiseerd is maar de kosten voor de verbindingen groeien de pan uit! Het bijbehorende verbindingssysteem wordt ook wel aangeduid als *space-multiplexed*.

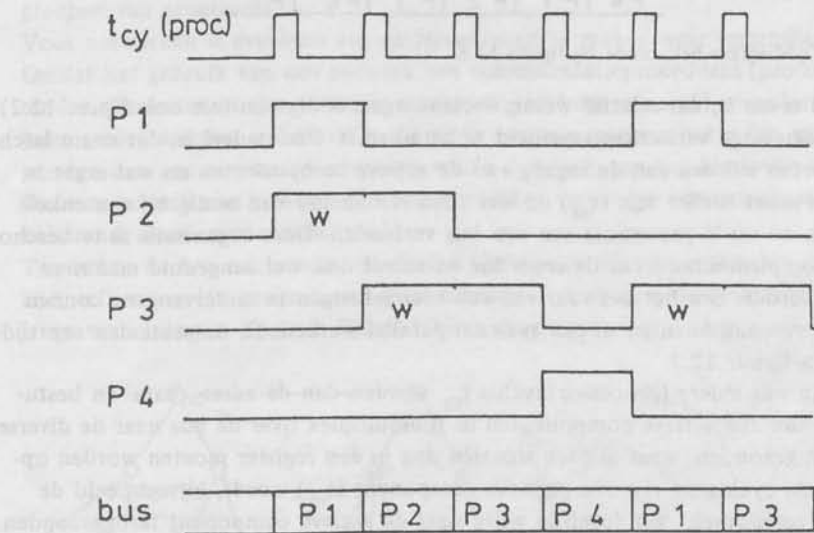
Figuur 12.2 geeft het verband tussen het aantal benodigde verbindingen en het aantal actieve en passieve componenten voor een 8-bits datasysteem bij toepassing van (onder andere) een crossbar-schakelaar.

De bus-georiënteerde systemen zijn weer onder te verdelen in een aantal klassen. Allereerst, het recht-toe-recht-aan bussysteem: asynchroon (eventueel met prioriteitscontrole). Een aantal actieve en passieve componenten is dan via een bus onderling verbonden en iedere eenheid die over de bus wil beschikken, doet hiertoe een busaanvraag. Na fiatte-ring kan dan het gewenste transport plaatsvinden waarna de bus weer wordt vrijgegeven. Deze organisatie is weergegeven in figuur 12.3.



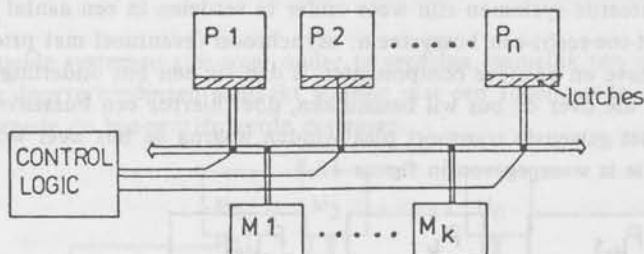
Figuur 12.3. Organisatie van een asynchroon bus-georiënteerd systeem.

Het voordeel is een eenvoudige verbinding, maar als er meer dan 3 à 4 processors aan een bus hangen gaat de efficiëntie van iedere processor achteruit omdat de wachttijden voor bustoegang groeien. Figuur 12.2 geeft een globale indruk van het aantal verbindingen tegenover het aantal componenten voor een aantal typen multiprocessorsystemen. Figuur 12.4 geeft de timing van een asynchroon bus-georiënteerd systeem. Ook is het ontstaan van de wachttijden (w) aangegeven. Tot aan een viertal processors toe, kan een rendement van 80 à 90% per processor gehaald worden.

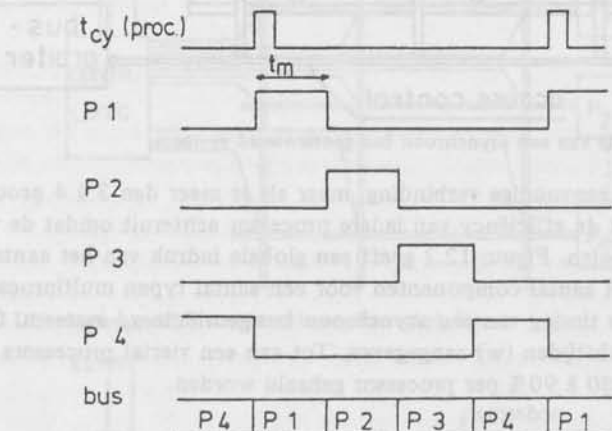


Figuur 12.4. Timing behorende bij figuur 12.3.

Meer processors zijn aan een bus te hangen in de opzet van figuur 12.5. Iedere cyclus van een actieve component is onderverdeeld in een aantal tijdseenheden. Alleen in de bijbehorende tijdseenheid heeft een actieve component toegang tot de bus om met een passieve component te communiceren. Aan de ingang van iedere actieve component moeten dan latches (registers) geplaatst worden om de data in op te slaan. Zo is na een cyclus t_{cy} voor iedere actieve component de data beschikbaar. In figuur 12.6 is een tijddiagram weergegeven dat correspondeert met deze opzet.



