DIGITALE ELEKTRONICA

VOOR

INDUSTRIEEL ONTWERPEN

ET15-73

ir.F.L.Muller

september 1999

Inhoudsopgave

1 Analoog-digitaal 7
1.1 Inleiding 7
1.2 Het begrip "digitaal" 7
1.3 Informatieverwerking 9
1.4 Codering 11
1.5 Bitverwerking 12

2 Data-representatie 13
2.1 Getallen 13
2.1.1 Integers 13
2.1.2 Getallen met teken 16
2.1.3 Reals 17
2.2 Characters 18
2.3 Booleans 19
2.4 Datastrukturen 19
2.4.1 Array 19
2.4.2 Record 20
2.4.3 Linked list 20

3 Combinatorische schakelingen 23
3.1 Boole algebra 23
3.1.1 Bewerkingen op functies 24
3.1.2 Waarheidstabellen 24
3.2 Poorten 26
3.2.1 Elementaire poorten 26
3.2.2 De electronica van de poort 28
3.2.3 Programmeerbare logica 30
3.2.4 Uitvoering 30
3.3 Schakelingen met poorten 31
3.3.1 Code omzetters 32
3.3.2 Karnaugh map 34
3.3.3 Computerschakelingen 35

4 Sequentiële schakelingen 43
4.1 Het geheugenelement 43
4.2 Registers 47
4.3 Schakelingen met registers 52

5 Verbindingssystemen 55
5.1 Three-state 55
5.2 Open collector 56

6 Geheugens 57
6.1 Overzicht halfgeleidergeheugens 57
6.2 Random Access Memory (RAM) 58
6.2.1 Statische RAM 58
6.2.2 Dynamische RAM 59
6.2.3 Bipolaire RAM 59
6.3 Read Only Memory (ROM) 60
6.3.1 Mask ROM's 60
6.3.2 PROM's 61
6.3.3 EPROM's 61
6.3.4 Seriële geheugens 61
6.4 Geheugenmodule 62

7 De Computer 65
7.1 Inleiding 65
7.2 Opbouw van de C.P.U. 67
7.3 Opbouw van de instructie 69
7.4 Instructieafhandeling 71
7.5 De instructie set 78
7.5.1 Copieer operaties 78
7.5.2 Data bewerkings operaties 78
7.5.3 Vergelijk en test operaties 79
7.5.4 "Control transfer" operaties 80
7.5.5 Overige operaties 81
7.6 Adresseringsmethoden 82
7.6.1 Direkte adressering 82
7.6.2 Onmiddellijke adressering 83
7.6.3 Registeradressering 83
7.6.4 Indirekte adressering 83
7.6.5 Geïndexeerde adressering 85
7.6.6 Relatieve adressering 86
7.6.7 Autoindexering 87
7.7 Instructieset van de 80C51 88

8 Assemblerprogrammering 91
8.1 Symbolische operatiecodes en adressen 91
8.2 Assemblerprogramma's 91
8.3 De funktie van de assembler 92
8.4 De struktuur van de 80C51 microcontroller 93
8.5 De instrukties van de 80C51 95
8.6 De vormen van adressering van de 80C51 100
8.7 Conventies en pseudoinstrukties van de assembler 103
8.8 Werken met de assembler 105
8.9 Voorbeelden van programma's voor de 80C51 106
8.10 Het gebruik van subroutines 113
8.11 Aanroep en terugkeer van een subroutine bij de 80C51 113
8.12 Parameteroverdracht 117
8.13 Variabelen 120
8.14 Herbetreedbaarheid 122
8.15 Voorbeelden van programma's met subroutines 123

9 Programma ontwikkeling 127
9.1 Inleiding 127
9.2 Vertalen 127
9.2.1 Assembler 128
9.2.2 Compiler 129
9.2.3 Interpretator 132
9.3 Voorbeelden van executie 133
9.4 Editor 136
9.5 Foutzoeken 136
9.5.1 Debuggers 136
9.5.2 Ontwikkelsystemen 137
9.6 Assemblertaal of HLL 139

10 Linking en Loading 141
10.1 Inleiding 141
10.2 Compile, load en go 142
10.3 De absolute loader 142
10.4 De relocating loader 143
10.5 Linken, libraries en de linking loader 145
10.6 De linkage editor 148
10.7 Overlays 148
10.8 Virtuele geheugens 150

11 Input en Output 151
11.1 Randapparatuur 152
11.2 Mens-computer communicatie 152
11.2.1 De terminal 152
11.2.2 Serie - parallel omzetting 153
11.2.3 Transmissieconfiguraties 154
11.2.4 Synchroon en asynchroon 155
11.2.5 Modem 156
11.2.6 De printer 157
11.2.7 De muis 158
11.2.8 De scanner 158
11.3 Achtergrondgeheugen 159
11.3.1 Magneetbanden 160
11.3.2 Magneetschijven 161
11.3.3 C D ROM 163
11.4 Sensoren en Aktuatoren 164
11.4.1 Sensoren of opnemers 164
11.4.2 Aktuatoren 164
11.5 Computerverbindingen 165
11.6 Interfacing 166
11.6.1 I/O onder programabesturing 167
11.6.2 I/O met interrupt 169
11.6.3 DMA 172
11.7 Interface voorbeelden 80C51 175
11.7.1 D/A en A/D omzetting 175

12 Operating System 181
12.1 Inleiding 181
12.2 Kernal 184
12.2.1 First level interrupthandler 185
12.2.2 WAIT en SIGNAL 186
12.2.3 Dispatcher 187
12.3 Geheugensystemen 187
12.3.1 Computergeheugen als hierarchie 187
12.3.2 Virtueel geheugen 189
12.3.3 Base en limit registers 190
12.3.4 Paginering 190
12.3.5 Segmentering 192
12.4 Input en Output 194
12.4.1 I/O procedures and handlers 195
12.4.2 Spooling 197

1 Analoog-digitaal

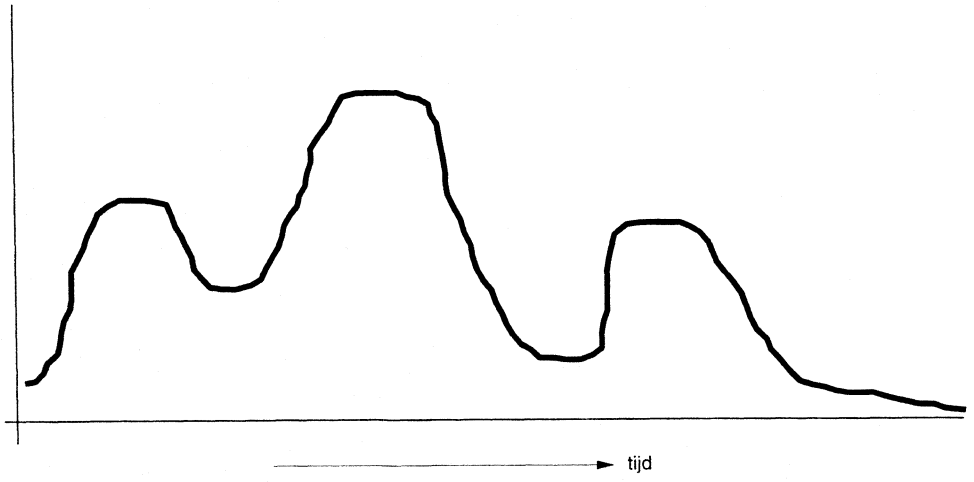
1.1 Inleiding

De wereld om ons heen beschouwen we meestal analoog, de hoeveelheid licht een nuance in kleur of smaak, de luidheid van een geluid of de toonhoogte daarvan worden als een continue spectrum beschouwd. Pas toen men ging tellen en waarde ging toekennen aan aantallen, was er een behoefte om dingen digitaal te gaan beschouwen. Een wijzerhorloge suggereert een oneindige nauwkeurigheid, elk tijdstip kan door de wijzers worden aangegeven. Maar als boven het examenwerk de tijdsduur moet worden aangegeven, dan worden niet 2 wijzerplaten getekend, maar schrijven we bijvoorbeeld 14.00 - 16.00. Om verwerking van diverse signalen mogelijk te maken moeten we de manier waarop we naar die signalen kijken, herzien.

1.2 Het begrip "digitaal"

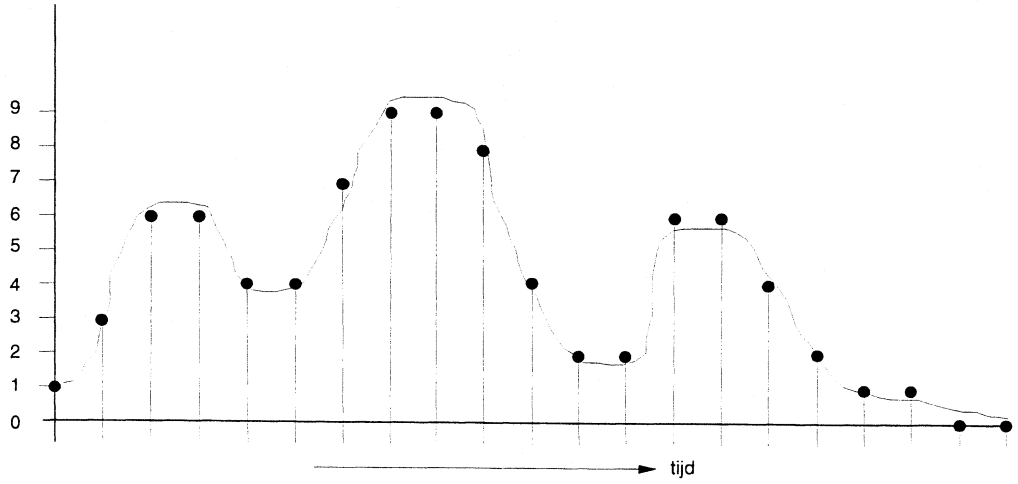
Een digitale schakeling, zoals bijvoorbeeld een computer of een digitaal horloge ,bevat aan essentiële componenten precies dezelfde onderdelen als een radio- of T.V.ontvanger, namelijk weerstanden, condensatoren, transistoren en dergelijke. Al deze apparaten ontvangen, verwerken en presenteren informatie, die inwendig in de vorm van spanningen of stromen wordt voorgesteld. Toch bestaat er een essentieel verschil, waardoor de ene kategorie zich onderscheidt van de andere kategorie, elk met hun eigen toepassingsmogelijkheden.

Het is verschil in beschouwingswijze van het elektrische signaal. In apparaten zoals radio's wordt de hoogte van een elektrische spanning (of stroomsterkte) beschouwd als de informatiedragende eigenschap van het te verwerken signaal. De waarden van de spanning of de stroom zijn op ieder moment recht evenredig met de te representeren geluidsdruk. Als het geluidssignaal een sinus met bepaalde frequentie en amplitude heeft, dan is er ook een elektrische grootheid (spanning, stroom of frequentie) aan te wijzen die daarmee evenredig is. Na een evenredige elektrisch-mechanishe omzetting van stroom naar luchtdruk d.m.v een luidspreker, verkrijgt men weer het oorspronkelijke geluid. Deze methode van afbeelding van informatie op een signaal noemt men continu of ook wel analoog.



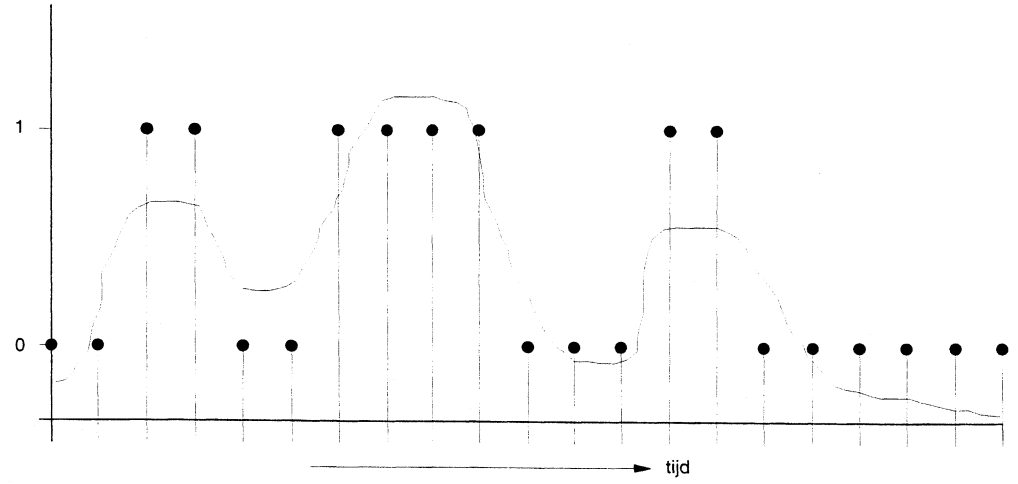
Afbeelding 1 Analoge beschouwing

Een andere beschouwingswijze is die, waarbij aan het elektrische signaal slechts een eindig aantal, goed gedefinieerde spanningswaarden worden onderscheiden. Tussengelegen spanningen worden afgerond naar de dichtsbijzijnde gedefinieerde waarde. Ook beschouwt men het signaal niet voortdurend, maar slechts op vaste, afgesproken tijdstippen. Kleine verstoringen hebben geen invloed, omdat na een nieuwe afronding de oude waarden weer kunnen worden hersteld. Men noemt deze beschouwingswijze digitaal



Afbeelding 2 Digitale beschouwing

In Afbeelding 2 wordt op discrete momenten de afgeronde waarde van het signaal uit Afbeelding 1 genomen. In plaats van de grafiek had ook de reeks getallen 1,3,6,6,4,4,7,9,9,8,4,2,2,6,6,4,2,1,1,0,0 als beschrijving van het signaal gegeven kunnen worden.



Afbeelding 3 Binaire beschouwing

De technisch meest simpele vorm van een digitale beschouwing is er één waarbij slechts 2 waarden van het signaal worden beschouwd. Deze tweewaardige beschouwingswijze wordt de binaire beschouwingswijze genoemd. Dit is aangegeven in Afbeelding 3. Het werken met slechts 2 gedefinieerde waarden geeft een hoge mate van storingsongevoeligheid. Voor elke waarde zijn nog slechts 2 mogelijkheden, bijvoorbeeld een hoge en een lage spanning. De hoeveelheid informatie van één waarde wordt een bit genoemd.

Het binaire signaal zou ook geschreven kunnen worden als 0,0,1,1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0
Het aantal toepassingen waarbij niet 2 maar meer discrete spanningsniveaus worden onderscheiden, is relatief gering. Om deze reden worden in het technisch spraakgebruik de begrippen "digitaal" en "binair" (= 2-waardig digitaal) door elkaar gebruikt. Maar met de term "digitaal" zal men vrijwel altijd een systeem aanduiden dat slechts met 2 signaalwaarden werkt.

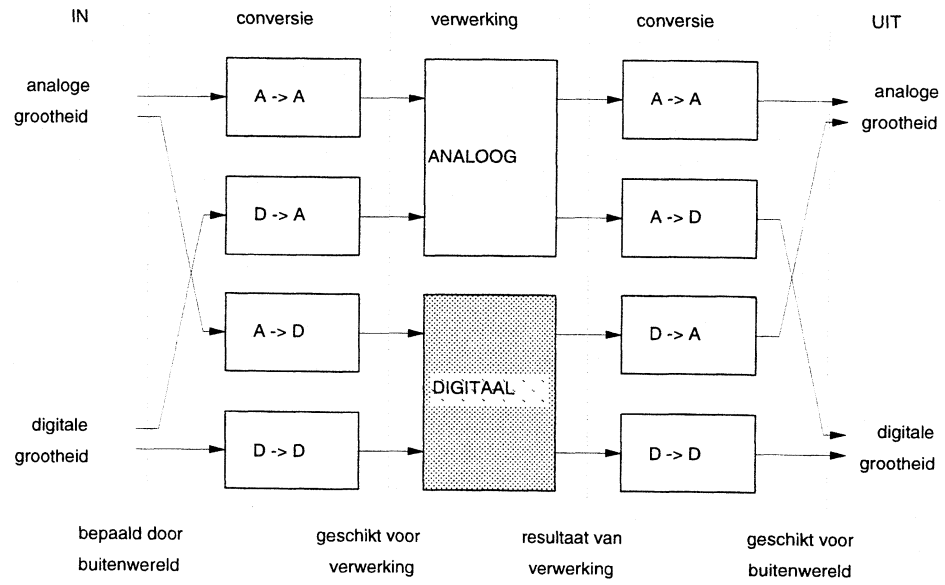
Een voorbeeld van zowel de analoge beschouwingswijze als de digitale (= binaire beschouwingswijze) vinden we in de geluidsregistratie. Bij de grammofoonplaat is in de groef duidelijk het geluidssignaal te herkennen. En bij een cassetterecorder zal de magnetisatie van de band direct een maat voor het geluidssignaal zijn. Dit zijn dus analoge registraties van het geluid. Maar bij een Compact Disc wordt het signaal vastgelegd door het al of niet aanbrengen van kleine putjes in een aluminiumplaatje. Er wordt niet naar de vorm van de putjes gekeken, alleen het aan- of afwezig zijn is belangrijk. Dit is dus duidelijk een vorm van digitale registratie.

1.3 Informatieverwerking

Wat is de zin van het werken met digitale grootheden? Er is een belangrijk verschil tussen het manipuleren met analoge en met digitale grootheden. Na een bewerking van een analoge grootheid (bijvoorbeeld het tienvoudig versterken van een elektrisch spanningsverschil: het bewaren van een spanningsverschil om dat een dag later weer te kunnen gebruiken.) is het lastig of zelfs onmogelijk de verkregen grootheid te ontdoen van onnauwkeurigheden die zijn ontstaan door onvoorspelbare afwijkingen tijdens de bewerking of onvolkomenheden in de informatiedrager bij het opslaan van de informatie. Bij analoge bewerkingen en opslag ontstaan allerlei afwijkingen, componenten zijn niet ideaal, zij hebben een onnauwkeurigheid in waarde, zijn warmteafhankelijk of gevoelig voor andere soorten ruis van buiten af, vertonen een niet lineair gedrag enz. Het verbeteren van de nauwkeurigheid van zo'n analoog systeem is duur en vaak zelfs onmogelijk.

Daarom verdient het aanbeveling naar alternatieve wegen te zoeken om toch de gewenste nauwkeurigheid te kunnen verkrijgen. Het antwoord kan worden gevonden in het toepassen van digitale technieken. Het toepassen van elementen die, door hun beschouwingswijze, geen afwijking vertonen, maakt het mogelijk om de vereiste nauwkeurigheid te bereiken. De hoeveelheid componenten en verwerkingsstappen zullen toenemen, maar zijn technisch uitvoerbaar. Grote hoeveelheden componenten zijn onder te brengen op kleine stukjes silicium. Door de hoge snelheden die de qua opbouw simpele componenten, kunnen halen is een groot aantal processtappen ook geen bezwaar meer. Alleen signalen, die analoog beschouwd, zeer hoge frequentiecomponenten hebben, zijn lastig digitaal en real-time te verwerken. Bij de keuze tussen een realisatie met een analoge verwerking of een digitale verwerking zal een aantal punten een rol kunnen spelen.

In Afbeelding 4 zijn de mogelijkheden voor een informatieverwerkend systeem aangegeven.



Afbeelding 4

Om een keuze te maken uit de mogelijkheden moet de "optimale" weg door het systeem gevonden worden. De belangrijke gegevens hierbij zijn dan:

1. Hoe wordt het signaal aangeboden analoog of digitaal ?

2. In welke vorm wil men de verwerkte gegevens hebben ?

3. Is de verwerking eenvoudiger analoog of digitaal uit te voeren ?
   (binnen de grenzen van geëiste snelheid en nauwkeurigheid)

4. Zijn de conversies van signaal naar verwerkingseenheid en omgekeerd
   van verwerkingseenheid naar gewenste grootheid eenvoudig ?

Er zijn sensoren die grootheden als licht, temperatuur, vochtigheid e.d. omzetten in een spanning. Als de gegevens van zo'n sensor digitaal verwerkt moeten worden, zal er nog een omzetting moeten plaatsvinden spanning naar digitale waarden. Maar er zijn ook sensoren die de gemeten grootheid omzetten in een pulsreeks waarvan de pulsbreedte of de frequentie de maat voor het gemeten signaal aangeeft. In het laatste geval kan een digitaal systeem door tijden te meten, direct over de gegevens beschikken. Het eventuele bezwaar van een D=>A (Digitaal naar Analoog) omzetting komt dan te vervallen. Evenzo als het toerental van een motor geregeld wordt door de aangeboden spanning te variëren zal bij een digitaal systeem een D=>A (Digitaal naar Analoog) omzetting moeten plaatsvinden. Maar vaak is het ook mogelijk om de motor te voeden met pulsen met een vaste spanning, maar een geregelde pulsduur. Ligt de frequentie van het signaal boven een bepaalde grens, dan zal door de rotatietraagheid van het draaiend systeem de motor geen hinder ondervinden van het pulserende karakter van de aangeboden spanning. De omzetting vanuit een digitaal systeem kan dan weer betrekkelijk eenvoudig zijn.
Samenstelling en componentkeuze van een analoog systeem zijn sterk afhankelijk van het type verwerking wat moet plaatsvinden. Een andere toepassing maakt een andere schakeling noodzakelijk. Het digitale systeem bestaat vaak uit een kleine processor de microprocessor of de microcontroller. Bij een andere toepassing zal de programmatuur voor het uit te voeren algoritme wijzigen, maar de gebruikte "hardware" kan dezelfde blijven. Om een verwerking digitaal uit te voeren moet

men over een verwerkingsalgoritme beschikken en een processor die snel genoeg werkt om de berekeningen waaruit het algoritme bestaat, binnen de gestelde tijd te kunnen uitvoeren.

1.4 Codering

De elementen binnen de digitale verwerking laten slechts 2 waarden toe, meestal aangegeven met 0 en 1. Er kon een directe binaire beschouwing van een signaal gemaakt worden volgens 2. Het signaal voor het digitale systeem is dan:

0,0,1,1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0

Als de nauwkeurigheid van dit signaal te klein is kan er gebruik gemaakt worden van het signaal uit Afbeelding 2 nl:

1,3,6,6,4,4,7,9,9,8,4,2,2,6,6,4,2,1,1,0,0

Om dit signaal geschikt te maken voor een digitaal systeem, kan dit signaal gecodeerd worden. Voor elke digitale waarde wordt dan een combinatie van bits (= binaire waarden) gebruikt. Als zo'n combinatie bijvoorbeeld uit 4 bits bestaat dan zijn er 2*2*2*2=16 mogelijke combinaties. In het hoofdstuk Data-representatie zullen we zien dat dit gewoon getallen zijn. Na codering zou het signaal er dan als volgt uitzien:

```
1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0
0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0
0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0
0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0
```

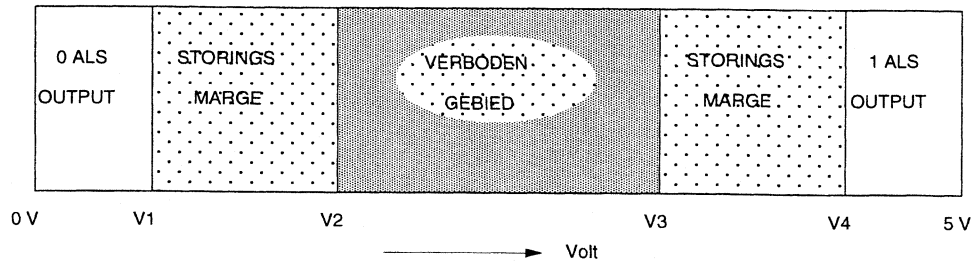
Er worden hier een aantal bits tegelijkertijd verwerkt. Dit is parallelle verwerking. De bits worden parallel aangeboden.
Een andere mogelijkheid is het serieel verwerken en aanbieden volgens:

```
0,0,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,1,1,0,0,-
1,1,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,1,0,0,1,1,0,0,1,-
0,0,0,1,1,0,0,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,-
0,0,0
```

Bij seriële verwerking is meer tijd nodig, maar minder hardwarecomponenten.

1.5 Bitverwerking

Dat een binaire beschouwingswijze van een signaal tot een storingsongevoelig systeem leidt is te zien in Afbeelding 5:



Afbeelding 5

Verondersteld wordt dat een element werkt op een spanning van 5 volt. Als het element een 0 produceert zal dit tussen 0 en V1 volt liggen. Een 1 zal een waarde tussen V4 en 5 Volt geven. Digitaal beschouwd zijn dit de enige mogelijke waarden. Door componentafwijkingen en ruis moet het element altijd in staat geacht worden zijn uitgangswaarden in deze twee gebieden te produceren. Wordt nu tijdens transport naar het volgende element het signaal, analoog beschouwd, verminkt en komt het in het gebied van V1 tot V2 Volt of V3 tot V4 volt terecht, dan zal herstel mogelijk zijn. De ingangsschakeling van het element bewerkt het zodanig dat aan zijn uitgang weer de gedefinieerde waarden voor 0 en 1 tevoorschijn komen. Slechts in het gebied van V2 tot V3 Volt is de waarde zo twijfelachtig dat herstel niet wenselijk meer is. Dit wordt als verboden gebied aangegeven.

2 Data-representatie

In het dagelijks leven worden gegevens vaak voorgesteld door getallen en teksten. Wij gebruiken daarvoor het 10-tallig stelsel en de arabische lettertekens als codering. Als dit soort gegevens moet worden verwerkt in een informatie-verwerkend systeem dat slechts twee toestanden kent, dan moeten deze gegevens opnieuw worden gecodeerd. De kleinste hoeveelheid informatie, bij een tweekeuze-mogelijkheid, wordt BIT (BInary digiT) genoemd. 8 bits vormen een byte en 4 bits zijn een nibble. Een willekeurige hoeveelheid bits bij elkaar vormen een woord. Als voor de twee toestanden de "0" en de "1" worden gebruikt, zullen alle voorkomende gegevens worden voorgesteld door rijtjes symbolen of woorden met 0 en 1. Als een woord uit n bits bestaat zijn er dus 2^n verschillende combinaties.

2.1 Getallen

Hier worden drie soorten getallen bekeken:

- de positieve gehele getallen of integers
- de getallen met teken
- de gebroken getallen of reals.

2.1.1 Integers

Wij zijn gewend om positieve gehele getallen weer te geven door een rijtje cijfersymbolen. De waarde van het getal wordt bepaald door het gewicht van het cijfersymbool en de plaats van het cijfersymbool in het getal. De waarde van het getal 4053 is samengesteld uit $4 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 3 \times 10^0$. De codering van zo'n getal vindt plaats in een zogenaamde polynoomcode, het getal 10 wordt het grondtal van het talstelsel genoemd. Als bij gebruik van een polynoomcode getallen onder elkaar gezet worden, komen de cijfers met gelijke plaatsgewichten netjes in kolommen boven elkaar. Dit maakt het cijfermatig verwerken van een aantal getallen, bijvoorbeeld optellen, eenvoudig. (De romeinen zullen hier meer moeite mee gehad hebben, en ook voor de chinezen is het iets moeilijker).

De waarde van het getal $C_{n-1}C_{n-2}.....C_1C_0$ gecodeerd in een polynomcode kan als volgt worden weergegeven:

$$W = \sum_{i=0}^{n-1} C_i . g^i$$

Hierin stelt C het cijfersymbool, i de plaats en g het grondtal van het talstelsel voor. Voor C moet gelden $0 \leq C < g$.
Als voor g nu 2 gekozen wordt, zijn er maar 2 cijfersymbolen, 0 en 1 nodig. De waarde van het zuiver binaire getal 101111101101 is dus:

$\mathbf{1}\times2^{11} + \mathbf{0}\times2^{10} + \mathbf{1}\times2^9 + \mathbf{1}\times2^8 + \mathbf{1}\times2^7 + \mathbf{1}\times2^6 + \mathbf{1}\times2^5 + \mathbf{0}\times2^4 + \mathbf{1}\times2^3 + \mathbf{1}\times2^2 + \mathbf{0}\times2^1 + \mathbf{1}\times2^0 = 3053_{10}$
of
$((((((((((\mathbf{1}\times2+\mathbf{0})\times2+\mathbf{1})\times2+\mathbf{1})\times2+\mathbf{1})\times2+\mathbf{1})\times2+\mathbf{0})\times2+\mathbf{1})\times2+\mathbf{1})\times2+\mathbf{0})\times2+\mathbf{1}) = 3053_{10}$

Om met getallen die zuiver binair gecodeerd zijn te kunnen rekenen moet men alleen de optel- en aftrek tabellen en de tafels van vermenigvuldiging aanpassen. De manier van rekenen blijft gelijk. Als op één cijferplaats respectievelijk de bewerkingen A+B en A-B worden uitgevoerd geven de volgende tabellen het resultaat.

OPTELLEN

carryin	B	A	carryuit	som
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

AFTREKKEN

borrowin	B	A	borrowuit	verschil
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Hierin heeft het begrip carry aan of er al of niet 1 onthouden moet worden en het begrip borrow staat voor het al of niet lenen van de volgende kolom.
De tafels van vermenigvuldiging zijn wel heel eenvoudig:

VERMENIGVULDIGEN

B	A	produkt
0	0	0
0	1	0
1	0	0
1	1	1

Voorbeelden:

OPTELLEN

```
 71 = 1 0 0 0 1 1 1
 43 =   1 0 1 0 1 1
      ---------------  +
som = 1 1 1 0 0 1 0  = 114
carry = 0 0 0 1 1 1 1
```

AFTREKKEN

```
 71 = 1 0 0 0 1 1 1
 43 =   1 0 1 0 1 1
      ---------------  -
verschil = 0 0 1 1 1 0 0  = 28
  borrow =   1 1 1 0 0 0
```

VERMENIGVULDIGEN

```
 71 = 1 0 0 0 1 1 1
 43 =   1 0 1 0 1 1
      ---------------  x
        1 0 0 0 1 1 1
      1 0 0 0 1 1 1
    0 0 0 0 0 0 0
  1 0 0 0 1 1 1
0 0 0 0 0 0 0
1 0 0 0 1 1 1
-------------------------------  +
1 0 1 1 1 1 1 0 1 1 0 1 = 3053
0 0 0 0 1 1 1 1 1 1 1 0 = carry
```

DELEN

```
71 = 1 0 0 0 1 1 1 / 1 0 1 1 1 1 1 0 1 1 0 1 \
                     1 0 0 0 1 1 1 |           | 1
                     ---------------           |
                       1 1 0 0 0 0 |           |
                       1 1 0 0 0 0 1 |         | 0
                       1 0 0 0 1 1 1 |         |
                       ---------------         | 1
                         1 1 0 1 0 1 |         |
                         1 1 0 1 0 1 0 |       | 0
                         1 0 0 0 1 1 1 |       |
                         ---------------       | 1
                           1 0 0 0 1 1 1 |     |
                           1 0 0 0 1 1 1 |     | 1
                           ---------------     |
                                             0 |
1 0 1 0 1 1 = 43 ------------------------------ /
```

Twee andere, veel gebruikte waarden van het grontal g zijn 8 en 16. Voor g is 8 heeft men te maken met het octale talstelsel. De 8 gebruikte cijfersymbolen zijn 0,1,2,3,4,5,6,7. Om een zuiver binair getal te vertalen naar het 8-tallig stelsel verdeelt men vanaf de laagstwaardige kant het 2-tallig getal in groepjes van 3 bits, en vervangt elk groepje van 3 bits door één cijfer met de waarde gelijk aan het 3 bits getal. Voorbeeld:

101111101101 opdelen in 101| 111| 101| 101 en voor elk groepje een cijfer invullen gelijk aan de waarde per groepje ⇒ 5 7 5 5 dus 5755_8.

Het 16-tallig of hexadecimale talstelsel heeft 16 cijfersymbolen,
nl 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Naast de ons bekende 10 cijfers uit het 10-tallig stelsel zijn de letters A t/m F met de bijbehorende waarden 10 t/m 15 toegevoegd. Het codeerrecept hier is verdelen in groepjes van 4 bits, en per groepje het cijfer met gelijk gewicht substitueren. Met bovenstaand voorbeeld wordt dit:

101111101101 opdelen in 1011| 1110| 1101 met per groepje de cijfersubstitutie gelijk aan de waarde per groepje ⇒ B E D dus BED_{16}.
De octale en hexadecimale talstelsels zijn voor menselijk gebruik bedoeld, de apparatuur kan er niet mee werken. Bij toepassing van lange bitpatronen bespaart zo'n octale of hexadecimale codering ruimte, papier en werk.

De conversies van het 2-tallig stelsel naar het octale en het hexadecimale talstelsel en omgekeerd zijn zo eenvoudig omdat zowel 8 als 16 machten van 2 zijn. Bestaat deze relatie niet dan is de methode van converteren iets meer bewerkelijk. De conversie van het p-tallig stelsel naar het q-tallig stelsel kan op 2 manieren worden uitgevoerd.

1. Uitgaande van het getal in het p-tallig stelsel wordt er herhaald door q gedeeld. De hierbij achtereenvolgens geproduceerde resten bepalen de cijfers van het getal in het q-tallig stelsel. De berekeningen zelf worden hier in het p-tallig stelsel uitgevoerd.

2. Uitgaande van de beginwaarde 0 wordt herhaald uitgevoerd: vermenigvuldig met p en tel er één cijfer van het getal uit het p-tallig stelsel bij op. Ga hiermee door tot alle cijfers op zijn. Hier wordt de berekening dus uitgevoerd in het q-tallig stelsel.

Welke methode u moet kiezen hangt af van het feit in welk van de twee talstelsels u het beste kunt rekenen. Meestal zal dit alleen het 10-tallig stelsel zijn. Als zowel p als q ongelijk 10 zijn, is het eenvoudige om maar 2 conversies uit te voeren. Dus van p-tallig naar 10-tallig en vervolgens van 10-tallig naar q-tallig.

Voorbeelden:

converteer 3053_{10} naar het hexadecimale stelsel:

```
3053 : 16 = 190 rest 13  => hexadecimaal het cijfer D
 190 : 16 =  11 rest 14  => hexadecimaal het cijfer E
  11 : 16 =   0 rest 11  => hexadecimaal het cijfer B
```

Dus $3053_{10} = BED_{16}$

converteer BED_{16} naar het 10-tallig stelsel:

```
B = 11    =>      0 * 16 + 11 =   11
E = 14    =>     11 * 16 + 14 =  190
D = 13    =>    190 * 16 + 13 = 3053
```

Dus $BED_{16} = 3053_{10}$

Het rekenwerk is in beide gevallen in het 10-tallig stelsel uitgevoerd.

Moet BED_{16} geconverteerd worden naar het 9-tallig stelsel, en zijn de tafels van vermenigvuldiging bij u en uw pocketcalculator in zowel het 16- als het 9-tallig stelsel onbekend, dan kan beter het 10-tallig stelsel worden tussengelast.
Dit levert dan op:

```
Eerder is al bepaald dat BED16 = 3053₁₀

3053 : 9 = 339 rest 2  =>  negentallig het cijfer 2
 339 : 9 =  37 rest 6  =>  negentallig het cijfer 6
  37 : 9 =   4 rest 1  =>  negentallig het cijfer 1
   4 : 9 =   0 rest 4  =>  negentallig het cijfer 4

Dus BED16 = 4162₉
```

Een andere manier om decimale getallen te coderen in 0 en 1 is het vasthouden aan het decimale talstelsel, maar elk cijfer te vervangen door een groepje bits. Omdat er 10 mogelijke cijfers gecodeerd moeten worden zijn er 4 bits per cijfer nodig. Elk cijfer wordt vervangen door een 4-bits getal met overeenkomstige waarde.

Het voorbeeld $3053_{10} \Rightarrow 0011|0000|0101|0011$ of kortweg 0011000001010011
Dit codeerrecept levert BCD-getallen op. BCD staat voor B(inary)C(oded)D(ecimal).
In BCD-getallen komen de 6 combinaties 1010,1011,1100,1101,1110 en 1111 dus niet voor.
Bij gebruik van deze code worden de rekenregels wat complexer, maar de conversie naar ons 10-tallig stelsel wat eenvoudiger. Daar waar erg veel decimale in en/of output nodig is en weinig rekenwerk kan deze code voordelen bieden. Sommige µ-processoren hebben een speciale instruktie voor het verwerken van BCD-code.

2.1.2 Getallen met teken

Bij een codering voor getallen met teken zou het prettig zijn als aftrekken van een getal bereikt kan worden door optellen van het getal met tegengesteld teken. De tabel voor het aftrekken is dan niet meer nodig.
Als voorbeeld een aftrekking in het 10-tallig stelsel:

```
                          091 - 076 =      de aftrekking
                      091 + (- 076) =      optellen van tegengestelde
      091 + (999 - 076) - 1000 + 1 =       10³ - 1 toevoegen
091 + {(9-0)(9-7)(9-6) + 1} - 1000 =       splitsen per cijfer
091 +        924            - 1000        optellen van het ten's complement
```

Het negeren of tegengesteld maken van 076 gebeurt door elk cijfer van het getal van 9 af te trekken en er 1 bij te tellen, wat de negatieve voorstelling 924 oplevert. Deze bewerking is tussen de accolades uitgevoerd en wordt ten's complement genoemd. De bewerking - 1000 wordt automatisch bereikt door de uitgaande carry van de optelling weg te laten. De getallen 000 t/m 499 zijn nu positief terwijl de getallen 500 t/m 999 negatieve getallen voorstellen.
Dezelfde aftrekking in het 2-tallig stelsel geeft:

```
01011011 - 01001100 =
01011011 + (- 01001100) =
01011011 + (11111111 - 01001100) + 1 - 100000000 =
01011011 + {(1-0)(1-1)(1-0)(1-0)(1-1)(1-1)(1-0)(1-0)+1} - 100000000
01011011 +              10110100                        - 100000000
```

De getallen 00000000 t/m 01111111 zijn positief en 10000000 t/m 11111111 zijn hier negatief. Het van 1 aftrekken (one's complement nemen) wordt inverteren genoemd. Recept om een getal van een tegengesteld teken te voorzien : Alle bits inverteren en er 1 bij optellen.
Deze manier van coderen van getallen wordt two's complement genoemd. Het eerste bit van een two's complement getal wordt het tekenbit genoemd. Een 0 geeft een positief getal aan en een 1 een negatief getal weer.