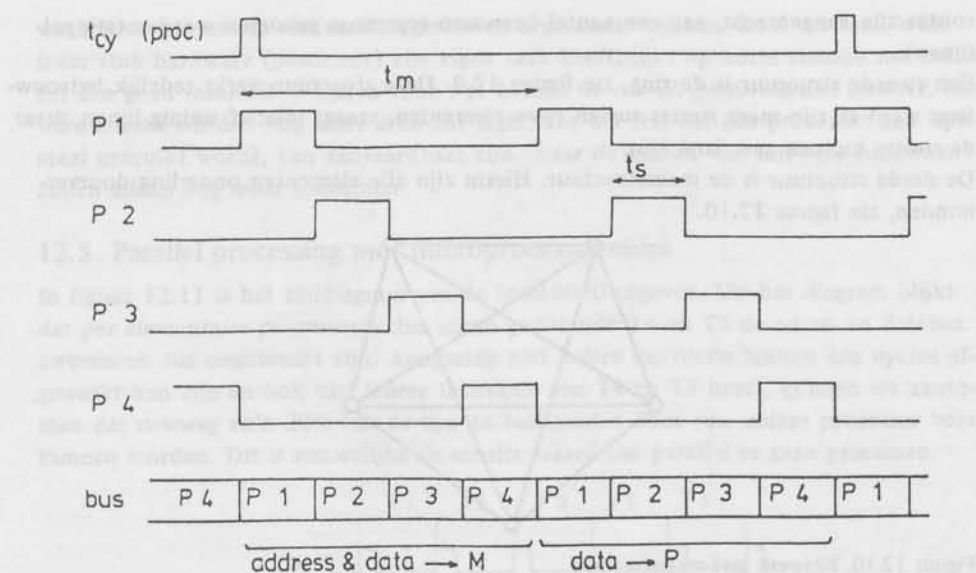
Figuur 12.5. Organisatie van een bus-georiënteerd systeem met meerdere processors.



Figuur 12.6. Tijddiagram behorende bij figuur 12.5.

Het voordeel ervan is, dat relatief weinig verbindingen nodig zijn (zie ook figuur 12.2) terwijl toch een hoge verwerkingssnelheid te bereiken is. Een nadeel is, dat er nu latches geplaatst moeten worden aan de ingang van de actieve componenten en wat erger is, het geheugen moet sneller zijn (t_m) en wel circa $N \times$ sneller dan nodig bij een enkele processor, als we nu N processors aan een bus verbinden. Deze organisatie is te beschouwen als de complementaire van de *cross bar* en wordt ook wel aangeduid met *time multiplex* structuur. Om het bezwaar van een snel geheugen te ondervangen, kunnen ook de passieve componenten in een systeem parallel werken. Er ontstaat dan een tijddiagram als in figuur 12.7.

Aan het begin van iedere (processor)cyclustijd t_{cy} worden dan de adres-, data- en besturingssignalen van alle actieve componenten in tijdmultiplex over de bus naar de diverse componenten gezonden, waar al deze signalen dan in een register moeten worden opgeslagen. Na de cyclustijd van een passieve component (t_m) wordt, bijvoorbeeld de data bij een leesopdracht, op dezelfde wijze naar de actieve component teruggezonden. Het bereikte voordeel is een, toch nog eenvoudig verbindingssysteem met meer draden dan in het vorige geval maar met geheugens met een, aan de processor aangepaste, snelheid.



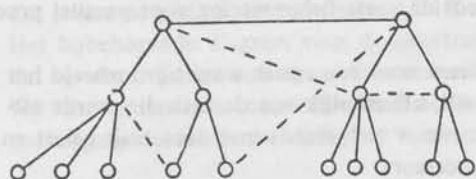
Figuur 12.7. Tijddiagram van een systeem met een time multiplex structuur.

Een nadeel vormen de registers die moeten worden toegevoegd ten behoeve van de opslag van de adressen, data en besturingsinformatie. Figuur 12.2 geeft een indruk van de, met dit systeem gepaard gaande, hoeveelheid verbindingen.

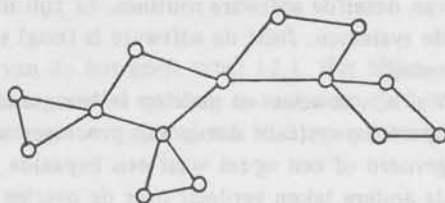
Als de latches (registers) bij iedere passieve component worden uitgebreid tot een aantal registers, dan hebben we een zogenaamd *cache*-geheugen gekregen, een snel tussen-geheugen tussen processor en geheugen. Dit resulteert in minder, maar wel langer durende, bustoegangen. In de praktijk blijkt vaak dat allerlei structuren zichzelf begrenzen tot 5 à 15 processors en wel door het toenemende aantal verbindingen en/of door de toe te voegen hardware faciliteiten of door een teruglopende efficiëntie van de processors. Nog complexere structuren worden dan gevormd door, al of niet hiërarchische, groepen van processors.

Voor netwerken is eveneens een onderverdeling te maken naar verschillende structuren. Omdat het gebruik van een netwerk om communicatieprocedures (protocollen) vraagt en omdat elementen in een netwerk, eerder als een (min of meer) compleet computersysteem gezien worden, zal hierbij toch meer aan een minicomputer (en groter) gedacht worden dan aan een microcomputer, welke dichter tegen het hardware niveau aanzit. Daarom zal volstaan worden met het vermelden van enige netwerkstructuren en de bijbehorende voor- en nadelen.

Ten eerste de boomstructuur, zie figuur 12.8, waar vrij veel verbindingslijnen bij nodig zijn, er is maar één route tussen twee elementen en tussen de eindelementen zitten lange verbindingen. Zoals in figuur 12.8 is aangegeven kan met een boom waarin dwars-



Figuur 12.8. Netwerk met boomstructuur.

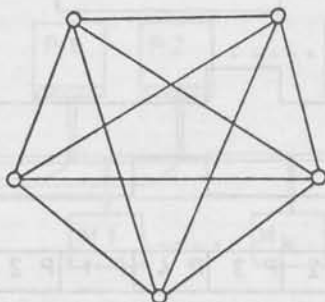


Figuur 12.9. Netwerk met ringstructuur.

routes zijn aangebracht, aan een aantal bezwaren tegemoet gekomen worden (stippellijnen).

Een tweede structuur is de ring, zie figuur 12.9. Deze structuur werkt redelijk betrouwbaar want er zijn meer routes tussen twee elementen, vraagt relatief weinig lijnen, maar de routes kunnen zeer lang zijn.

De derde structuur is de maasstructuur. Hierin zijn alle elementen onderling doorverbonden, zie figuur 12.10.



Figuur 12.10. Netwerk met maasstructuur.

Voordelen zijn korte routes, een betrouwbaar netwerk want er zijn veel routes tussen twee elementen, maar wel zijn er veel verbindingen nodig. Vaak wordt dan ook een gedegenererde maasstructuur toegepast met minstens een 2-voudige verbinding tussen iedere twee componenten.

Zoals al eerder genoemd zijn dit structuren, die eerder bij grotere computersystemen voorkomen en hier willen we ons beperken tot die organisaties, die toepasbaar zijn voor microcomputers, zodat netwerken hier verder onbesproken blijven.

12.4. Software implicaties

Ook al kan voor iedere toepassing een optimale hardware configuratie ontworpen worden, zal dit toch inhouden dat dan nog iedere keer ontwerp- en produktiekosten gemaakt moeten worden. Daarom wordt een hardware organisatie gekozen die in (vrijwel) alle gevallen zal voldoen. Dit houdt dan tevens in dat maar éénmalig software voor dit systeem ontwikkeld hoeft te worden. Zoals in de vorige paragraaf bleek, zijn er diverse structuren mogelijk. De netwerken zullen, per element, een lokaal operating-systeem hebben, met speciaal voor de communicatie via het netwerk toegesneden protocol software. Hier ligt de nadruk voor wat betreft de speciaal te ontwikkelen software, meer op software voor het aansturen van de verbindingen dan op het niveau van een operating-systeem. De meer direct gekoppelde systemen leveren een heel arsenaal van problemen op. Rekening dient onder andere gehouden te worden met communicatie tussen de diverse processors, bescherming van data, en de problemen die zich voordoen als verschillende processors gebruik maken van dezelfde I/O-apparatuur en van dezelfde software routines. Er zijn nauwelijks ervaringen opgedaan met gerealiseerde systemen. Juist de software is (nog) steeds de grote belemmering voor parallel processing.

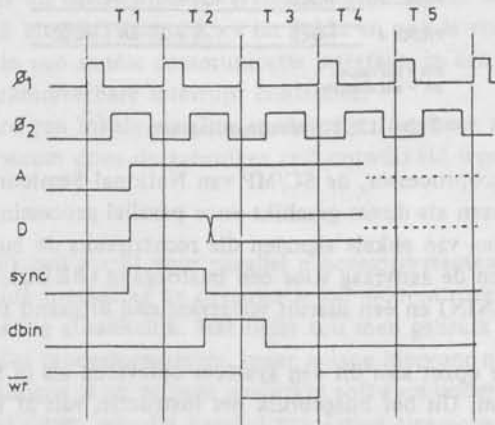
Wel zijn er voor- en nadelen te beargumenteren voor een opzet waar bijvoorbeeld het operating-systeem steeds van processor wisselt, afhankelijk van de taak die wordt uitgevoerd of een opzet waar een bepaalde processor zich steeds met deze taak belast en de andere taken verdeelt over de overige processors.

Deze laatste organisatie gaat al meer de richting op van een parallel processorsysteem

waarbij de processors een taak-toegesneden organisatie hebben. Zo'n systeem, waar ieder stuk hardware (processor) zijn eigen taak heeft, lijkt op korte termijn het snelst tot een goed resultaat te leiden voor wat betreft de *overall performance*. Immers, hardware kosten worden nog ieder moment lager, dus het feit dat een processor niet optimaal gebruikt wordt, kan aanvaardbaar zijn, maar de kosten van software (uurlonen) zullen alleen nog maar toenemen.

12.5. Parallel processing met microprocessor-chips

In figuur 12.11 is het tijddiagram van de Intel 8080 gegeven. Uit het diagram blijkt dat per elementaire processorcyclus alleen gedurende T4 en T5 de adres- en databus zweven en dus ongebruikt zijn. Aangezien niet iedere instructie binnen een cyclus afgewerkt kan zijn en ook niet iedere instructie een T4 en T5 heeft, kunnen we aannemen dat ruwweg zo'n 20% van de tijd de bussignalen door een andere processor bezet kunnen worden. Dit is nauwelijks de moeite waard om parallel te gaan processen.



Figuur 12.11. Tijddiagram van de Intel 8080.

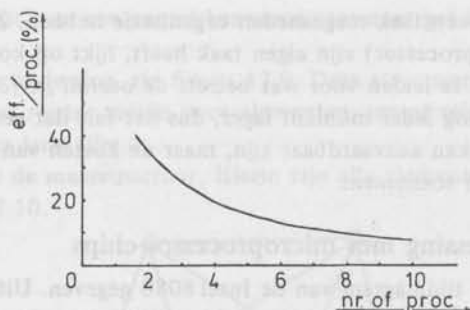
De Intel 8080 heeft ook een 'HOLD-request'-ingang, via deze aansluiting kunnen we de processor alle acties laten stopzetten, zodat de externe aansluitingspunten vrij zijn voor ander gebruik dan door de processor zelf.

Na de herkenning van zo'n 'HOLD-request' zal de Intel 8080 in de 'HOLD'-toestand komen en daarin net zolang blijven als het requestsignaal blijft staan. Dan blijkt dat aan begin en eind van iedere HOLD-sequence circa 1 μsec verloren gaat. Rekenen we met een gemiddelde instructietijd van 3,5 μsec en plaatsen we N processors op een bus dan is het rendement per processor aan te geven met

$$n = \frac{t_{\text{instr}}}{N t_{\text{instr}} + t_{\text{HD}}}$$

Dit levert de grafiek op van figuur 12.12.