Voorbeeld:

```
347 = 0101011011                                    0101011011
284 = 0100011100  => two's complementrecept =>    1011100100
    ------------ -                                ------------ +
                        carry weggooien =>    1<= 0000111111 = +63
```

Een andere manier om getal met teken te coderen is de sign-magnitude codering. Hierbij is het eerste bit weer het tekenbit met 0 = + en 1 = -, maar rest van het getal wordt als een positieve integer gecodeerd. Deze getalscodering is voor het vermenigvuldigen eenvoudiger.

2.1.3 Reals

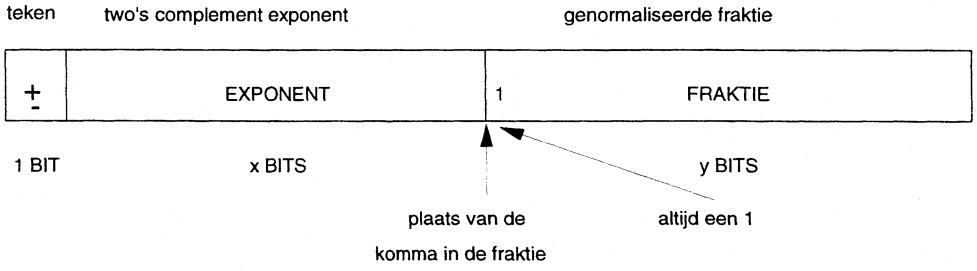
De zuiver binaire codering en de two's complementcodering hebben een beperkt bereik, en beperken zich tot de gehele getallen. Reserveert men n bits voor een getal, dan geldt:

$$\begin{aligned} 0 &\le \text{zuiver binair getal} < 2^n \quad \text{en} \\ -2^{n-1} &\le \text{two's complement getal} < 2^{n-1} \end{aligned}$$

Wil men ook gebroken getallen kunnen weergeven en bovendien het bereik fors uitbreiden ten koste van de nauwkeurigheid, dan wordt overgestapt op een wetenschappelijke notatie met fraktie of mantisse en exponent. Deze floating point notatie kan ook worden gebruikt in het 2-tallig stelsel. Het getal wordt opgebouwd uit drie delen, de fraktie of mantisse, de exponent en het teken. Voor de waarde van het getal geldt:

$$W = \pm \text{FRAKTIE} \cdot 2^{\text{EXPONENT}}$$

Hierin is de exponent volgens het two's complementsysteem gecodeerd. Voor de fraktie geldt: $0.5 \le \text{mantisse} < 1$. Het voorste bit is het tekenbit van het gehele getal. Ook hier geldt 0 = + en 1 = -.



$$-2^{x-1} \le \textit{exponent} \le 2^{x-1} - 1 \qquad 0.5 \le \textit{fraktie} \le \frac{2^y-1}{2^y}$$

Een genormaliseerde fraktie betekent dat de fraktie naar links geschoven wordt totdat de hoogstwaardige 1 op de eerste plaats staat. Het aantal significante bits is dan zo groot mogelijk.
Het bereik van deze codering valt uiteen in 2 intervallen, nl:

$$\left[-2^{+2^{x-1}-1} \cdot \frac{2^y-1}{2^y} \;,\; -2^{-2^{x-1}} \cdot 0.5 \right] \quad \textit{en} \quad \left[2^{-2^{x-1}} \cdot 0.5 \;,\; 2^{+2^{x-1}-1} \cdot \frac{2^y-1}{2^y} \right]$$

Het getal 0 ligt in geen der 2 intervallen. Vaak worden daarom de combinaties met de kleinste exponent als 0 gedefinieerd.
Grote computers hebben speciale hardware om met floating pointgetallen te rekenen soms met een speciale co-processor. Is deze hardware niet aanwezig dan zal dit rekenwerk geheel in software moeten worden uitgevoerd, wat veel meer computertijd vergt.

2.2 Characters

Characters omvatten de voor de mens leesbare lettertekens, cijfertekens en leestekens. daarnaast komen er allerlei besturingskarakters voor. Op de tekens zelf worden geen bewerkingen uitgevoerd, zij worden gebruikt voor in en uitvoer van computers en voor communicatiedoeleinden. De meest bekende code is de ASCII-code. Dit zijn de beginletters van American Standard Code for Information Interchange. De code bevat 128 codecombinaties die in 7 bits gecodeerd zijn. De karakters zijn aaneengesloten alfabetisch gerangschikt. In bijgaande tabel is de codering aangegeven.

ASCII	$b_6b_5b_4$	000	001	010	011	100	101	110	111
$b_3b_2b_1b_0$	hex	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	¦
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Een tweede door IBM veel gebruikte code is de EBCDIC code, wat staat voor Extended Binary Coded Decimal Interchange Code. Dit is een 8 bits code die wel alfabetisch maar niet aaneensluitend is gecodeerd. Meestal worden voor het opbergen van een karakter één byte gebruikt.
Er komen ook karaktercodes voor die niet alfabetisch gerangschikt zijn. Een voorbeeld hiervan is de code die gebruikt werd voor de besturing van de printkoppen van IBM schrijfmachines. Hierin stonden de afzonderlijke besturingscodes van het printbolletje. De letterfrequentie van het Engels is verwerkt in deze besturingscode. De code voor de "e" gaf geen verdraaiing van het bolletje, terwijl de code voor de "q" een samengestelde beweging in gang bracht.

2.3 Booleans

Booleans zijn beslisvariabelen die slechts 2 waarden kunnen hebben, TRUE of FALSE. In hogere programmeertalen komen termen voor als

if CONDITION then ACTION1 else ACTION2

of

while CONDITION do ACTION3

hierin is CONDITION een boolean expression die als antwoord TRUE of FALSE kan opleveren. Meestal wordt TRUE = 1 en FALSE = 0 gekozen. In principe is er slechts één bit nodig om een boolean te coderen. Maar in een groot aantal computers zijn de bits niet afzonderlijk aan te wijzen. Daarom wordt in die gevallen meestal een heel byte gebruikt voor de codering. In de voorbeeldcomputer, de microcontroller 80C51 zijn van 32 bytes de bits wel afzonderlijk te adresseren.

2.4 Datastrukturen

De tot nog toe beschouwde datatypen zijn z.g enkelvoudige typen. Het zijn op zichzelf staande getallen of letters. In een dataverwerkend systeem (meestal een general purpose computer) is ook behoefte om onderling verbanden aan te brengen. Een hoeveelheid letters geven pas een zinvolle tekst als er goed op de volgorde wordt gelet. Om de gegevens te structureren is extra informatie nodig. Deze extra informatie moet ook weer in nullen en enen worden verpakt. Daarom nu een drietal "voorgedefinieerde" datastrukturen.

- Array
- Record
- Linked list

2.4.1 Array

Een array is een lijst met data waarin de afzonderlijke data-elementen met behulp van een index kunnen worden aangegeven. De data-elementen binnen de lijst zijn allen van hetzelfde type, bijvoorbeeld allemaal integers. In het computergeheugen worden de elementen opgeborgen in opvolgende plaatsen. Zo'n plaats wordt aangegeven met een adres. Het array moet eerst gedefinieerd worden voor het gebruikt kan worden. Het volgende voorbeeld is gegeven in de programmeertaal PASCAL.

var TELEFOON: array[10..19] of integer;

Deze opdracht maakt een lijst met 10 plaatsen waarin integers kunnen worden opgeborgen. De aangemaakte struktuur krijgt de naam TELEFOON. Met TELEFOON[13] wordt één element uit de lijst aangewezen. Bij codering of vertaling wordt dit:

adres	inhoud	functie
2400	10	ondergrens index
2401	19	bovengrens index
2402	integer	TELEFOON[10]
2403	integer	TELEFOON[11]
2404	integer	TELEFOON[12]
2405	integer	TELEFOON[13]
2406	integer	TELEFOON[14]
2407	integer	TELEFOON[15]
2408	integer	TELEFOON[16]
2409	integer	TELEFOON[17]
2410	integer	TELEFOON[18]
2411	integer	TELEFOON[19]

Ook meer-dimensionale arrays (matrices) komen voor. In dat geval wordt de plaats aangewezen met een aantal indices bijvoorbeeld BUFFER[5,5,2]. Het aantal indices is gelijk aan de dimensie van het array. Omdat het toch één-dimensionaal wordt opgeslagen wordt er een z.g. dope vector toegevoegd om de elementen te kunnen terugvinden. Voorbeeld:

```
var BUFFER array[3..5,4..5,1..3] of integer
```

maakt een tabel voor integers met een lengte van 18 (3*2*3) plaatsen.
Na vertaling:

adres	functie	adres	functie	dope vector	functie
1200	BUFFER[3,4,1]	1209	BUFFER[4,5,1]	1200	verwijzing
1201	BUFFER[3,4,2]	1210	BUFFER[4,5,2]	3	aantal dimensies
1202	BUFFER[3,4,3]	1211	BUFFER[4,5,3]	3	dim.1 low bound
1203	BUFFER[3,5,1]	1212	BUFFER[5,4,1]	5	dim.1 high bound
1204	BUFFER[3,5,2]	1213	BUFFER[5,4,2]	6	dim.1 multiplier
1205	BUFFER[3,5,3]	1214	BUFFER[5,4,3]	4	dim.2 low bound
1206	BUFFER[4,4,1]	1215	BUFFER[5,5,1]	5	dim.2 high bound
1207	BUFFER[4,4,2]	1216	BUFFER[5,5,2]	3	dim.2 multiplier
1208	BUFFER[4,4,3]	1217	BUFFER[5,5,3]	1	dim.3 low bound
				3	dim.3 high bound
				1	dim.3 multiplier

Om het adres waar BUFFER[5,5,2] staat te bepalen wordt m.b.v de dope vector uitgerekend:

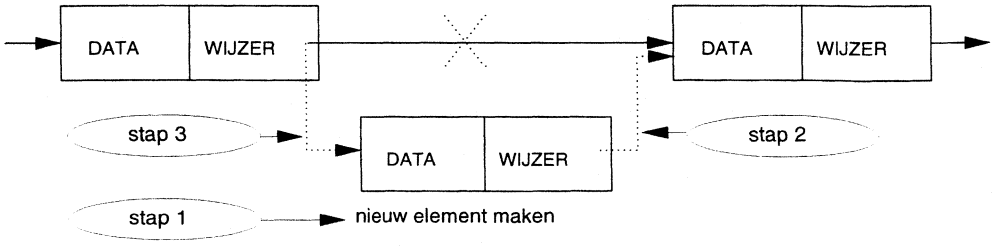
1200 + (5-dim.1low)*dim.1mul + (5-dim.2low)*dim.2mul + (2-dim.3low)*dim.3mul =
1200 + (5-3)*6 + (5-4)*3 + (2-1)*1 = 1216.
De high bound of bovengrenzen worden gebruikt om te controleren of het gevraagde element wel binnen de tabel ligt.

2.4.2 Record

Een andere manier om gegevens gestruktureerd op te slaan is m.b.v. records. Een record bestaat uit "velden" van mogelijk verschillende typen, b.v. integers, reals, karakters, etc. Elk veld kan afzonderlijk aangeduid worden via de "naam" van het veld. In het geheugen heeft elk veld een vaste plaats ten opzichte van het begin van het record.

2.4.3 Linked list

Als data-elementen die bij elkaar horen niet aaneengesloten, maar willekeurig worden opgeborgen, dan kan elk data-element gekoppeld worden aan de plaatsaanduiding van het volgende element of aan een eindkenmerk. Als eindkenmerk wordt vaak op de plaats van de wijzer de waarde 0 gebruikt.
Deze datastruktuur heeft voordelen bij sorteren, immers als een element tussengevoegd moet worden kan dit uitgevoerd worden zonder dat een groot aantal data-elementen van plaats moet verschuiven.



In Afbeelding 7 is aangegeven hoe dit in 3 stappen kan worden gedaan:

1. Maak een nieuw data-element aan.
2. Laat de wijzer van dit nieuwe element naar de opvolger wijzen.
3. Verander de wijzer van de voorganger.

In het nu volgende voorbeeld zijn als data-elementen characters genomen.

	adres	inhoud	adres	inhoud
	1200	O,1208	1200	O,1208
	....		....	
woord:	1204	B,1200	1204	B,1210
	....		....	
	1208	E,1209	1208	E,1209
	1209	R,0	1209	R,0
			1210	R,1200

Duidelijk is te zien dat om van het woord BOER, dat op de plaats woord begint, BROER te maken een nieuwe plaats 1210 is toegevoegd met de letter R en de verwijzing naar 1200. Vervolgens is de verwijzing op de plaats 1204 van 1200 in 1210 veranderd. De inhouden van de overige plaatsen zijn gelijk gebleven.

3 Combinatorische schakelingen

Digitale schakelingen werken met 0 en 1. De elementaire bewerkingen die hierop worden uitgevoerd zijn AND, OR en NOT. Slechts met deze ingrediënten moeten alle complexe bewerkingen worden samengesteld. De Boole algebra is een rekenkunde die laat zien hoe dit kan, en welke aanvullende rekenregels nog nodig zijn.

3.1 Boole algebra

Al in 1854 heeft George Boole de Boole algebra gepubliceerd. Deze rekenkunde laat zien welke bewerkingen met AND, OR en NOT, toegepast op tweewaardige variabelen, kunnen worden samengesteld en hoe met functies van tweewaardige signalen kan worden gerekend. In de notatie worden niet de termen AND, OR en NOT gebruikt, maar respectivelijk de symbolen . + en ' .
AND en OR werken op 2 of meer variabelen NOT werkt op 1 variabele.



George Boole

bewerking	notatie	uitspraak
AND op A en B	F = A.B of AB	A maal B
OR op A en B	F = A+B	A plus B
NOT op A	F = A' of $\bar{A}$	A invers, of niet A

In veel boeken wordt voor het aangeven van de inversie de overstreping van de te inverteren vorm gebruikt. Om druktechnische redenen wordt hier de voorkeur aan de ' gegeven.

Basiseigenschappen

	0' = 1	1' = 0
	0.0 = 0	0 + 0 = 0
	0.1 = 0	0 + 1 = 1
	1.1 = 1	1 + 1 = 1
	A.0 = 0	A + 0 = A
	A.1 = A	A + 1 = 1
	A.A = A	A + A = A
	A.A' = 0	A + A' = 1
Commutatieve wet	A.B = B.A	
	A + B = B + A	
Associatieve wet	A.B.C = A.(B.C) = (A.B).C	
	A + B + C = (A + B) + C = A + (B + C)	
Distributieve wet	A.(B + C) = A.B + A.C	
	A + B.C = (A + B).(A + C)	
Absorptie wet	A + A.B = A	
	A + A'.B = A + B	
	A.(A'+ B) = A.B	
Theorema van De Morgan	(A.B)' = A' + B'	
	(A + B)' = A'.B'	

Het theorema van De Morgan zegt dus dat het inverteren van een functie uitgevoerd kan worden door alle elementen van die functie te inverteren of dualiseren. Elke variabele wordt geïnverteerd, en elke operator wordt gedualiseerd. Dualiseren van de operator betekent:

```
. wordt +, + wordt ., en ' valt weg, want (A')' = A
```

Dus bijvoorbeeld

```
((A + B)' + C.D')' =
((A + B)')' (+)' (C.D')' =
(A + B) . (C'(.)'(D')') =
(A + B) . (C'+ D)
```

3.1.1 Bewerkingen op functies

Als voorbeeld van het bewerken van een functie wordt hier de 2e distributieve wet m.b.v de basiseigenschappen bewezen:

```
(A + B).(A + C) = A.A + A.C + A.B + B.C
                = A + A.C + A.B + B.C
                = A.(1 + C) + A.B + B.C
                = A.1 + A.B + B.C
                = A + A.B + B.C
                = A.(1 + B) + B.C
                = A.1 + B.C
                = A + B.C
```

Het volgende voorbeeld laat zien dat de bewerkingen gebruikt kunnen worden om een formule van een functie te vereenvoudigen:

```
F = A'.B.C' + A'.B.C + A.B'.C + A.B.C' + A.B.C
  = B.(A'.C' + A'.C + A.C' + A.C) + A.B'.C
  = B.(A'.(C'+ C) + A.(C'+ C)) + A.B'.C
  = B.(A' + A) + A.B'.C
  = B + A.B'.C
  = B + A.C
```

De keuze van de volgorde van samen nemen van termen is van grote invloed op het gemak waarmee het eindresultaat bereikt wordt.
Breng als oefening eerst de A buiten haakjes volgens

```
F     = A'.(B.C'+ B.C) + A.(B'.C + B.C' + B.C)
```

en maak de vereenvoudiging zelf verder af.

3.1.2 Waarheidstabellen

In plaats van het specificeren van een functie met behulp van een formule kan dit ook gedaan worden met een tabel. De tabel geeft voor alle mogelijke combinaties van ingangswaarden de functiewaarde.

Voorbeeld:

De functie F = A.B kan ook worden beschreven met de tabel:

ingangsvariabelen uitgang

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Als er n variabelen zijn is de lengte van de tabel dus 2^n. Als een functie gegeven is in de vorm van een formule, en men wil de waarheidstabel opstellen, dan kan dit natuurlijk door alle combinaties van 0 en 1 in de formule te substitueren en de functiewaarde te bepalen. Dit kan een tijdrovend werk zijn. Daarom hier nog een andere methode aan hand van een voorbeeld:
Stel van de functie F = A + B'.C wil men de waarheidstabel vormen. Nu kan de formule zodanig herschreven worden dat in elke term van de formule alle variabelen voorkomen.

```
A + B'.C = A.(B + B').(C + C') + (A + A').B'.C
         = A.B.C + A.B.C' + A.B'.C + A.B'.C' + A.B'.C +
           A'.B'.C
         = A.B.C + A.B.C' + A.B'.C + A.B'.C' + A'.B'.C
```

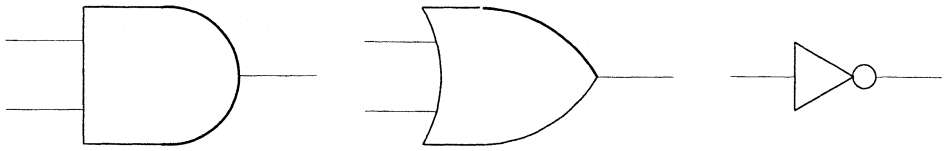
Elke term van deze formule zal juist bij één combinatie een 1 als waarde opleveren. Deze 1 kan op de betreffende regel van de tabel worden ingevuld. De resterende regels hebben als waarde 0.
Zulke termen waarin elke variabele voorkomt worden de mintermen genoemd.
Omgekeerd, als de tabel gegeven is, kan de formule worden samengesteld door de termen waar in de tabel een 1 als uitgangswaarde staat, met "+" tekens ertussen achter elkaar te zetten.