Het bijbehorende diagram voor de bezetting van de bus geeft tabel 12.1. Het blijkt dat op deze manier ook nauwelijks een efficiënt parallel systeem te realiseren valt. We moeten dan over naar structuren zoals aangegeven in paragraaf 12.3, waarvoor een toevoeging van hardware nodig is.



Figuur 12.12. Rendement per processor indien meerdere processors zijn aangesloten op één bus.

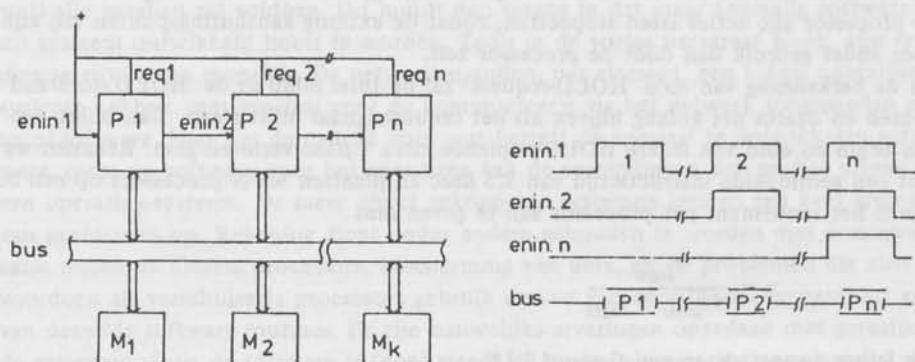
PROC. 1	RQ	BA	HOLD	RQ	BA
PROC. 2	HOLD	RQ	BA	HOLD	RQ
PROC. 3	HOLD	RQ	BA	HOLD	RQ
PROC. 4	HOLD	RQ	BA	HOLD	RQ

RQ = bus request
BA = bus available

Tabel 12.1. Bezettingsdiagram.

Een ander type microprocessor, de SC/MP van National Semiconductor Corporation, wordt wel aangeprezen als direct geschikt voor parallel processing op chipniveau. De SC/MP is voorzien van enkele signalen die rechtstreeks de buscontrole verzorgen. Deze signalen regelen de aanvraag voor een bustoegang (BREX), een binnenkomend besturingssignaal (ENIN) en een hieruit voortvloeiend uitgaand toestemmingssignaal (ENOUT).

In de meest simpele opzet kan dit een systeem opleveren als in figuur 12.13, met bijbehorend tijddiagram. Uit het busgebruik per instructie, valt af te leiden dat circa 30% van de tijd die de totale instructieverwerking beslaat, de bus bezet is. Dit houdt in dat 3 SC/MP's op één bus zouden kunnen werken zonder noemenswaardig verlies in processing capaciteit.



Figuur 12.13. Opzet van de SC/MP met bijbehorend tijddiagram.

Worden meer processors aangesloten, dan levert dit problemen op, ook nog omdat in deze configuratie, door het verbinden van ENIN met de voorgaande ENOUT de prioriteit bepaald is door deze doorverbinding; een processor achterin de keten komt zelden aan de beurt. Dan moet dus een ander prioriteitsmechanisme worden toegepast.

Ook hier blijkt dus al vrij snel dat andere structuren nodig zijn om wat meer effectiviteit te behalen.

12.6. Parallel processing met boards

Van de verschillende *single computer boards* (dit is een printkaart waarop een processor, geheugen en I/O-faciliteiten zijn ondergebracht) is er op dit moment één verkrijgbaar die speciaal ontwikkeld is voor parallel processing. Dit is het 80/20-systeem van onder andere Intel. Een board dat een zogenaamde Multibus aansluiting heeft, zie ook hoofdstuk 5. Zonder in te gaan op alle specificaties van dit board moeten er in dit verband toch wel enkele genoemd worden. In ieder geval zijn er signalen die voor de bus-arbitrage zorgen. Een 'request'- en een 'busy'-signaal alsmede een 'priority in'- en een 'priority out'-signalering, deze laatste twee voor een seriële prioriteit bij directe verbinding of voor een extern toe te voegen prioriteitsdecoder.

Via de databus is een transportsnelheid van 8 Mbyte per seconde te halen. Het gaat hier wel om een kaart die op zich al een systeem vormt. Op de kaart is namelijk plaats voor maximaal 4 K $\times$ 8 bit PROM en 2 K $\times$ 8 bit RAM en ook is voorzien in 48 programmeerbare I/O-lijnen, in een seriële communicatie interface, in een programmeerbare timer en in een programmeerbare interrupt controller.

Hoewel voor dit board een lokale monitor te verkrijgen is, moet alle software voor een multiprocessorsysteem door de gebruiker zelf ontwikkeld worden.

12.7. Conclusie

Er bestaat wel degelijk een markt voor parallel processorsystemen, echter de manier waarop in een bepaalde toepassing de organisatie zal moeten plaatsvinden is nu nog sterk van deze toepassing afhankelijk. Het liefst zou men gebruik maken van een general-purpose parallel processorsysteem, maar zolang hiervoor nog geen geschikte processortype te verkrijgen is en er geen geschikte software ondersteuning bestaat zal het nog wel enige tijd duren voordat parallel processing algemeen toepassing zal vinden, anders dan in systemen met taakgebonden processors.

Uitgaven van de Delftse Uitgevers Maatschappij op het terrein van de elektrotechniek en computerkunde:

DIGITALE TECHNIEK

van problemspecificatie tot realisatie
door ir. A.P. Thijssen, ir. H.A. Vink en
prof. ir. C.H. Eversdijk

In twee delen wordt een overzicht gegeven van de digitale (schakel-) techniek. De boeken worden gebruikt bij de colleges *Digitale Techniek I, II* en *III* voor studenten Elektrotechniek en Informatica.

Deel 1 behandelt de basiskennis van de componenten, waaronder: schakelalgebra, Karnaugh-diagrammen, SSI/MSI en LSI combinatorische schakelingen, flip-flops, schuifregisters en tellers. Veel aandacht wordt besteed aan het niet-ideale gedrag van de bouwstenen, waaronder de timing van de signalen.

Deel 2 behandelt het ontwerpen van grotere schakelingen, gebaseerd op het datapad-besturing ontwerpmodel. De specificatie en realisatie van besturingen neemt een groot stuk van dit deel in beslag. Daarnaast komen ook onderwerpen als microprogrammering, de afweging van hard/software oplossingen, testbaarheid, betrouwbaarheid en synchronisatie ruimschoots aan bod.

Vele opgaven en literatuurverwijzingen zijn toegevoegd. Uitwerkingen van opgaven zijn in beide delen opgenomen.

De systematische behandeling maakt de boeken ook uitstekend geschikt voor zelfstudie en voor gebruik op HTS en HIO.

De stof is zo geordend dat een eerste, meer globale, oriëntatie op het vakgebied eruit gedocerd kan worden, waarna desgewenst één of meer onderwerpen kunnen worden uitgewerkt. Dit laatste vooral voor de richting Elektrotechniek.

Deel 1, 368 pag., ISBN 90-6562-007-9

Deel 2, 160 pag., ISBN 90-6562-014-1
(herziening deel 2 verschijnt in 1985)

Voor docenten zijn lay-outs van *overhead-sheets* (ca. 300 stuks) verkrijgbaar. Voor nadere inlichtingen wende men zich tot de uitgever.

DIGITEST

Opgaven behorend bij *Digitale Techniek*

De in deze bundel opgenomen vraagstukken hebben in de jaren 1982 en 1983 deel uitgemaakt van de examens *Digitale Techniek* aan de TH-Delft. De vraagstukken zijn geordend naar onderwerp, overeenkomstig de hoofdstukindeling van het theorieboek. De vragen zijn

gesteld in multiple choice vorm. De antwoorden zijn op de laatste bladzijde weergegeven.

80 pag., ISBN 90-6562-046-x

MICROPROCESSORS

door ir. C.J. van Spronsen en ir. F. Bruggeman

In dit boek wordt de microprocessor geïntroduceerd, uitgaande van een algemeen model, aan de hand waarvan de interne opbouw en de instructie-afhandeling worden toegelicht. De diverse methoden van adressering, zoals microprocessors die kennen, worden behandeld. Uitgebreide aandacht krijgt de in- en uitvoer, alsmede de speciaal daarvoor ontwikkelde circuits. Standaards voor zowel aansluiting met de 'buitenwereld' als voor de koppeling van modules tot een (multi-)processor-systeem komen aan de orde.

Ook de programmatuur wordt niet vergeten, zoals blijkt uit de hoofdstukken over de ontwikkeling van software en de daarvoor bestaande ondersteuning in apparatuur en programmatuur.

Bij de behandeling van de stof in dit boek wordt aangenomen dat de lezer een (beperkte) kennis heeft van digitale technieken.

160 pag., ISBN 90-6562-034-6

COMPUTERARCHITECTUUR

door prof.dr.ir. A.J. van de Goor en
ir. H.A. Spanjersberg

Computerarchitectuur behandelt achtereenvolgens: begrippen en definities, machine-niveaus, representatie van data, machinetalen, adressering van data, dataoperaties, instructievolgorde, invoer en uitvoer van data en geheugenbeheer.

244 pag., ISBN 90-6562-025-7

ELEKTRONISCHE VERSTERKERS EN PHASELOCK LOOP

door prof.dr.ir. J. Davidse

Deze handleiding omvat een aantal onderwerpen die aan de orde komen in de colleges in de electronica aan de TH-Delft. De behandelde stof heeft betrekking op de moderne versterkertechniek en op de inrichting en toepassing van de zgn. phaselock loop.

Hoewel de moderne electronica in toene-

mende mate gebruik maakt van digitale technieken voor de bewerking van elektronische signalen, blijft analoge signaalversterking onmisbaar bij de bron van de signalen. In het bijzonder op deze plaats in de signaalketen is ruis de grootste vijand van de correcte overdracht van het signaal. Om deze reden is aan dit aspect relatief veel aandacht besteed.

De phaselock loop is een schakeling die vooral van groot belang is in de moderne communicatietechniek.

232 pag., ISBN 90-6562-021-4

ELEKTROTECHNISCH METEN door prof.dr.ir. K.B. Klaassen

De elektrotechniek en met name de elektronica, heeft ons vele hulpmiddelen voor het meten opgeleverd. Daarnaast vormt de elektrotechniek ook zelf een uitgebreid toepassingsgebied voor het meten.

Dit boek over elektrotechnisch meten stelt daarom beide categorieën aan de orde, dus zowel de elektrotechnische meetmiddelen als elektrotechnische metingen.

Het doel van het boek is een zodanige behandeling van de *grondslagen* van het meten, dat, naast het verstrekken van de benodigde *basiskennis*, het *inzicht* in het meten wordt bevorderd. Het einddoel daarbij is het zelfstandig kunnen *oplossen* van allerlei *meetproblemen*.

In dit boek is de aanpak van de vanouds sterk mechanisch en energie-technisch georiënteerde elektrotechnische meettechniek verlaten ten gunste van een meer systeem-technisch georiënteerde aanpak.