A	B	C	F		F =
0	0	0	0		
0	0	1	1	=>	A'.B'.C
0	1	0	0		
0	1	1	0		
1	0	0	1	=>	+ A.B'.C'
1	0	1	1	=>	+ A.B'.C
1	1	0	1	=>	+ A.B.C'
1	1	1	1	=>	+ A.B.C

Er zijn in een formule dus evenveel mintermen als er enen zijn in de waarheidstabel.

3.2 Poorten

Nu de begrippen AND, OR en NOT bekend zijn kan er gekeken worden naar de componenten die de bewerkingen kunnen uitvoeren. Deze blokjes worden poorten genoemd. De bijbehorende grafische symbolen zijn hieronder aangegeven.

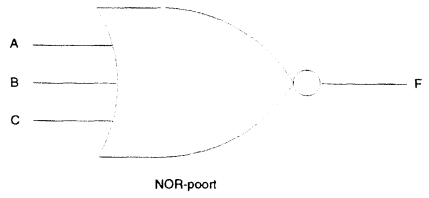


AND OR NOT

3.2.1 Elementaire poorten

De geïntroduceerde blokjes AND-poort, OR-poort en INVERTER (de meer gebruikelijke naam voor de NOT-poort) zijn voldoende om elke gewenste digitale schakeling samen te stellen. Toch zijn voor het gemak nog een aantal variaties toegevoegd. Zo zijn er nog de geïnverteerde versies van de AND- en OR-poort, de NAND-poort en de NOR-poort. Verder is het aantal ingangen van een poort niet altijd 2, maar kan ook 3 tot 8 zijn. Vindt men dit nog niet genoeg dan zijn er nog uitbreidbare poorten. Verder is de EXCLUSIVE-OR-poort nog beschikbaar, met als formule F = A ⊕ B of in de andere operatoren uitgedrukt F = A.B'+ A'.B.

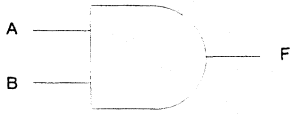
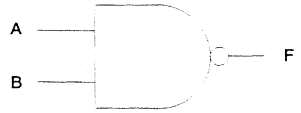
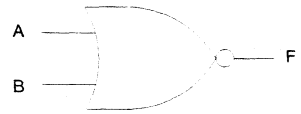
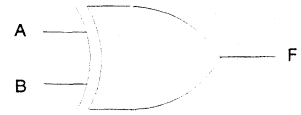
Voorbeeld van een poort met 3 ingangen:



F = (A + B + C)' = A'.B'.C'

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

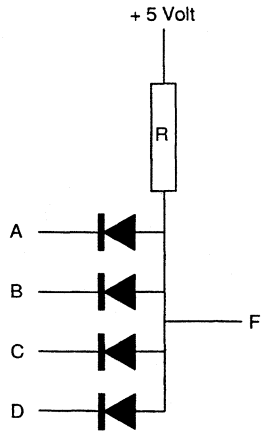
Welke combinaties van functies en aantal variabelen gebruikt kunnen worden is in de verschillend databoeken van chip-fabrikanten terug te vinden.

Naam	Symbool	Functie	Waarheids tabel
AND	 AND-poort	F = A.B	A B F 0 0 0 0 1 0 1 0 0 1 1 1
OR	 OR-poort	F = A+B	A B F 0 0 0 0 1 1 1 0 1 1 1 1
NAND	 NAND-poort	F = (A.B)'	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR	 NOR-poort	F = (A+B)'	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR	 EXCLUSIVE-OR poort	F = A ⊕ B of F = A.B'+A'.B	A B F 0 0 0 0 1 1 1 0 1 1 1 0
NOT	 INVERTER	F = A'	A F 0 1 1 0

Overzicht van de meest bekende poorten met 2 (1) ingangen.

3.2.2 De electronica van de poort

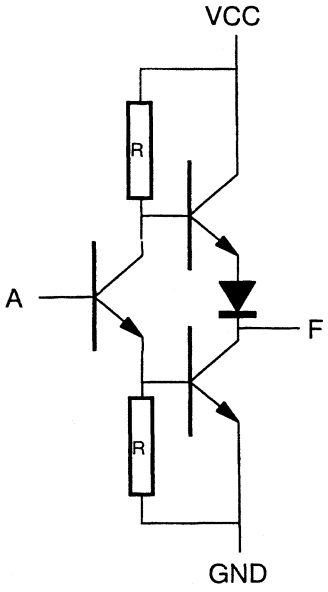
De werking van de AND-poort op het electronische niveau kan verklaard worden met het schema in Afbeelding 17. Voor de electrische spanningen geldt hier de afspraak dat de 0 door een lage spanning, en de 1 door een hoge spanning wordt gerepresenteerd. Als de ingangen A, B, C en D allemaal op een hoge spanning staan, zullen alle diodes gesperd zijn en loopt er dus geen stroom. Er is dan geen spanningsval over de weerstand en de uitgangsspanning is dus ook hoog. Maar als er slechts één ingang een lage spanning krijgt aangeboden, dan kan er een stroom door de bijbehorende diode gaan lopen. De spanningsval over de weerstand zorgt nu voor een lage spanning aan de uitgang. Dus één 0 aan de ingang geeft al een 0 aan de uitgang, alleen als alle ingangen 1 zijn zal de uitgang 1 zijn. Dit is de werking van de AND-gate. Om de werking van de OR-gate te krijgen kunnen de diodes worden omgedraaid en de weerstand met de lage spanning worden verbonden. In de praktijk zullen ook de diodes spanningsval hebben. Het gevolg is dat als er een aantal van dit soort poorten achter elkaar geschakeld worden de spanning steeds slechter de waarden voor 0 en 1 benadert. Daarom wordt dan achter zo'n trap een versterker geschakeld die de spanning weer op het gedefinieerde niveau brengt. Deze uitvoering wordt RTL (Resistor Transistor Logic) genoemd.



AND-poort

Afbeelding 17

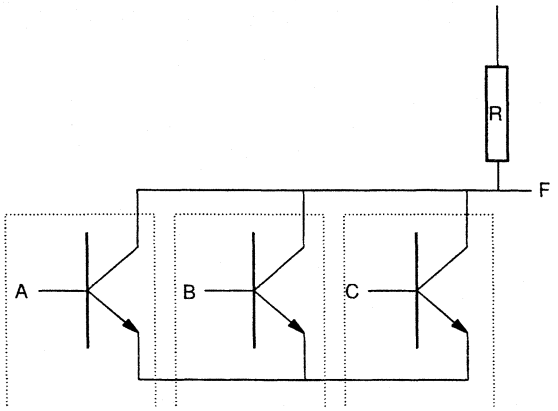
Een betere methode is de TTL (Transistor Transistor Logic) uitvoering. Hierbij wordt de uitgangsspanning niet bepaald door een weerstand, maar ingesteld door 2 transistoren in serie. In nevenstaande Afbeelding 18 is deze manier van schakelen toegepast in de realisatie van een inverter. De uitgangsschakeling wordt een totem-pole genoemd. Als de spanning bij A hoog is, dan wordt de onderste transistor van de totem-pole open en de bovenste transistor dicht gezet. Dus wordt de uitgang bijna de laagste spanning. F is dus een 0. Als de spanning bij A laag is, dan wordt juist de onderste transistor van de totem-pole dicht gezet en de bovenste komt in geleiding. De uitgangsspanning zal dan hoog zijn, wat voor F een 1 betekent. Ook van deze TTL-bouwstenen zijn nog verschillende uitvoeringen. De vertragingstijd van een TTL-poort loopt van 5 tot 20 nanosec. (nano = 10^{-9}).



TTL-inverter

Afbeelding 18

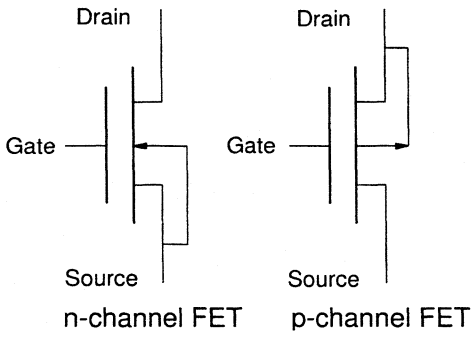
De RTL techniek wordt nog wel gebruikt in schakelingen waarbij in de uitgangstrap de bovenste transistor ontbreekt. Men spreekt dan van een open-collector output. De uitgangen van dit soort blokjes kan direct aan elkaar gehangen worden. Als één of meer uitgangen 0 zijn zal het totaalresultaat bij F ook 0 zijn. Door de verbinding van de uitgangen ontstaat dus een AND-functie. Deze constructie wordt wired-or genoemd.



wired-OR

Afbeelding 19

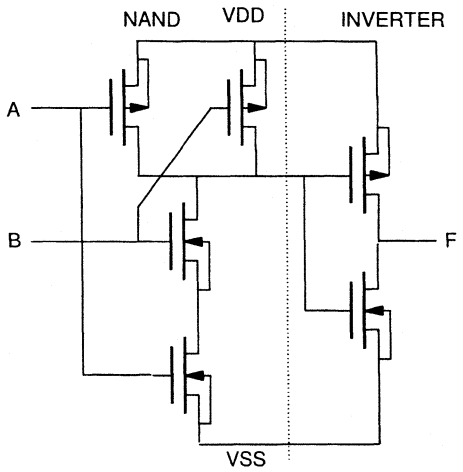
Een meer energiezuinige poort kan gemaakt worden door de MOS-transistoren te gebruiken. MOS staat voor Metal Oxide Silicon. Als de spanning aan de drain, Vdd, positief is t.o.v de source waar de spanning Vss is aangesloten, dan kunnen de n-channel FET en de p-channel FET als schakelaars werken. Bij de n-channel FET is de schakelaar gesloten (maakt verbinding) als de gate spanning in de buurt van de drain spanning komt. De p-channel FET vormt een gesloten schakelaar als de gate spanning in de buurt van de source spanning komt. Het bedienen van de schakelaars kost bijzonder weinig energie. Door nu de n-channel en de p-channel FET in paren te gebruiken is altijd maar één tak gesloten, zodat in de uitgangstrap bijna geen stroom loopt. Het energieverbruik kan daarom erg laag blijven. De technologie van het toepassen zowel n-channel FETs als p-channel FETs heet CMOS (voor Complementary MOS).



MOS transistoren

Afbeelding 20

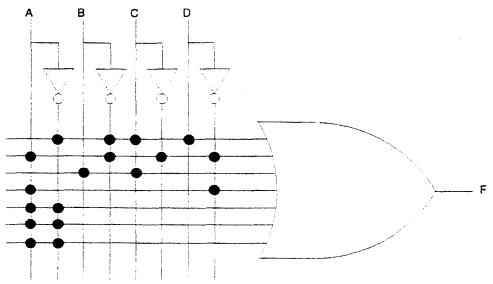
In nevenstaand schema is de realisatie van een AND-poort te zien. Als zowel A als B hoog zijn, dan is de serieschakeling van n-channel FETs gesloten zodat op de gates van de schakeling rechts van de stippellijn een lage spanning komt. Van deze laatste tak wordt de p-channel FET geleidend en wordt de uitgang dus hoog. Voor de stippellijn is een NAND-poort opgebouwd, die samen met de INVERTER-functie, rechts van de stippellijn, een AND-functie vormt.



AND-poort in CMOS

Afbeelding 21

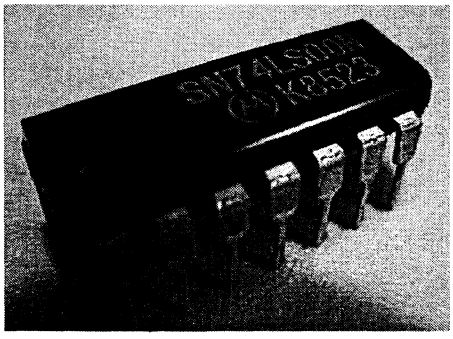
3.2.3 Programmeerbare logica



Afbeelding 22 PAL

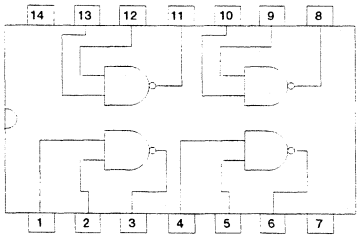
De ontwerper heeft naast de reeks standaardpoorten ook programmeerbare poorten tot zijn beschikking. In de Afbeelding 22 is het principe van een z.g.n. PAL aangegeven. PAL is de afkorting van Programmed Array Logic. Op elke horizontale lijn kan een productterm gevormd worden door een variabele of zijn inverse te verbinden met een soort wired-or (= AND) constructie. Het aanbrengen van de verbindingen, het programmeren, gebeurt electrisch en is bij sommige typen onherroepelijk en bij andere typen wisbaar en daarna herprogrammeerbaar. Het laatste type heet ook wel GAL. In Afbeelding 22 is de formule F = A'.B'.C.D + A.B'.C'.D' + B.C + A.D' samengesteld. Een ander type programmeerbare bouwsteen is de PLA wat staat voor Programmable Logic Arrays. Bij dit type bouwsteen zijn twee programmeerbare matrices. Niet alleen de producttermen zijn samen te stellen maar ook de somterm is door de ontwerper te programmeren. In de PAL is het aantal termen wat op de OR-poort staat vast.

3.2.4 Uitvoering

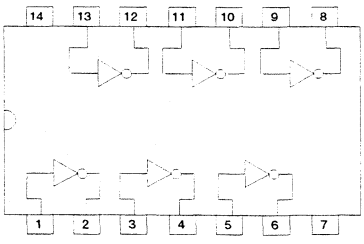


Afbeelding 23 chip

Een veel gebruikte uitvoering waarin poorten geleverd worden is het dual-in-line package. Het plastic huisje is voorzien van 2 rijen aansluitpennen. Het aantal pennen van zo'n chip loopt van 2*4 tot 2*32. Bij poorten is dit aantal meestal 14 of 16. Een veel gebruikte TTL familie van Integrated Cicuits of afgekort I.C.'s is de 74xx reeks. De voedinsspanning van deze I.C.'s moet 5 volt zijn. Een meer energiezuinige familie is de 40xx reeks die in MOS is uitgevoerd. Hier kan de spanning gekozen worden van 5 volt tot 15 volt. (De I.C.'s van één schakeling moeten onderling wel dezelfde voedigsspanning hebben). Een MOS familie met de typenummers 74HCTxx zijn energiezuinig en pin-compatible met de TTL 74xx reeks.
Twee voorbeelden van TTL I.C.'s:



SN7400

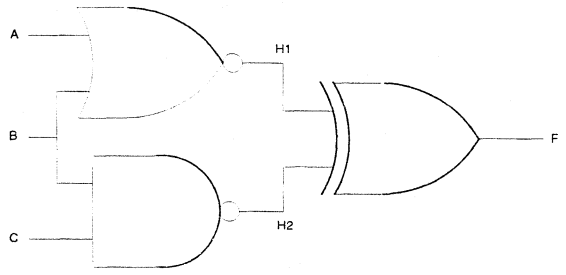


SN7404

Op de pennen 7 en 14 wordt de voedingsspanning aangesloten. Pin 7 is ground en pin 14 is +5 volt.

3.3 Schakelingen met poorten

De schakelingen die nu volgen zijn samengesteld uit poorten, en hebben bovendien de eigenschap dat de uitgangen volkomen bepaald zijn door de waarden van de ingangen. Dit zijn de z.g.n combinatorische schakelingen.
De ontwerper zal in het algemeen geconfronteerd worden met het probleem van de synthese van een poortschakeling. Maar hier wordt eerst aandacht aan de analyse van zo'n poortschakeling geschonken.
Vraag hoe ziet de funtie F(A,B,C) van de schakeling in Afbeelding 25 eruit.
Formule-methode:
Bepaal de formules op de tussenliggende punten (hier H1 en H2) en daarna de uitgang.



Afbeelding 25

```
H1 = (A + B)'
H2 = (B.C)'
 F = (A + B)'.((B.C)')' + ((A + B)')'. (B.C)'
   = (A + B)'.B.C + (A + B).(B'+ C')
   = A'.B'.B.C + A.B'+ A.C'+ B.C'
   = 0 + A.B' + A.B'.C' + A.B.C' + B.C'
   = A.B' + B.C'
```

In bovenstaande vereenvoudiging is de term A.C' gesplitst in A.B.C' + A.B'.C om vervolgens door tweemaal het toepassen van de absorptie-wet geheel weg te vallen.
Tabelmethode:
Maak een waarheidstabel met de ingangsvariabelen A,B en C met achtereenvolgens als uitgangstabellen H1,H2 en F. Neem de mintermen die bij F 1 zijn samen. Neem vervolgen deze termen samen.

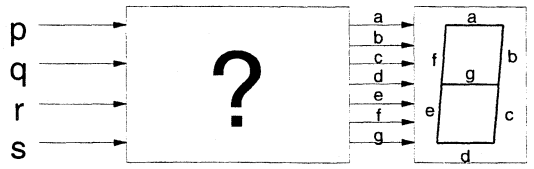
A	B	C	H1	H2	F		
0	0	0	1	1	0		
0	0	1	1	1	0		
0	1	0	0	1	1	=>	A'.B.C'
0	1	1	0	0	0		
1	0	0	0	1	1	=>	A.B'.C'
1	0	1	0	1	1	=>	A.B'.C
1	1	0	0	1	1	=>	A.B.C'
1	1	1	0	0	0		

```
F = A'.B.C'+ A.B'.C'+ A.B'.C + A.B.C'    anders rangschikken
  = A'.B.C'+ A.B.C' + A.B'.C'+ A.B'.C
  = B.C'+ A.B'
```

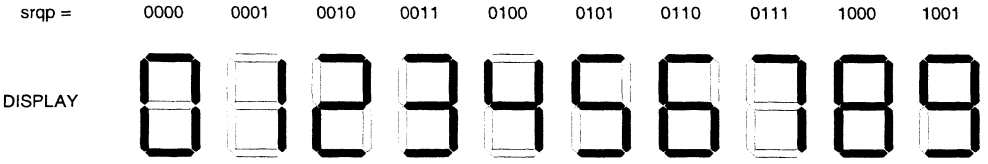
Het splitsen en anders rangschikken om termen te kunnen samennemen is soms wat puzzel-werk.