Na een korte samenvatting van de fundamenteën van het vak (*meettheorie*) blijkt het voor kwantitatief meten nodig te zijn eenheden af te spreken (*eenhedenstelsels*).

Om dit optimaal te kunnen meten, dit is: met de geringst mogelijke inspanning het gestelde doel bereiken, wordt een aantal alternatieve *meetmethoden* besproken.

Een meting verschaft slechts een eindige zekerheid; er worden altijd fouten gemaakt.

De *foutentheorie* bespreekt daarom de *soorten fouten*, *foutenvoortplanting* en *foutenoorzaken*.

Een meetsysteem heeft een bepaalde *structuur* waarin verschillende functies voorkomen (*transductie*, *signaalbewerking*, *indicatie* en *registratie*).

Deze meetfuncties worden in het hoofdstuk *elektrotechnische meetmiddelen* besproken. Daarna komen de volledig *elektronische meetsystemen* aan de orde. Hierbij is een belangrijke plaats ingeruimd voor het auto-

matisch meten met de computer, waarbij belangrijke zaken als *bemonstering*, *multiplexing* en *aliasing* aan de orde komen.

320 pag., ISBN 90-6562-033-8

ELEKTRISCHE NETWERKEN door ir. A. Henderson

Dit boek is in eerste instantie opgezet als dictaat van het college Elektrische Netwerken voor de eerstejaarsstudenten aan de afdeling Elektrotechniek van de TH-Delft. De aanpak bleek echter al ras ook docenten aan de HTS-en aan te spreken.

Na een bondige uiteenzetting van de basisbegrippen worden behandeld de netwerkstellingen, bestuurd bronnen, wisselstromen en wisselspanningen, complexe grootheden en enkele eigenschappen van netwerken. Daarna komen aan de orde gekoppelde spoelen en (ideale) transformatoren, driefasensystemen, complexe frequentie, Fourieranalyse en schakelverschijnselen. Het boek wordt afgesloten met een hoofdstuk over computergerichte analyse.

340 pag., ISBN 90-6562-044-4

VRAAGSTUKKEN ELEKTRISCHE NETWERKEN I

door ir. A. Henderson

De stof in deze bundel is verdeeld in hoofdstukken, die parallel lopen met het boek Elektrische Netwerken.

De bundel is voorzien van antwoorden.

152 pag., ISBN 90-6562-005-2

ELEKTRISCHE SCHAKEL- VERSCIJNSELEN

door ir. A. Henderson

Het hoofdstuk schakelverschijnselen uit de theorie van de elektrische netwerken is een dankbaar onderdeel. Enerzijds heeft het de charme van een streng mathematisch betoog, anderzijds komt men voortdurend in aanraking met de fysische werkelijkheid. Soms blijken mathematische begrippen onverwacht een fysische betekenis te hebben. Vele uitgewerkte voorbeelden en vraagstukken zijn opgenomen.

120 pag., ISBN 90-6562-xxx-x

DISCRETE SIGNALLEN

door ir. A. Henderson

Digitale signaalbewerking komt steeds meer in de belangstelling. Niet alleen de digitale

filters zijn in opmars, ook bij computer-gerichte netwerkanalyse komt men in aanraking met discrete signalen.

Bij de bestudering van deze stof wordt men getroffen door de analogie met de theorie van de continue signalen, zoals schakelverschijnselen, stationaire toestand, toestandsvergelijkingen, wiskundige transformaties, beginvoorwaarden, polen en nulpunten, enz. Kennis van de theorie van continue signalen is echter niet noodzakelijk voor de bestudering van deze stof. Wel wordt bekendheid verondersteld met gelijk- en wisselstroomtheorie, complexe grootheden, reeksen en (eenvoudige) matrixrekening.

72 pag., ISBN 90-6562-044-3

ELEKTRISCHE EN MAGNETISCHE VELDEN

door ir. A. Henderson

In dit beknopte boek wordt de theorie van de elektrische en magnetische velden voor technici behandeld; daarbij wordt naast de noodzakelijke formules ook aandacht gegeven aan de ontwikkeling van het fysisch inzicht.

In een inleidend hoofdstuk wordt een beknopt overzicht gegeven van de vectoralgebra. Daarna volgen de elektrostatica, de elektrische stromen en magnetische velden, en vervolgens de wetten van Maxwell in integraalvorm. Daarbij komen ook de voor de netwerktheorie noodzakelijke wetten van Kirchhoff naar voren.

Tenslotte volgen de wetten in differentiaalvorm, waarbij tevens wordt ingegaan op de beginselen van de vectoranalyse.

Tevens is een zeventigtal vraagstukken met een antwoordenlijst opgenomen.

96 pag., ISBN 90-6562-027-3

Uitgaven van de Delftse Uitgevers Maatschappij op het terrein van de wiskunde:

ANALYSE

door dr. J.H.J. Almering e.a.

'Analyse' behandelt de analyse op moderne wijze. De hoofdstukken bevatten vele voorbeelden, vraagstukken en oefeningen.

Ze behandelen de grondbegrippen, complexe getallen, limieten en continuïteit, differentiëren, integratie, afbeeldingen, differentiaalvergelijkingen, meervoudige integralen, lijn- en oppervlakteintegralen en reeksen. Tevens zijn vele opgaven (mét antwoorden) opgenomen.

Het boek doet o.a. dienst bij het analyse-onderwijs bij nagenoeg alle afdelingen van de TH-Delft.

Het is samengesteld door:

prof.dr. J.M. Aarts, dr. J.H.J. Almering, drs. N.C. Balkenende, dr. H. Bavinck, ir. J.J.I.M. van Kan, drs. J.S. Mahler en drs. L. Sleutelberg.

400 pag., ISBN 90-6562-002-8

Van bovengenoemd Analyseboek wordt een selectie van de *numerieke onderwerpen* apart uitgegeven ten behoeve van de M.O.-cursussen (52 pag.).

ANALYSE

209 tentamenopgaven met uitwerkingen door dr. H. Bavinck

Aansluitend aan Analyse door dr. J.H.J. Almering e.a. redigeerde dr. H. Bavinck een boek met opgaven en uitwerkingen.

Het aan studenten ter beschikking stellen van vraagstukken met uitwerkingen, betekent didactisch gezien een risico; van de gebruikers moet dan ook, wil men het boekje met vrucht hanteren, een zekere zelfdiscipline worden verwacht.

96 pag., ISBN 90-6562-060-5

DIFFERENTIAALVERGELIJKINGEN

220 voorbeelden en opgaven met oplossingen en beknopte theorie door drs. A. Schuitman

Aan de hand van voorbeelden en vele opgaven wordt in dit boek een overzicht gegeven van de verschillende typen differentiaalvergelijkingen. Ook wordt ingegaan op het oplossen van stelsels van differentiaalvergelijkingen en op toepassingen op partiële differentiaalvergelijkingen. Tenslotte worden Laplacetransformaties en randwaardenproblemen behandeld.

Het boek is vooral bedoeld als vraagstukkenverzameling naast een college of leerboek.

160 pag., ISBN 90-6562-026-5

NUMERIEKE ANALYSE

door prof.dr. J.A. Zaat

Dit boek is bedoeld als leidraad voor de studie van de numerieke analyse aan de TH-Delft, HTS'en, Hogere Zeevaartscholen en andere instellingen van het h.b.o.

In een achttal hoofdstukken worden behandeld het rekenen met benaderende getallen, benaderingsmethoden voor het oplossen van vergelijkingen, niet-lineaire vergelijkingssystemen, interpolatie, differentiatie en integratie, numerieke integratie van differentiaalvergelijkingen, de methode der kleinste kwadraten, stelsels lineaire algebraïsche vergelijkingen en eigenwaarden en eigenvektoren van een matrix.

De bijlagen bevatten dertien voorbeelden van algo-programma's.

176 pag., ISBN 90-6562-0xx-x

DISCRETE ANALYSE

door Peter Nooy, Jan Vons en Rob Eveleens, syllabus bij het college van prof.dr. H.J.A. Duparc

Naast de conventionele analyse, die wel wordt aangeduid als 'continue' analyse, komt steeds meer de 'discrete' analyse naar voren. In tegenstelling tot de continue analyse wordt in de discrete analyse het begrip 'limiet' niet of nauwelijks gebruikt. Vanuit andere wetenschappen dan wiskunde en de techniek dienen zich vele discrete problemen aan; zo is voor het zoeken van oplossingen met behulp van een computer vaak discretisering nodig.

Bij de opzet van dit vak kan dankbaar gebruik worden gemaakt van de continue analyse.

De behandelde onderwerpen en hun (tussen haakjes geplaatste) analogon uit de continue analyse zijn:

differentierekening (differentiaalrekening)
somrekening (integraalrekening)
reeksen (reeksen)
differentievergelijkingen (differentiaalvergelijkingen)
Z-transformatie (Laplace-transformatie)

52 pag., ISBN 99-6562-0xx-x

DICTAAT LINEAIRE ALGEBRA

door dr. G.W. Decnop, ir. H. van Iperen
en dr. R. Martini

In een systematische opbouw behandelen de auteurs de lineaire algebra, zoals die wordt gegeven aan de TH-Delft. Daarbij zijn vele voorbeelden en vraagstukken opgenomen. Vectorruimten, matrices en rekentechnieken in $\mathbb{R}^n$, lineaire afbeeldingen en bilineaire vormen, inwendige productruimten, stelsels lineaire vergelijkingen, determinanten, lineaire operatoren van vectorruimten, lineaire operatoren van inwendige productruimten en kwadratische vormen zijn de onderwerpen van de hoofdstukken.

236 pag. (formaat 19x26)

ISBN 90-6562-036-2

ELEMENTAIRE STATISTIEK

door ir. J. van Soest

Het bekende boekje van ir. Van Soest richt zich vooral op de toepassingen van de statistiek. Achtereenvolgens worden behandeld de beschrijvende statistiek, de kansrekening, stochastische variabelen, populatie en steekproef, de binomiale verdeling, de Poisson-verdeling, de normale verdeling, functies van continue stochastische variabelen, de centrale limietstelling, statistische toetsen voor ligging, toetsen voor verschil in ligging en toetsen voor varianties, regressie- en correlatierekening. Tal van vraagstukken zijn opgenomen.

176 pag., ISBN 90-6562-003-6

aanvulling ELEMENTAIRE STATISTIEK

door ir. J. van Soest, ir. A.J. Meelen
en ir. J.M.G. Vermeulen

Ten behoeve van een meer mathematische benadering van de statistiek is een aanvulling beschikbaar, die een verdieping inhoudt van hetgeen in de hoofdstukken 3, 7, 8 en 13 van Elementaire Statistiek is weergegeven.

60 pag., ISBN 90-6562-006-0

overige uitgaven op het terrein van de
wiskunde

REPETITIEDICTAAT ANALYSE I en II

door ir. W.J. Vollewens

deel I 342 pag.

deel II 240 pag.

VRAAGSTUKKEN OVER ANALYTISCHE MEETKUNDE EN LINEAIRE ALGEBRA

door drs. B.W. Steggerda e.a.

192 pag.