3.3.1 Code omzetters

Bij de synthese van schakelingen komt er t.o.v. de analyse van schakelingen nog een probleem bij, nl het probleem van de keuze van de typen poorten die gebruikt zullen gaan worden. Als voorbeeld het ontwerp van de codeomzetter voor het aansturen van een 7-segments display. Als ingangssignalen zijn gegeven s,r,q en p die een zuiver binaire code aanbieden aan de te ontwerpen poortschakeling. Bij de combinaties 0000 t/m 1001 moet het bijbehorend cijfer worden gevormd. Er moet dus voor elk der 7 segmenten een signaal worden gevormd dat aangeeft of het betreffende segment brandt of niet. Het betreft hier dus alleen het stuursignaal. Het hangt van de uitvoering van het display af hoe het betreffende stuursignaal moet worden omgevormd tot het daadwerkelijk laten branden van het segment.



Afbeelding 26 7-segment codering



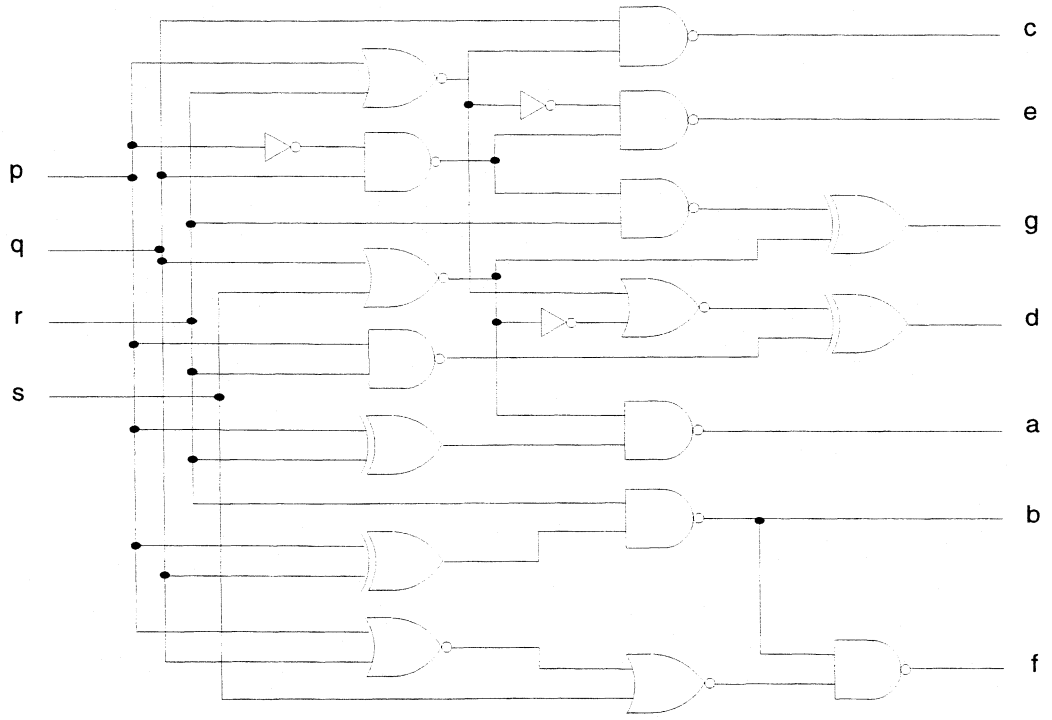
Allereerst wordt er een waarheistabel gemaakt met de 10 voorgeschreven combinaties en nog 6 combinatie (11 t/m 15) waarvan ons het gedrag niet interesseert.

s	r	q	p	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	–	–	–	–	–	–	–
1	0	1	1	–	–	–	–	–	–	–
1	1	0	0	–	–	–	–	–	–	–
1	1	0	1	–	–	–	–	–	–	–
1	1	1	0	–	–	–	–	–	–	–
1	1	1	1	–	–	–	–	–	–	–

Hieruit kunnen de 7 functies voor de aansturing van de segmenten worden afgeleid.

```
a = (p ⊕ r)'+ (q + s)
b = (p ⊕ q)' + r'
c = (p + r) + q'
d = q'.s'.(p + r) ⊕ (p' + r')
e = p'.r'+ p'.q
f = r.(p ⊕ q) + (s + p'.q')
g = q'.s' ⊕ (r' + p'.q)
```

Deze formules moeten in poorten worden vertaald. Een extra gegeven hierbij is dat gelijke uitdrukkingen die in twee of meer formules voorkomen slechts éénmaal behoeven te worden gemaakt. Er moet dus geprobeerd worden in de formules gelijke stukken te krijgen. Dit is puzzelwerk. Bovenstaande formules zijn al zodanig voorbewerkt dat de gemeenschappelijke termen p'.r', p + q' en q'.s' of hun inversen te herkennen zijn.

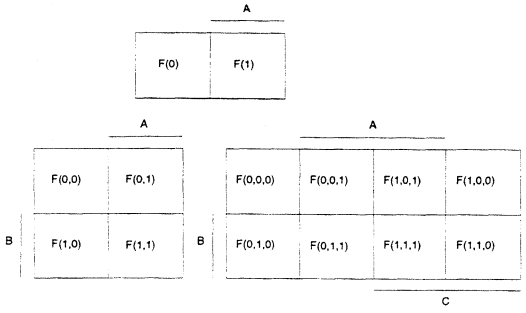


Afbeelding 28 7-segments display aansturing

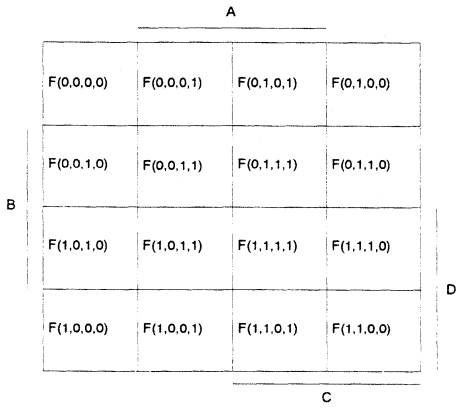
De schakeling kan met minder poorten worden opgebouwd. Er geldt de betrekking a = d + q waardoor een EXCLUSIVE OR-poort en een NAND-poort door 1 OR-poort kan worden vervangen, maar dan is de verdeling over de typen poorten slechter. Nu zijn er 8 NAND's, 5 NOR's, 4 XOR's en 3 INVERTERS nodig. Als voor de 3 INVERTERS 3 NOR's (waarvan de ingangen met elkaar verbonden zijn) gebruikt worden, dan kan de gehele schakeling met 5 i.c. blokjes worden uitgevoerd.
De 6 niet voorkomende ingangscombinaties 1001 t/m 1111 zijn gebruikt als don't cares d.w z dat het gedrag van de uitgangen daar niet is voorgeschreven. De invulling is gekozen op grond van zo eenvoudig mogelijk te realiseren formules. Als men deze combinaties toch aan de schakeling aanbiedt komen er ongedefinieerde patronen uit.
De totale schakeling is ook in 1 i.c blokje verkrijgbaar bijvoorbeeld onder codenummer 7448.

3.3.2 Karnaugh map

Tot nog toe zijn twee methoden gebruikt om een functie weer te geven nl de formule of de waarheidstabel. Beide beschrijvingsmethoden zijn voor menselijk gebruik niet zo geschikt als de functie vereenvoudigd moet worden. Met de Boole algebra is een formule te herschrijven, maar het overzicht welke termen moeten worden samengenomen is niet aanwezig. Daarom is er een grafische methode ontwikkeld, die erop berust dat twee mintermen die slechts in één variabele van elkaar verschillen, als buren worden weergegeven. Bij een functie van 4 variabelen heeft elke minterm (term waar elke variabele in voorkomt) dus 4 buren.



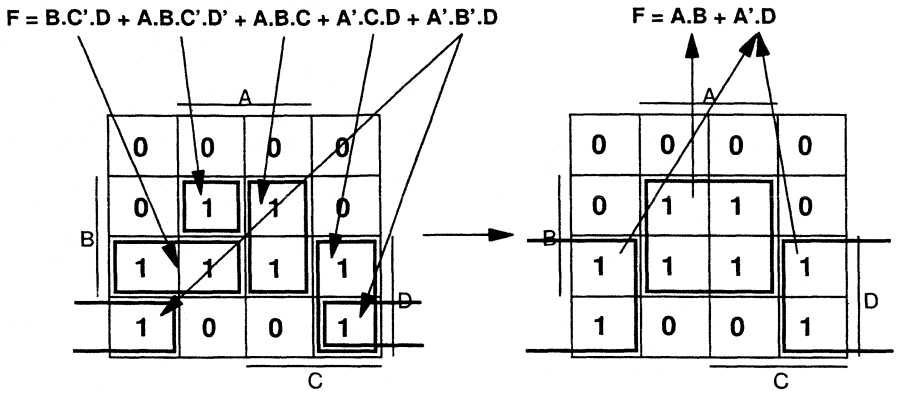
Afbeelding 29 Karnaugh-maps voor 1,2 en 3 variabelen



Afbeelding 30 Karnaugh-map voor 4 variabelen

In Afbeelding 29 is dit voor 1,2 en 3 variabelem en in Afbeelding 30 voor 4 variabelen aangegeven. De hokjes aan de rand zijn aan de buitenzijde aangrenzend met de hokjes aan de tegenovergestelde rand. Men moet de randen rondgevouwen denken tot een binnenband (of wiskundig tot een torus). Als de Boole algebra zegt dat A'.B.C.D'+ A'.B'.C.D'= A'.C.D' dus dat voor twee termen die slechts in één variabele van elkaar verschillen één nieuwe term zonder die variabele geschreve kan worden, dan betekent dit in de Karnaughmap twee aangrenzend vakjes samengenomen kunne worden tot een (dubbel)vakje. Zo moet men voor een zo eenvoudig mogelijke functiebeschrijving proberen alle enen uit de map te "vangen" in een zo klein mogelijk aantal vakken bestaande uit 1, 2, 4, 8 of zelfs 16 kleine vakjes. Aan de randen van de Karnaugh map staat het gebied aan gegeven waar de betreffende variabele 1 is. Om bijvoorbeeld van een F(D,C,B,A) de term B.C'.D te plaatsen zoeken we naar het gebied binnen de "streep" B, buiten de "streep" C en binnen de "streep" D en vinden dan een vak bestaande uit de vakjes F(1,0,1,0) en F(1,0,1,1).
Een voorbeeld van een vereenvoudiging is in Afbeelding 31 aangegeven. Eerst worden alle termen van de te vereenvoudigen formule in de map ondergebracht door alle vakken, die door de termen van de functie worden bepaald, 1 te maken. De resterende vakken worden 0 gemaakt. Vervolgens worden de enen samengenomen tot een zo klein mogelijk aantal vakken (die op zich zo groot mogelijk zijn). Vertaal de vakken weer in termen van de nu vereenvoudigde formule.

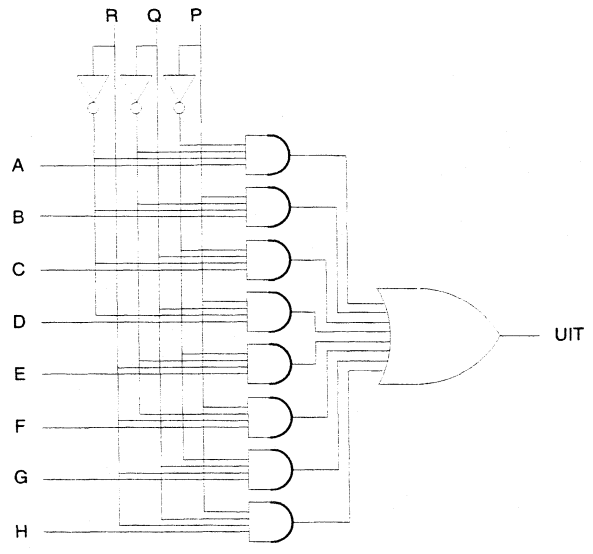
Als de invulling vanuit een formule plaatsvindt zullen er uitsluitend nullen en enen in de map komen. Wordt echter een waarheidstabel overgebracht naar een Karnaughmap, dan kunnen daar ook - of don't cares in voorkomen. Deze don't care vakjes kunnen gebruikt worden om de vakken bij het samennemen zo groot mogelijk te krijgen. De don't care vakjes mogen zowel 0 als 1 gekozen worden en alleen als het voor het samennemen goed uitkomt, worden ze in zo'n term opgenomen.



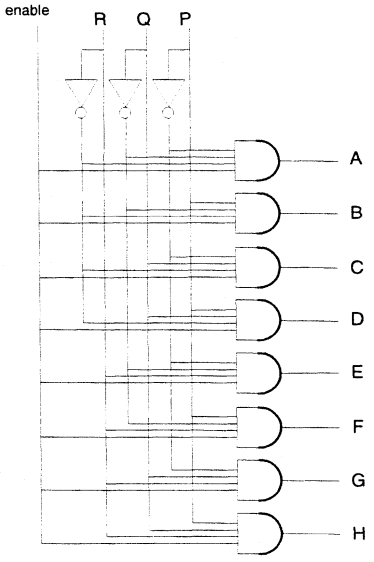
Afbeelding 31 voorbeeld

3.3.3 Computerschakelingen

Nu volgen een aantal voorbeelden van combinatorische schakelingen die veel in computers voorkomen.



Afbeelding 32 selector of multiplexer



Afbeelding 33 decoder of demultiplexer

Een schakeling waarmee het mogelijk is om een keuze te maken uit een aantal signalen wordt selector of multiplexer genoemd. In Afbeelding 32 is te zien dat 3 ingangen van elke EN-poort bestuurd worden door een combinatie van de signalen P,Q en R. Bij slechts één van de 8 EN-poorten kunnen de 3 stuursignalen 1 zijn. De andere 7 poorten hebben de uitgangswaarde 0, terwijl de geselecteerde poort de waarde van het bijbehorende ingangssignaal zal doorgeven.

In formule:

UIT = A.P'.Q'.R' + B.P.Q'.R' + C.P'.Q.R' + D.P.Q.R' + E.P'.Q'.R + F.P.Q'.R + G.P'.Q.R + H.P.Q.R

Een soortgelijke schakeling is de decoder of demultiplexer. Met deze schakeling is het mogelijk om een item aan te wijzen. Hier geeft, bij alle mogelijke combinaties van P,Q en R, één van de uitgangen het signaal enable door. Welke uitgang dat is wordt bepaald door de waarden van P,Q en R.
In formule:

A = P'.Q'.R'.enable B = P.Q'.R'.enable enz.

Beide systemen zijn uitbreidbaar. Wil men een selector voor 64 ingangen construeren dan kan dit met 9 van de getekende selectoren. De 64 ingangssignalen worden aangesloten aan 8 van de getekende selectoren die allen bestuurd worden door de signalen P,Q en R. Hierdoor ontstaan 8 UIT-signalen. Met behulp van de 9^e selector kan, onder besturing van 3 nieuwe stuursignalen S,T en U, hieruit een keuze worden gemaakt. Er kan dan m.b.v. 6 stuursignalen P,Q,R,S,T en U een keuze uit 64 ingangssignalen gemaakt worden. Evenzo kan een demultiplexer met 64 uitgangen gemaakt worden met 9 van de getekende demultiplexers door elk van de 8 uitgangen van 1 demultiplexer te laten fungeren als enablesignaal voor 8 daarop volgende demultiplexers. Ook hier zijn dan 6 stuursignalen nodig voor de aanwijzing.

Een ander voorbeeld van een combinatorische schakeling is de opteller van twee cijfers in het tweetallige stelsel. De op te tellen cijfers zijn 0 en 1, waarbij nog rekening gehouden moet worden met een eventuele carry (1 onthouden) uit de vorige kolom. Dus als A en B cijfers op overeenkomstige plaatsen van de twee op te tellen getallen zijn, dan geldt:

zonder carry

	+	A = 0	A = 1
B	0	0	1
	1	1	0*

met carry

	+	A = 0	A = 1
B	0	1	0*
	1	0*	1*

Hierbij geeft het sterretje aan dat er een overdracht naar de volgende kolom is.
Deze gegevens kunnen overgebracht worden naar een waarheidstabel:

C_{in}	B	A	C_{uit}	SOM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

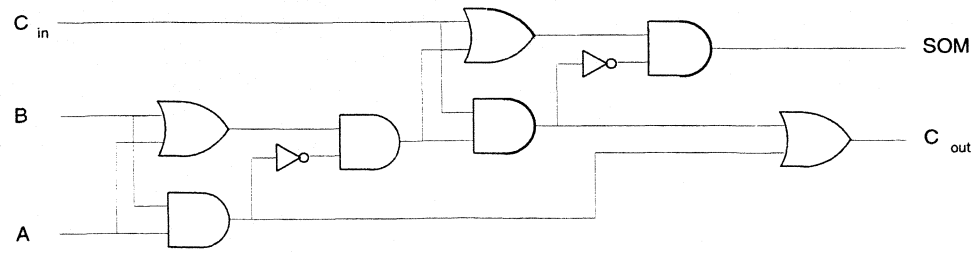
Of in formulevorm:

$C_{uit} = A.B + C_{in}.(A + B)$ maar ook $C_{uit} = A.B + C_{in}.(A \oplus B)$

en

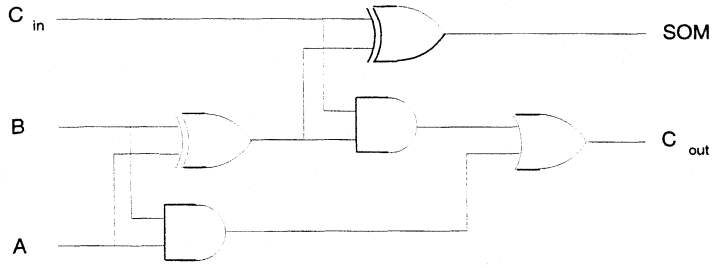
$SOM = (A \oplus B) \oplus C$

Omgezet in AND's , OR's en INVERTOREN levert dit:



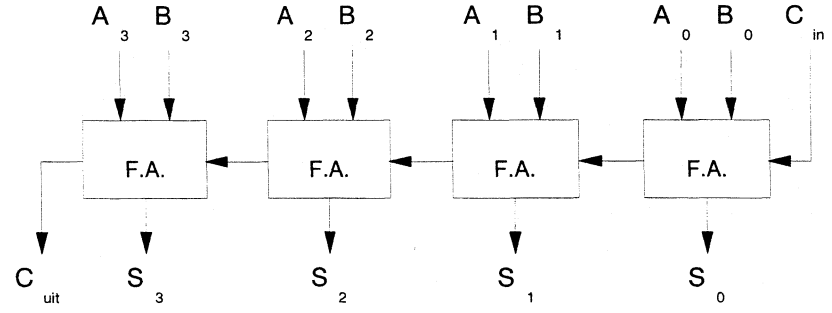
Afbeelding 34 Full Adder 1

Als ook EXCLUSIVE-OR poorten gebruikt worden wordt het iets eenvoudiger:



Afbeelding 35 Full Adder 2

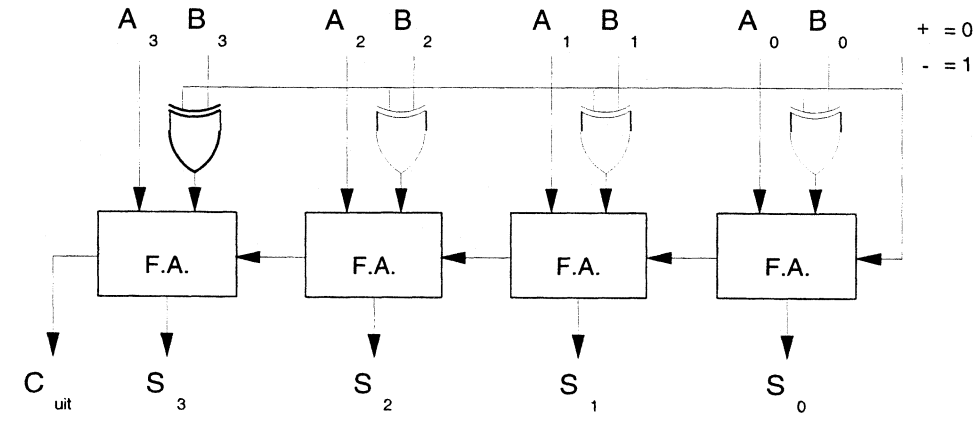
Om getallen van n bits bij elkaar op te tellen zijn dus n van deze sectie nodig. De uitgaande carry van een sectie moet verbonden worden met de ingaande carry van de sectie één cijferplaats hoger. Hier is het aangegeven voor 4 cijfers.



Afbeelding 36 opteller voor 4 bits

Dit soort blokjes zijn in TTL uitvoering verkrijgbaar onder typenummer 7483.
Om grotere getallen op te tellen kunnen de uitgaande carry's weer verbonden worden met de ingaande carry's van de volgende blokjes.

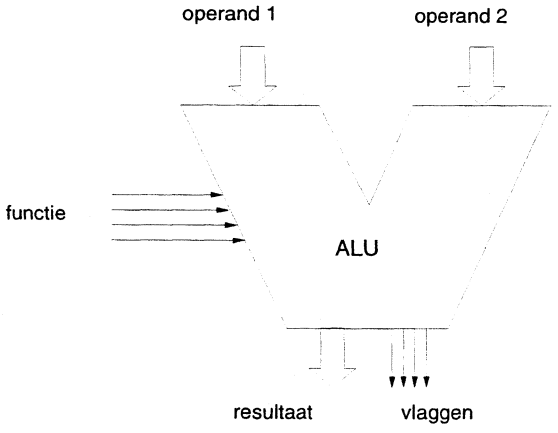
Om ook getallen van elkaar af te kunnen trekken wordt de schakeling voorzien van een aantal EXCLUSIVE-OR poorten. Immers om een getal ergens van af te trekken kan men het two's complement van dat getal erbij optellen. Two's complement betekent alle bits van het getal inverteren en er 1 bij optellen. Het inverteren gebeurt met de EXCLUSIVE-OR poorten. Wordt op de stuuringang een 0 gezet, dan wordt het getal gewoon doorgelaten. Maar staat er een 1 op de stuuringang, dan wordt het bit geïnverteerd doorgegeven. Door nu de stuuringang ook met de ingaande carryingang te verbinden wordt er al of niet 1 bij opgeteld.
Met de volgende schakeling kan dus A + B en A - B berekend worden.



Afbeelding 37 opteller/aftrekker

Ook hier is de schakeling uitbreidbaar door de uitgaande carry te verbinden met de ingaande carry van het volgende blokje.

Naast optellen en aftrekken is er ook behoefte om de logische bewerkingen zals AND OR en NOT woordbreed te kunnen uitvoeren. Met wat meer poorten zijn een aantal logische en aritmetische bewerkingen binnen één schakeling te brengen. Deze schakeling wordt ALU genoemd, wat een afkorting is voor Arithmetic and Logic Unit. Als symbool hiervoor wordt vaak de tekening in Afbeelding 38 gebruikt.



Arithmetic and Logic Unit

Afbeelding 38

In de TTL-reeks is zo'n blokje met het nummer 74181. In Afbeelding 39 is dit aangegeven. Met S0,S1,S2 en S3 is een keuze te maken uit 16 verschillende bewerkingen op de getallen A en B. Met de stuuringang M is te kiezen tussen de aritmetische groep met 16 bewerkingen of de logische groep met 16 bewerkingen. Bovendien wordt nog op gelijkheid getest. Het resultaat is het getal F. Met Cn als carryingang en Cn+4 als uitgaande carry zijn weer grotere bewerkingseenheden samen te stellen. De verwerkingstijd wordt bepaald door het langste pad in de schakeling. Dit zal bij grote eenheden ongetwijfeld de carrylijn zijn, want dit signaal loopt door alle samenstellende blokjes. Om de tijd voor de carrydoorloop te verkorten zijn op elk blokje nog een Propagate en een Generate uitgang aangebracht. De generate-uitgang geeft aan of binnen het subblok zelfstandig een carry wordt opgewekt. (dus onafhankelijk van de ingaande carry). De propagate-uitgang bepaalt per blok of een binnenkomende carry doorgegeven moet worden of niet. Dit signaal kan een carrypad buiten het blok besturen. Daardoor is het mogelijk de carrylooptijd niet lineair, maar logaritmisch met het aantal bits te laten toenemen. Er zijn dan nog wel extra circuits nodig.
De meeste kleine computers hebben als bewerkingsblok voldoende aan een 8 of 16 bits ALU.

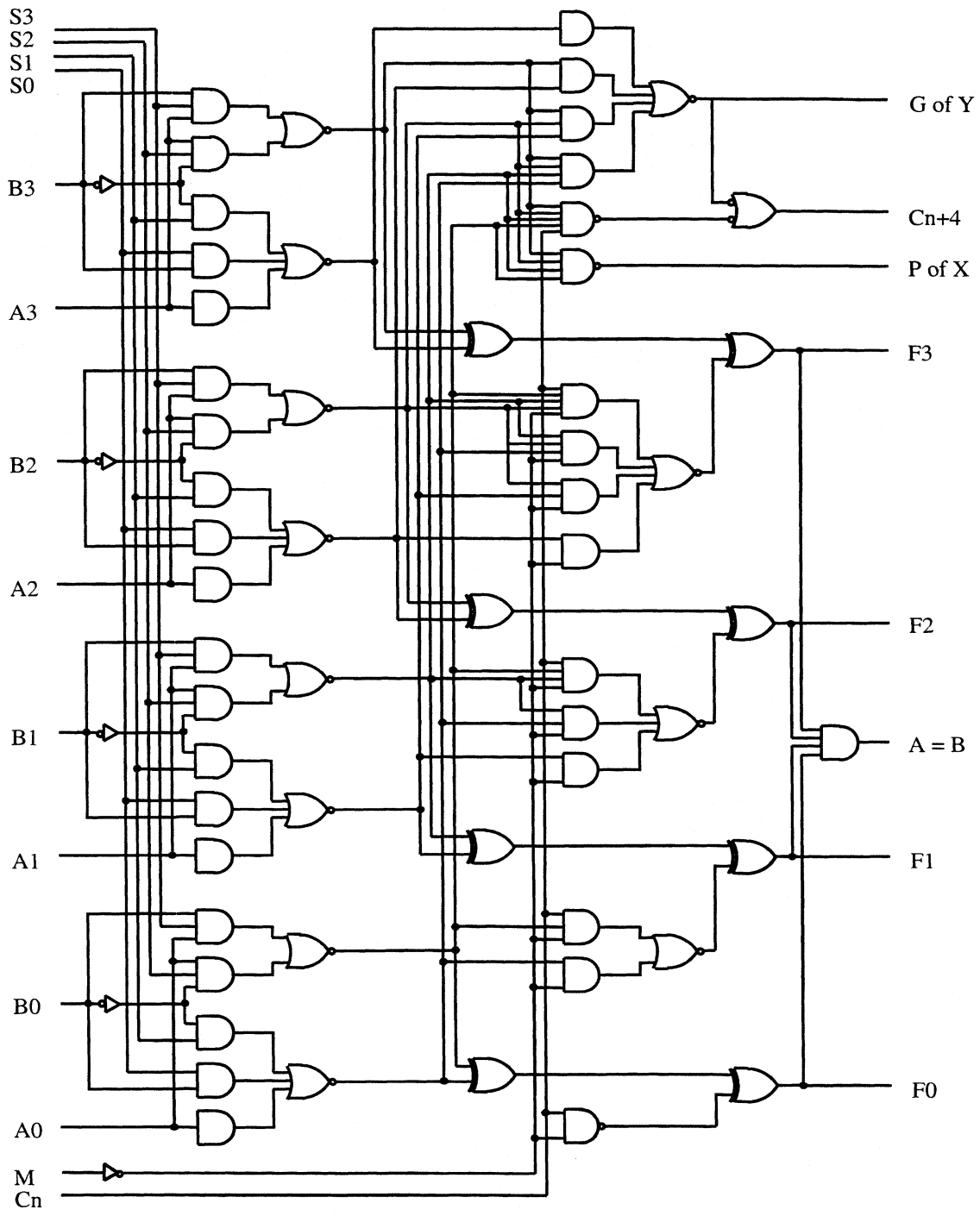
Naast het bepalen van het antwoord van een logische of rekenkundige bewerking op 2 getallen heeft de ALU ook de taak om z.g.n status- of conditievlaggen te produceren. Het zijn antwoorden van 1 bit. Als de 2 te bewerken getallen, stel A en B, met elkaar vergeleken worden ontstaan de vlaggen A=B en A>B. De vlag A=B is in het ALU-schema al voorbereid, maar om dit voor een bijvoorbeeld 16-bits bewerking vast te stellen moet nog extra hardware worden toegevoegd. Dit is ook het geval voor de vlag A>B waarbij de hoogstwaardige unit een belangrijke rol vervult.
Een vlag die ook pas aan de "voorkant" of hoogstwaardige deel van het getal bepaald kan worden is het begrip "overflow". Deze vlag geeft aan dat het antwoord van de bewerking te groot is om in het beschikbare aantal bits weer te geven. Bij enkel positieve getallen is dit makkelijk te zien door naar de uitgaande carry te kijken. Is er nog een carry naar een plaats waar geen bits meer zijn dan is er overflow. Maar als er in een two's complementsysteem gewerkt wordt is overflow moeilijker te bepalen. In dat geval kunnen de tekenbits van A,B en resultaat R gebruikt worden. Als een positief getal en een negatief getal opgeteld worden kan er nooit overflow optreden. Als twee getallen met hetzelfde teken opgeteld worden is er overflow als het teken van het resultaat niet klopt met die van de opgetelde getallen. In tabelvorm:

A	B	R	conclusie
+	+	+	geen overflow
+	+	−	overflow
+	−	+	geen overflow
+	−	−	geen overflow
−	+	+	geen overflow
−	+	−	geen overflow
−	−	+	overflow
−	−	−	geen overflow

Het overflow-bit kan dus worden bepaald uit de tekenbits t_A,t_B en t_S met de formule:

$$overflow = t_A.t_B.t_S' + t_A'.t_B'.t_S$$

De meeste rekenunits gebruiken hiervoor echter de inkomende carry en de uitgaande carry op de plaats van het tekenbit.



Afbeelding 39 arithmetic and logic unit

De full-adder relaties op de plaats van het tekenbit zijn:

$$t_S = t_A \oplus t_B \oplus C_{in} \qquad ==> \qquad C_{in} = t_A \oplus t_B \oplus t_S$$

$$C_{uit} = t_A.t_B + (t_A + t_B).C_{in}$$

In een waarheidstabel geeft dit:

t_S	t_A	t_B	C_{in}	C_{uit}	
0	0	0	0	0	
0	0	1	1	1	
0	1	0	1	1	
0	1	1	0	1	overflow
1	0	0	1	0	overflow
1	0	1	0	0	
1	1	0	0	0	
1	1	1	1	1	

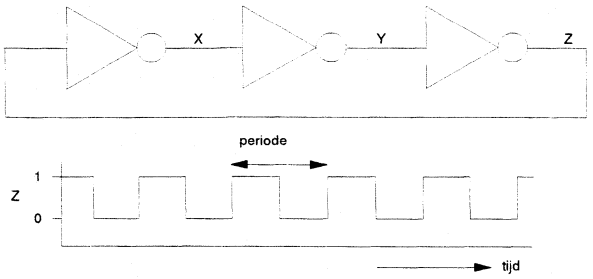
Voor het overflow-bit kan geschreven worden:

$$overflow = C_{in} \oplus C_{uit}$$

Bij de meeste rekenunits wordt de test op deze manier uitgevoerd.
Ook bij de nog te behandelen µ-controller 80C51 wordt de overflow-vlag gezet als op de plaats van het tekenbit de ingaande carry ongelijk is aan de uitgaande carry.

4 Sequentiële schakelingen

Poortschakelingen zonder terugkoppeling, de combinatorische schakelingen zoals in het vorige hoofdstuk behandeld, hebben de eigenschap dat de uitgangen volkomen bepaald zijn door de ingangen. Brengt men wel een terugkoppeling aan, dan komt hier verandering in.



Afbeelding 40 oscillerende schakeling

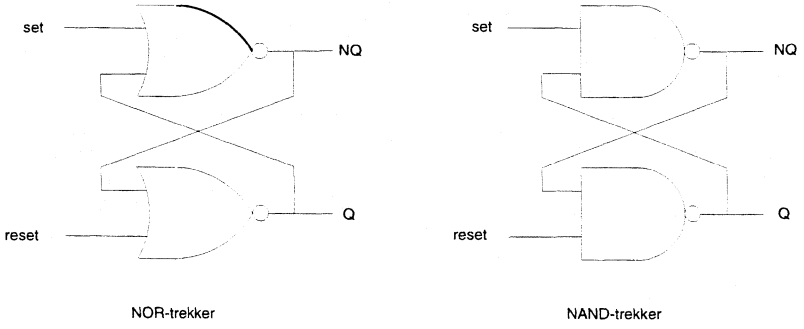
Voor de serieschakeling van 3 invertoren, in Afbeelding 40, geldt

$$Y = X' \text{ en } Z = Y' = (x')' = X$$

Wordt nu de terugkoppeling aangebracht, dan moet gelden X = X' hetgeen tot een zichzelf voortdurend corrigerende schakeling leidt. Vanaf het moment dat X een bepaalde waarde krijgt zal het ongeveer 3 poortvertragingen duren tot het tegendeel waar wordt. De schakeling zal gaan oscilleren met een periode van ongeveer 6 poortvertragingen.

4.1 Het geheugenelement

De oscillatie wordt veroorzaakt door een oneven aantal inversies in het terugkoppelpad. De nu volgende voorbeelden hebben een even aantal inversies in de lus.



Afbeelding 41 trekkers

De formules van de uitgangen NQ en Q van de NOR-trekker in Afbeelding 41 zijn:

```
NQ = set'.Q' = set'.(reset + NQ)   en
Q = reset'.NQ' = reset'.(set + Q)
```

Voor set = 0 en reset = 0 wordt dit NQ = NQ en Q = Q en Q = NQ'. De uitgangen worden niet door de ingangen, maar door de voorgeschiedenis bepaald.

De uitgangen van de NAND-trekker leveren op:

```
NQ = set'+ Q'  = set'+ reset.NQ    en
Q  = reset'+ NQ' = reset'+ set.Q
```

Hier zijn de formules voor set = 1 en reset = 1 niet oplosbaar.
Ook hier is dan de relatie NQ = NQ en Q = Q en Q = NQ'

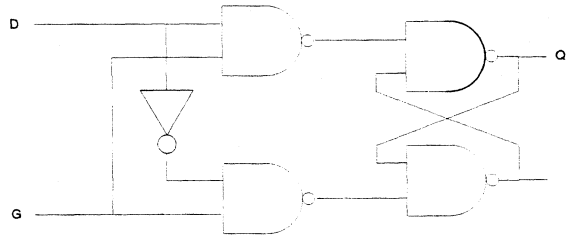
In de vorm van een waarheidstabel:

NOR-trekker			
set	reset	Q	NQ
0	0	Q	Q'
0	1	0	1
1	0	1	0
1	1	0	0

NAND-trekker			
set	reset	Q	NQ
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	Q'

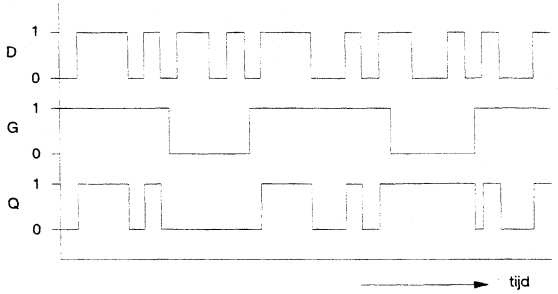
Voor de NAND-trekker geldt dus dat na de overgang van (set=0,reset=1) naar (set=1,reset=1) Q de waarde 0 zal onthouden, terwijl na de overgang van (set=1,reset=0) naar (set=1,reset=1) Q de waarde 1 behoudt. Wat er gebeurt na de overgang (set=0,reset=0) naar (set=1,reset=1) is onvoorspelbaar. Kleine verschillen in vertragingstijden bepalen dan of Q = 0 of Q = 1 gaat gelden.
Drie van de vier inganscombinaties zijn dus maar bruikbaar nl set-reset = 0-1 om de trekker te resetten of 0 te maken, de combinatie 1-0 om de trekker te setten of 1 te maken en de combinatie 1-1 als onthoudstand. Met behulp van 2 extra poorten kunnen deze ingangscombinaties effectief worden gebruikt.

In Afbeelding 42 is deze schakeling getekend. Is de stuuringang G = 0 dan wordt de trekker met 1-1 in de situatie "onthouden" gebracht. (G staat voor gate). Als G = 1 wordt met D = 0 via de ingangscombinatie 1-0 de trekkeruitgang Q op 1 gezet, en met D = 1 via de ingangscombinatie 0-1 de uitgang op 0 gezet. De ongewenste ingangscombinatie 0-0 op de trekker kan niet meer voorkomen.
(D staat hier voor data).