ANALYTISCHE MEETKUNDE I en II

door dr. J. Bijl en drs. W.J.H. Salet

deel I, 200 pag.

deel II, uitverkocht

VECTORANALYSE

door prof.dr. R. Timman en dr. J.W. Reijn

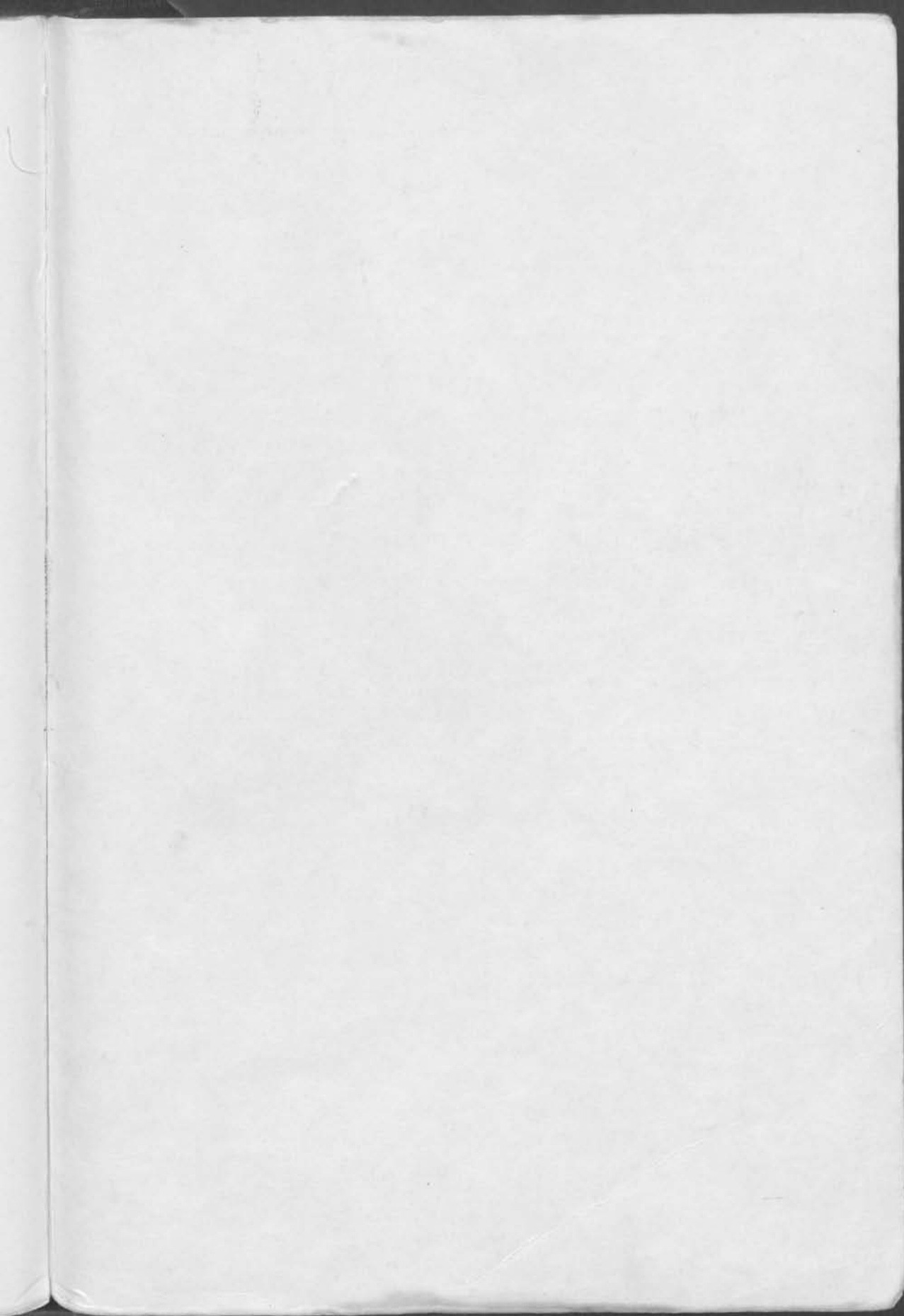
164 pag.

VRAAGSTUKKEN OVER WAARSCHIJNLIJKHEIDSREKENING (met inleidend overzicht van de grondbeginselen)

door dr. P.J.A. Kanthers

178 pag.

Delftse Uitgevers Maatschappij b.v.
Postbus 2851, 2601 CW Delft
Tel. 015-123725



Dit boek wordt ten behoeve van de Vereniging voor Studie- en Studentenbelangen te Delft (VSSD) uitgegeven door de Delftse Uitgevers Maatschappij.

De VSSD is een vereniging van studenten aan de Technische Hogeschool Delft, die zich ten doel stelt de belangen van de studenten te behartigen.

Deze belangenbehartiging heeft vele, overigens samenhangende, kanten. De verdediging van de kwaliteit van het onderwijs, bezinning op de beroepspraktijk en het bevorderen van de toegankelijkheid van het wetenschappelijk onderwijs voor alle lagen van de bevolking zijn de hoofdzaken van wat de 'ideële' kant van de belangenbehartiging genoemd zou kunnen worden.

De 'materiële' kant betreft het opkomen voor een aanvaardbaar inkomen voor de student, voor goede leefomstandigheden (huisvesting, voedsel) en goedkoop studiemateriaal e.d.

Bij het verzorgen en doen uitgeven van boeken zoals deze zijn de beide aspecten vertegenwoordigd: de beschikbaarheid van goede en handzame werken vergroot de kwaliteit van het onderwijs en verbetert de studieresultaten, anderzijds worden de boekwerkjes zo goedkoop mogelijk (d.i. tegen kostprijs) aan de leden van de VSSD beschikbaar gesteld. Daarbij kunnen ook anderen tegen een, zij het (vanwege de verkoopkosten) hogere, doch ook nog lage prijs deze werkjes kopen.

ISBN 90 6562 034 6