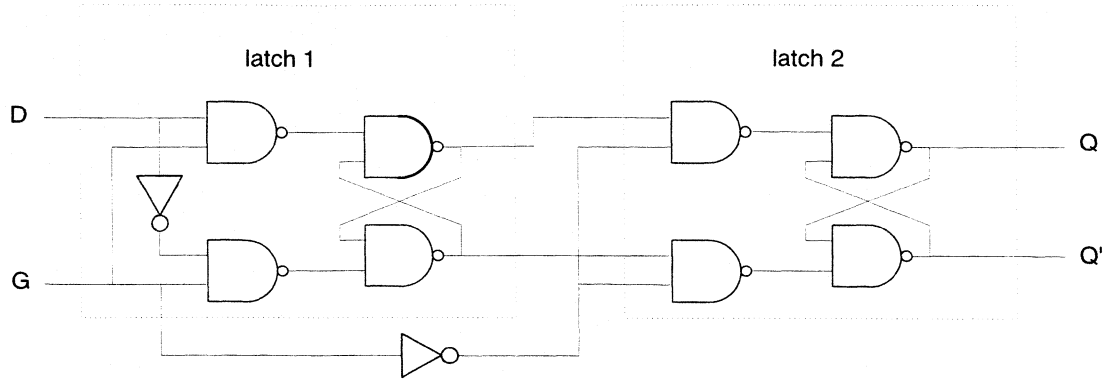
Afbeelding 42 latch

De formule voor de uitgang van de latch volgt uit de substituties SET = D'+ G' en RESET = D + G'.
Er geldt dan Q = G.D + G'.Q. Met G = 1 volgt de uitgang de waarde van de ingang D.
Met G = 0 wordt de waarde van D, op het moment van 0 worden van G, vastgehouden in de trekker.
Voorbeeld van dit timingsgedrag:



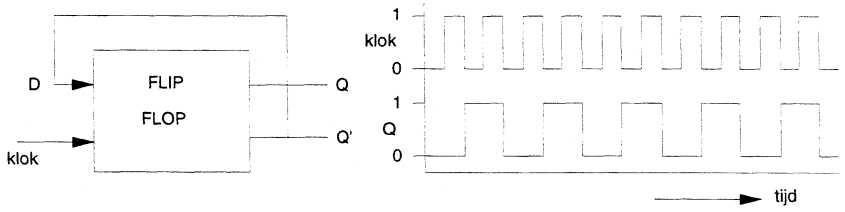
Afbeelding 43 Timing van de latch

Om de uitgang geheel te isoleren van de ingang maakt men gebruik van 2 latches volgens Afbeelding 44



Afbeelding 44 FLIP-FLOP opgebouwd uit latches

Met G = 1 wordt de waarde van D overgedragen aan latch 1, terwijl latch 2 is geblokkeerd. Wordt vervolgens G = 0 gemaakt dan is latch 1 geblokkeerd, en de waarde wordt aan latch 2 doorgegeven. Het is een soort schutsluis voor de data. Er is nooit een doorgaande verbinding van D naar Q. Dit soort geheugen wordt een Master en Slave geheugenelement of een flipflop genoemd. Om het systeem te laten werken moet G achtereenvolgens 1 worden om de data op te nemen en 0 om de uitgang zijn waarde te geven. Het achtereenvolgens 1 en daarna 0 worden van een signaal wordt een puls genoemd. Daarom wordt dit soort flipflop's puls-triggered genoemd. Het bijbehorend besturingssignaal zal voortaan klok of clock genoemd worden. Een ander veel voorkomende vorm van klok-besturing heet edge-triggered. Dat betekent dat zowel het opnemen van informatie als het veranderen van de uitgangswaarde van de flipflop op dezelfde flank gebeuren. Dit kan dan een opgaande flank (positive edge-triggered) of een neergaande flank(negative edge-triggered) van het kloksignaal zijn.
Omdat in flipflop schakelingen geen directe verbinding van ingang naar uitgang meer is mogen nu ongestraft terugkoppelingen worden aangebracht. Als de inverse uitgang Q' verbonden wordt met de D-ingang verkrijgt men een twee-deler.



Afbeelding 45 tweedeler

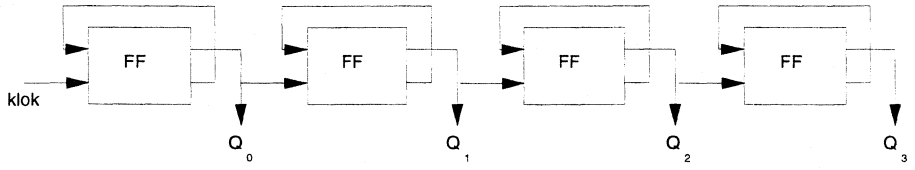
In het timingsdiagram is te zien dat na elke neergaande flank van de klokpuls de uitgang van de flipflop de tegengestelde waarde zal aannemen. De periode is tweemaal die van de klokpuls en de frequentie dus de helft van de klokpulsfrequentie.

De waarde van een flipflop na een klokpuls is een functie van de waarden van de flipflop's voor de klokpuls en de ingangen. Om dat in een formule duidelijk te maken wordt de flipflop-variabele van een tijdmarkering voorzien. In het geval van de tweedeler kan geschreven worden:

$$Q_i^{n+1} = (Q_i^n)'$$

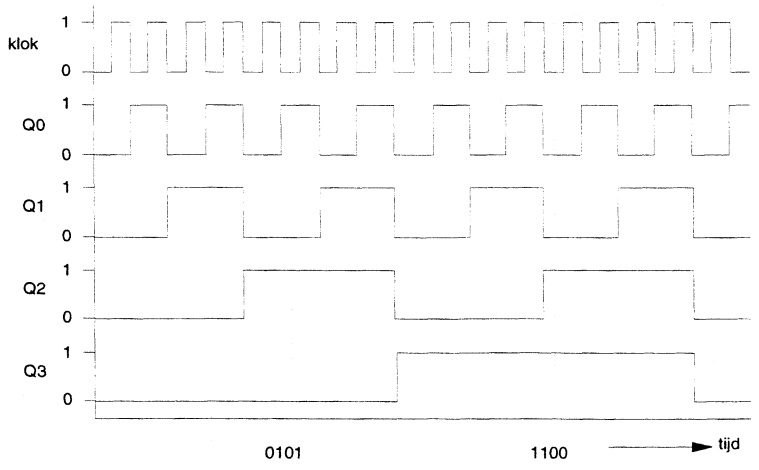
Wat gelezen moet worden als Q na de klokpuls krijgt de inverse waarde van Q voor de klokpuls.

Vier twee-delers achter elkaar geschakeld leveren het volgend schema:



Afbeelding 46 asynchrone teller

De uitgang van een flipflop wordt weer gebruikt als klok van de volgende flipflop. Omdat elke flipflop een zekere tijd nodig zal hebben om om te klappen, zal het veranderen van Q_3 pas 4 omslagtijden na de ingangsklok plaatsvinden. De totale vertragingstijd neemt lineair toe met de lengte van de teller en is sterk afhankelijk van de data. De werking van dit soort schakelingen wordt asynchroon genoemd. De omslagtijden in het timingsdiagram van de asynchrone teller zijn getekend op de plaats van de neergaande flanken van de klok, maar in werkelijkheid liggen ze iets later.



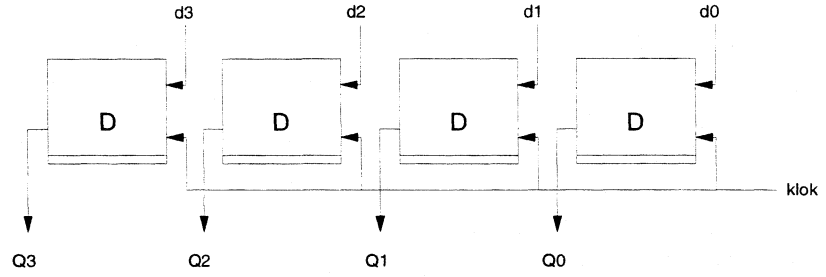
Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Afbeelding 47 Timingsdiagram asynchrone teller

In de tabel is te zien dat de uitgangen van de teller de normale binaire code doorlopen. Na elke klokpuls geeft de teller aan hoeveel klokpulsen er al geweest zijn. Om de teller goed uit te kunnen lezen moet gewacht worden tot alle flipflop's zijn omgeklapt. Om van 0111 naar 1000 te komen worden snel de tussenstanden 0110, 0100 en 0000 doorlopen. Deze z.g.n race-conditions zijn het probleem bij asynchrone schakelingen.

4.2 Registers

De beschreven flipflop wordt de D-flipflop genoemd (nu staat D voor delay). Het is de basis-bouwsteen voor de bouwblokken die men REGISTER's noemt. Registers zijn er in vele soorten en maten. Ze hebben allen de eigenschap dat een aantal bits onthouden kunnen worden, maar in veel gevallen kan de inhoud van zo'n register, altijd onder besturing van een klokpuls, bewerkt worden. Als de klokingangen van alle flipflops in een register met elkaar zijn doorverbonden en aangesloten op de klokingang van het register, dan spreekt men van een synchroon register. De inwendige flipflops zullen allen tegelijk hun nieuwe waarde aannemen. Veel registers hebben een stuuringang clear (vaak **niet** synchroon met de klok) waarmee alle flipflops de waarde 0 krijgen. In de schema's is dit niet aangegeven.
Nu volgen de meest bekende registerorganisaties.
De meest eenvoudige is aangegeven in Afbeelding 48.

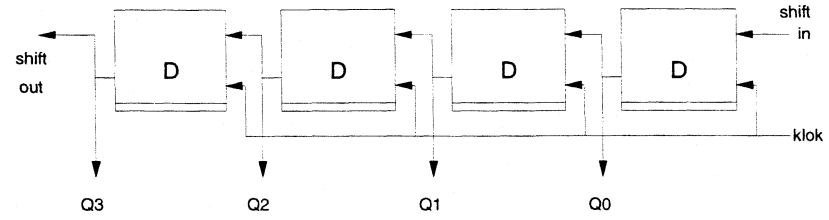


Afbeelding 48 (parallel load) register

Onder klokpulsbesturing kan een aantal bits parallel, een woord, opgeslagen worden. Vaak komt bij dit soort registers nog een stuuringang inhibit (of het tegengestelde enable) voor. Met deze stuuringang kan de inhoud van de flipflops op de oude waarde worden gehouden, ook al komt er een klokpuls. Formules voor de flipflops:

$$Q_i^{n+1} = d_i \quad \text{of} \quad Q_i^{n+1} = \text{inhibit}'.d_i + \text{inhibit}.Q_i^n$$

Het volgende voorbeeld is het schuifregister in Afbeelding 49

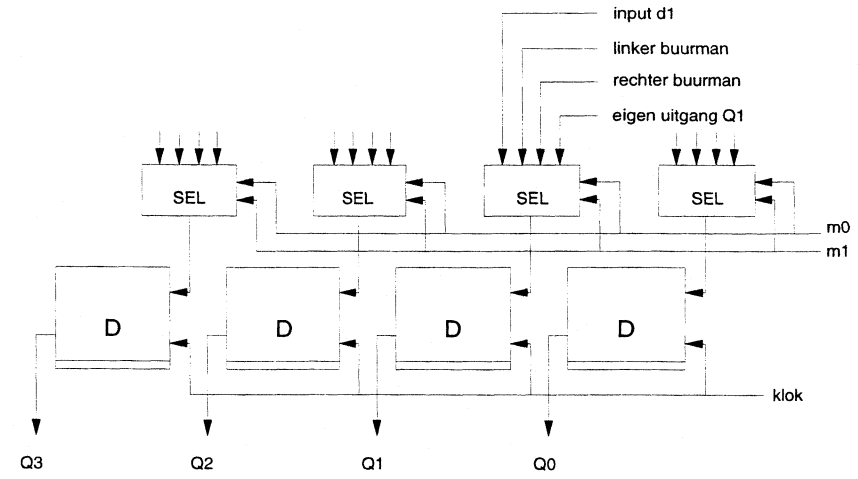


Afbeelding 49 Schuifregister

Bij het schuifregister is de ingang van elke flipflop verbonden met de uitgang van zijn buurman. Bij elke klokpuls schuiven alle bits 1 plaats op. In het getekende schema zal het meest linkse bit verloren gaan, terwijl aan de rechterzijde de bits worden aangevuld met een bit dat van buiten af wordt aangeboden. Als formule van de flipflops geldt dus:

$$Q_i^{n+1} = Q_{i-1}^n$$

In de handel zijn vele typen schuifregisters te krijgen. Bij de 4-bits en 8-bits uitvoeringen zijn alle flipflop-uitgangen verbonden met uitwendige aansluitpennen, maar bij de typen met meer bits is slechts een beperkt aantal uitgangen naar buiten uitgevoerd. Er zijn typen met 1024 bits.



Afbeelding 50 load-shift register

Als elke flip-flop ingang voorzien is van een selector, dan kan met mode-bits bepaald worden wat de functie van het register is. In Afbeelding 50 is een voorbeeld met 4 functies aangegeven.

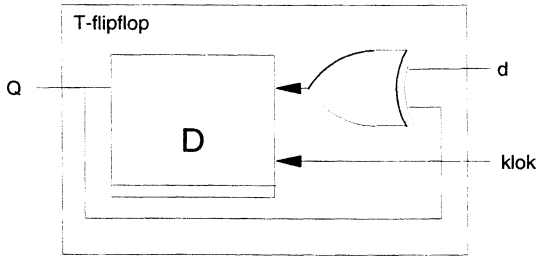
m0	m1	functie
0	0	inhibit
0	1	shift left
1	0	shift right
1	1	parallel load

Buiten deze 4 functies die onder klokbesturing worden uitgevoerd, is vaak nog een signaal clear (reset) aanwezig dat niet op de klok wacht, dus asynchroon, alle flipflops op 0 zet. In de TTL-reeks heet dit type 74194.

Een ander type registers zijn de tellers. Deze tellers doorlopen onder besturing van de klokpuls een aantal codecombinaties. Dit is meestal de zuiver binaire code, maar er zijn ook tellers die het getelde aantal klokpulsen in een andere code weergeven.

Als basisblok voor tellers wordt vaak een hulpblokje, de T-flipflop gebruikt. In Afbeelding 51 is dit getekend. Als de ingang d=0 dan behoudt de flipflop zijn oude waarde, maar is d=1 dan klapt de flipflop op elke klokpuls om van waarde.

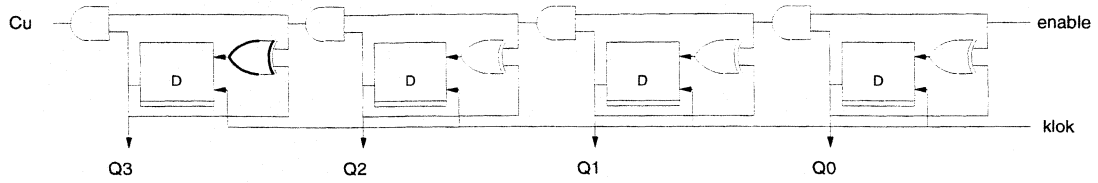
$$Q_i^{n+1} = d_i \oplus Q_i^n$$



Afbeelding 51 T-flipflop

Dit is dus een synchrone, bestuurbare tweedeler.

De gewone binaire counter of teller is in Afbeelding 52 gegeven.



Afbeelding 52 Binaire teller

Als enable=0 dan zal de keten van and-poorten voor alle ingangen van de T-flipflops een 0 opleveren. Alle flipflops behouden de oude waarde. Een stuuringang van de T-flipflop kan slechts 1 worden als enable=1 en als alle flipflops voor de lager gewaardeerde bits 1 zijn.

code tabel:

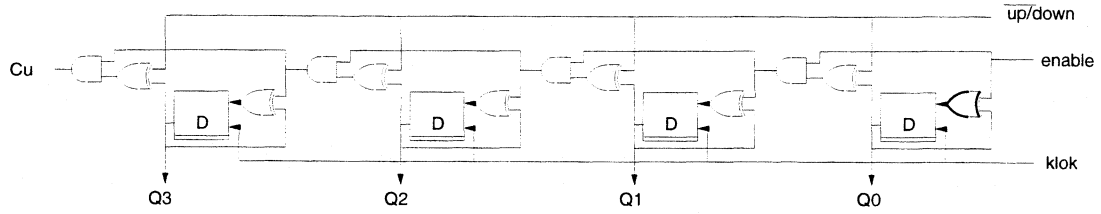
Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Dit kan korter maar minder doorzichtig worden weergegeven met de formule:

$$Q_i^{n+1} = Q_i^n \oplus enable \cdot \prod_{j=0}^{i-1} Q_j^n$$

Ook tellerblokjes hebben naast de telfunctie de clear of reset en de parallel load functie. Welke functie aktief is wordt weer bestuurd d.m.v. mode-signalen.

Een andere mogelijkheid is dat tussen 2 telfuncties gekozen kan worden.



Afbeelding 53 Binaire up-down counter

In Afbeelding 53 is een up/down teller gegeven. Hiermee kan onder besturing up'/down heen of teruggeteld worden. Met het invers aangeven van **up** in de naam van de besturingsingang, geeft men aan dat 0 => heentellen en 1 => terugtellen.
Bij heentellen klapt een flipflop om als alle secties rechts van die flipflop 1 zijn. Bij terugtellen klapt een flipflop om als alle flipflops rechts ervan de waarde 0 hebben.
Met up'/down=1 worden juist alle geinverteerde uitgangen van de flipflops aan de keten van and-poorten toegevoerd.

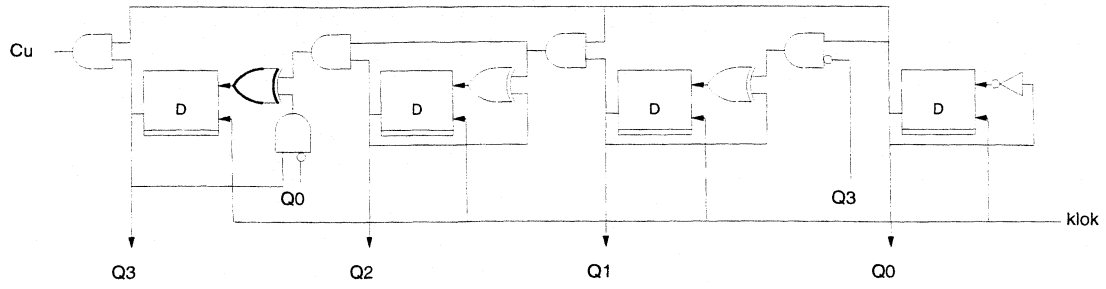
code tabel:

heen-tellen				terug-tellen			
Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

Of in een formulevorm:

$$Q_i^{n+1} = Q_i^n \oplus enable \cdot \prod_{j=0}^{i-1} (down \oplus Q_j)$$

Een teller voor de BCD-code is aangegeven in Afbeelding 54. In het schema zijn enkele ingrepen gedaan om ervoor te zorgen dat na de stand voor 9 weer de stand 0 volgt.



Afbeelding 54 BCD counter

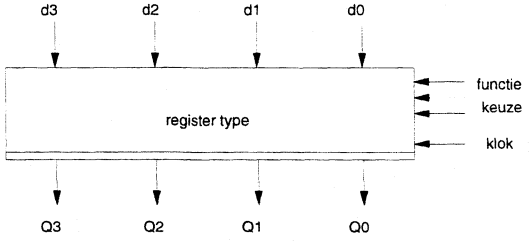
code tabel:

Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
0	0	0	0

Of in formule:

$$Q0^{n+1} = (Q0^n)'$$
$$Q1^{n+1} = Q1^n \oplus Q0^n.(Q3^n)'$$
$$Q2^{n+1} = Q2^n \oplus Q0^n.Q1^n$$
$$Q3^{n+1} = Q3^n.(Q0^n)' \oplus Q0^n.Q1^n.Q2^n$$

Ook BCD counters zijn in verschillende vormen als bouwblokjes te verkrijgen. Vaak zitten er meer decaden in één huisje.
In grote schakelingen worden de registers niet in detail getekend, maar als blok met functie of typenummer aangegeven:



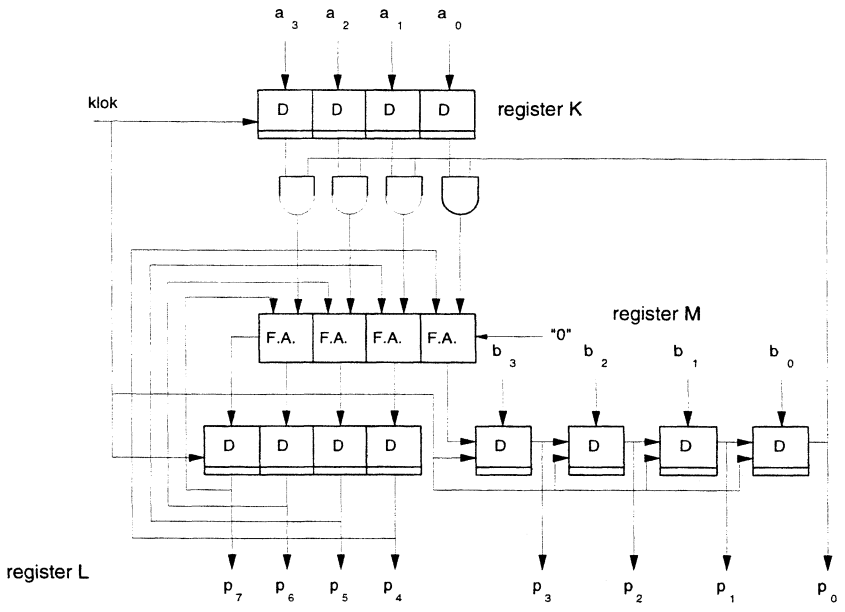
Afbeelding 55

4.3 Schakelingen met registers

Als voorbeeld van registergebruik nu de serie-parallelvermenigvuldiger. Als 2 getallen A en B van 4 bits met elkaar vermenigvuldigd worden, dan ziet dat er op papier als volgt uit:

$$
\begin{array}{cccccccc}
 & & A = & & a_3 & a_2 & a_1 & a_0 \\
 & & B = & & b_3 & b_2 & b_1 & b_0 \\
 & & & & \multicolumn{4}{c}{\text{———————————}} & x \\
 & & & & a_3.b_0 & a_2.b_0 & a_1.b_0 & a_0.b_0 \\
 & & & a_3.b_1 & a_2.b_1 & a_1.b_1 & a_0.b_1 & \\
 & & a_3.b_2 & a_2.b_2 & a_1.b_2 & a_0.b_2 & & \\
 & a_3.b_3 & a_2.b_3 & a_1.b_3 & a_0.b_3 & & & \\
 & \multicolumn{7}{c}{\text{———————————————————————————}} & + \\
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Het subproduct wordt gevormd door één bit van B te vermenigvuldigen met alle bits van A. Als b_i=0 is wordt het subproduct 0 en als b_i=1 is worden de bits van A doorgegeven. Dit kan worden uitgevoerd met and-poorten. De subproducten worden verschoven bij elkaar opgeteld, dit wordt in de schakeling uitgevoerd door het tussenresultaat te verschuiven.



Afbeelding 56 serie-parallel vermenigvuldiger

In de schakeling in Afbeelding 56 zijn drie registers K,L en M getekend. Register K is een parallel-load register, waarvan de waarde gedurende de vermenigvuldiging niet verandert. Register L is een parallel-load register waarin het resultaat van de optellingen wordt bijgehouden. Register M is een schuifregister, en heeft twee taken nl het bit voor bit aanbieden van het getal B en het bewaren van de bekende antwoordbits.
Om de schakeling te testen zal $11_{10} * 13_{10}$ of $1011_2 * 1101_2$ uitgerekend worden.

K	L	M	full-adder naar L en M
1 1 0 1	0 0 0 0	.1 0 1 1	1 1 0 1 + 0 0 0 0 = 0 1 1 0.1
1 1 0 1	0 1 1 0	1.1 0 1	1 1 0 1 + 0 1 1 0 = 1 0 0 1.1
1 1 0 1	1 0 0 1	1 1.1 0	0 0 0 0 + 1 0 0 1 = 0 1 0 0.1
1 1 0 1	0 1 0 0	1 1 1.1	1 1 0 1 + 0 1 0 0 = 1 0 0 0.1
1 1 0 1	1 0 0 0	1 1 1 1.	

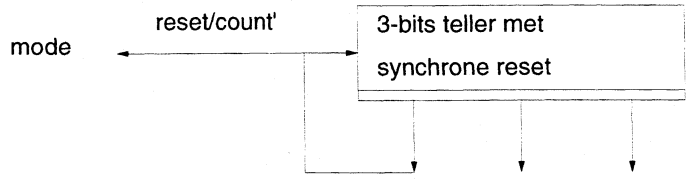
antwoord: L,M = $10001111_2 = 143_{10}$

Werkwijze:

1. Geef alle registers de beginwaarde.
2. Bepaal met de registeruitgangen en de ingangen van de schakeling (hier geen), de ingangen van de registers.
3. Vul op grond van de vorige stap de registers opnieuw om een klokpuls uit te voeren.
4. Ga naar stap 2.

Bij de bovenstaande berekening is ervan uitgegaan dat de te vermenigvuldigen getallen al in de registers aanwezig waren. Maar het totale vermenigvuldigproces omvat ook de aanvoer van de te vermenigvuldigen getallen en het uitlezen van het antwoord.
De modi waarin de registers werken is achtereenvolgens:

K	L	M	TELLER
laad	reset	laad	reset
inhibit	laad	shift	tel
inhibit	laad	shift	tel
inhibit	laad	shift	tel
inhibit	laad	shift	tel
laad	reset	laad	reset



Afbeelding 57 besturing

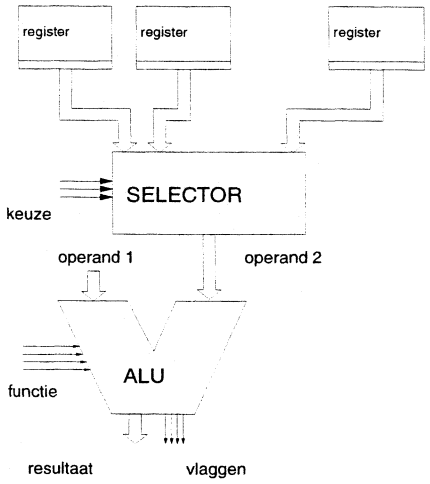
Hierbij wordt verondersteld dat het uitlezen van het antwoord en het laden van de nieuwe getallen tegelijkertijd plaatsvindt. Elk register heeft twee modi van werken. In Afbeelding 57 is een teller zodanig geschakeld dat om de 5 klokpulsen het signaal 1 klokpuls lang 1 wordt. Dit kan gebruikt worden om alle registers te sturen.
Het gedeelte van een schakeling dat de bewerking op de bitpatronen uitvoert wordt het data-pad genoemd. Hier is dat de vermenigvuldigschakeling. Het gedeelte van een schakeling dat de modi van de registers en de instellingen van de verbindingswegen regelt wordt de besturing genoemd. Hier is de besturing een simpele teller, maar bij complexe schakelingen kunnen dit hele lijsten van instelgegevens zijn die achtereenvolgens worden afgewerkt. Zo'n besturing levert elke klokpuls alle instelgegevens van registers en datapaden. Als de besturing door de hardware-ontwerper veranderd kan worden wordt het microprogrammering genoemd.

5 Verbindingssystemen

5.1 Three-state

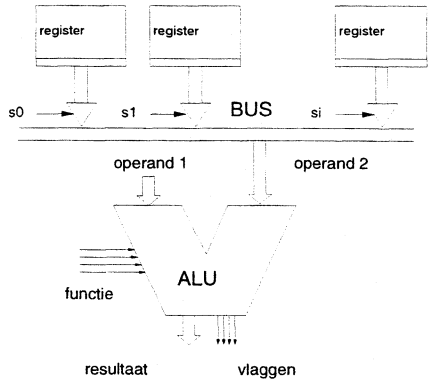
In computers zitten een aantal blokken, bijvoorbeeld registers, die selectief met de ingang van een ALU moeten kunnen worden verbonden. Wil men dat met de tot nog toe behandelde blokken doen, dan zou de ALU-ingang met een selector kunnen worden uitgerust om te kunnen kiezen welk register de operand zal leveren.

Bestaan de registers uit n bits, dan zullen er ook n selectoren nodig zijn. Elk van de getekende registers heeft n verbindingen met de selectoren. Dit is aangegeven in Afbeelding 58. Naast het complexiteitsprobleem is er ook nog het probleem van de uitbreidbaarheid. Stel dat het systeem oorspronkelijk ontworpen was voor 8 registers, en men wil in een volgende versie een negende register toevoegen, dan zullen er andere selectoren moeten worden gebruikt. Daarom is voor dit soort verbindingsproblemen het bussysteem ontwikkeld.



Afbeelding 58

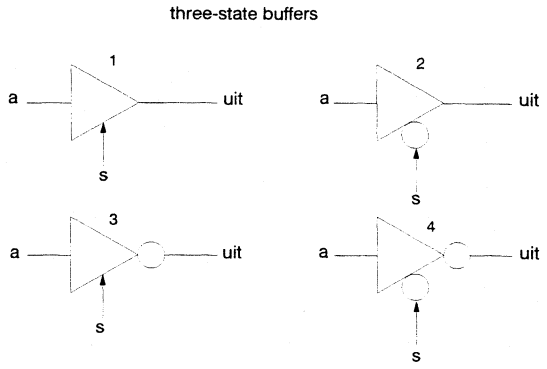
In Afbeelding 58, Afbeelding 59 is elk register voorzien van elementen die met behulp van een stuursignaal si open of dicht gezet kunnen worden. De uitgangen van deze elementen worden door het bussyteem gewoon met elkaar doorverbonden. Deze elementen worden three-state buffers genoemd. Om de uitgangen met elkaar door te kunnen verbinden kan zo'n uitgang 0,1 of hoogohmig zijn. Als men er via de stuuringangen nu voor zorgt dat slechts één buffer als uitgangswaarde een gedefinieerde waarde 0 of 1 kan hebben, en de rest van de uitgangen hoogohmig is, kan er geen kortsluiting optreden.



Afbeelding 59

Het hoogohmig zijn van de uitgang van een three-state buffer in TTL-techniek wordt bereikt door in de eindtrap, die uit twee transistoren bestaat, beide transistoren dicht te zetten. De uitgang is dan nergens mee verbonden, hij is hoogohmig en heeft geen invloed op het spanningsniveau op de buslijn. Als geen der sturende elementen geselecteerd is, wordt de spanning op de buslijn bepaald door afsluitweerstanden.
De bouwblokken die in de handel verkrijgbaar zijn hebben de three-state buffers vaak al ingebouwd. Zo zijn er registers, geheugens en selectoren met three-state uitgangen. Met behulp van een ingangssignaal output-enable kunnen de uitgangen aktief gemaakt worden.

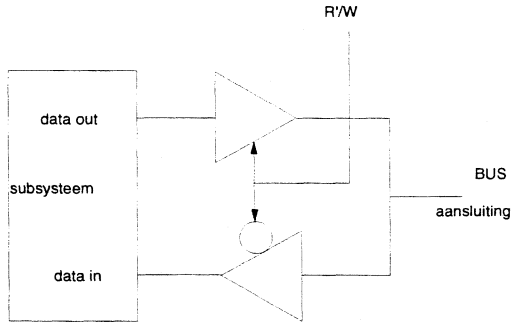
In Afbeelding 60 zijn de symbolen van de verschillend uitvoeringsvormen getekend.
Zowel de sturing als de data kunnen geinverteerd zijn.



Afbeelding 60 three-state buffers

geval 1			geval 2			geval 3			geval 4		
a	s	uit	a	s	uit	a	s	uit	a	s	uit
0	0	hoogohmig	0	0	0	0	0	hoogohmig	0	0	1
0	1	0	0	1	hoogohmig	0	1	1	0	1	hoogohmig
1	0	hoogohmig	1	0	1	1	0	hoogohmig	1	0	0
1	1	1	1	1	hoogohmig	1	1	0	1	1	hoogohmig

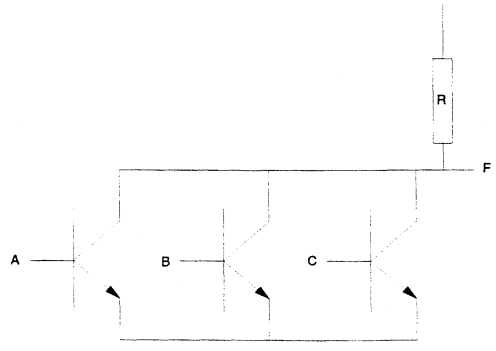
Sommige blokken zijn uitgerust met een bi-directionele busdriver zoals aangegeven in Afbeelding 61. Het R(ead)'/W(rite) signaal bepaalt of er naar de bus geschreven, of van de bus gelezen kan worden.



Afbeelding 61 bi-directionele databuffer

5.2 Open collector

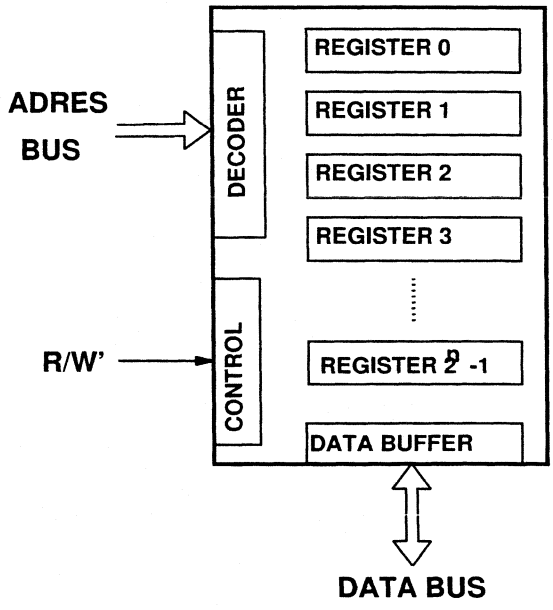
Een andere oplossing om uitgangen te verbinden is de open-collector uitgang. In de uitgangstrap is alleen een transistor naar aarde is aanwezig. De uitgang kan naar 0 geforceerd worden door de transistor open te zetten.
Als geen der aangesloten uitgangen de waarde 0 hebben, zorgt een afsluitweerstand voor het 1 worden van de lijn. Deze verbinding van uitgangen wordt wel wired-or genoemd, functioneel gezien wordt echter een AND-poort gevormd. Om één uitgang de bus te kunnen laten besturen moeten alle "niet aktieve" uitgangen van de andere units op 1 gehouden worden.



Afbeelding 62 Open-collector verbindingen

6 Geheugens

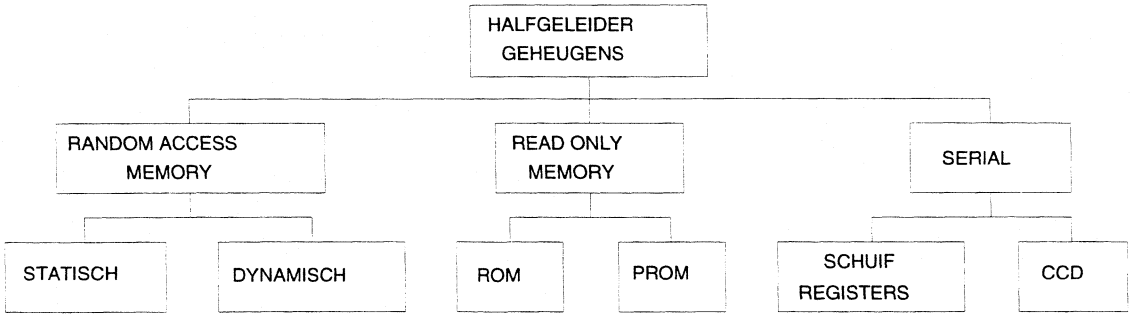
Men kan een computergeheugen voorstellen als een groot aantal registers van gelijke lengte. Om het aantal aansluitdraden beperkt te houden, is er maar 1 register tegelijk te gebruiken. Daarom zijn de registers genummerd en kunnen door een decoder of demultiplexer worden aangewezen. Het getal dat de decoder bestuurt wordt adres genoemd. Het adres bepaalt dus welk register geaktiveerd wordt. Voor de in het schema aangegeven 2^n registers zijn n adres draden nodig voor de aansturing. Het controle signaal R/W' bepaalt de richting van het data transport. R/W'= 0 betekent dat er data naar het geheugen wordt geschreven, en R/W'= 1 betekent dat er data naar de DATA BUS "geschreven" wordt. Het aantal datalijnen wordt bepaald door de registergrootte. De bij elkaar horende databits in een register worden een woord genoemd.
Om de interne bitgeheugens zo eenvoudig mogelijk te maken worden geen flipflops, maar een soort trekkers gebruikt. Daarom heeft een geheugen geen klokaansluiting.
Als het aantal benodigde adreslijnen erg groot wordt past men z.g.n. time-multiplexed adres sering toe. Dat betekent dat in een aantal opvolgende tijdsmomenten het adres, in stukken, aangeboden wordt.
Bij het aangeven van de grootte van een geheugen de factor 2^{10} = 1024 als k(ilo) weergegeven. Dus 2^{19} woorden van 8 bits wordt vaak geschreven als 512k bytes.



Afbeelding 63 computergeheugen

6.1 Overzicht halfgeleidergeheugens

Geheugens die in computers veelvuldig toegepast worden zijn in Afbeelding 64 aangegeven. Eerst is onderscheid te maken in de wijze waarop toegang gekregen kan worden tot de geheugencellen.



Afbeelding 64 Typen 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, éénmalig, of door de fabrikant, of door de gebruiker geprogrammeerd. Sequential Access Memories zijn seriële geheugens, waarbij de toegangstijd kan varieren afhankelijk van het

gevraagde adres en de toestand van het geheugen.

Bij de keuze van het geheugen zal de ontwerper een aantal eigenschappen op prijs stellen nl:

.Hoge snelheid van lezen en schrijven.
.Geen verlies van informatie.
.Eenvoudige besturing.
.Kleine afmetingen (hoge integratiedichtheid).
.Laag energieverbruik (weinig warmteontwikkeling en koeling).
.Lage prijs.

Het is nog niet zo ver dat deze eigenschappen allen in één type geheugenchip verenigd zijn. Daarom zal er altijd een compromis tussen deze eigenschappen gevonden moeten worden. Dat verklaart ook dat er zoveel typen geheugen naast elkaar blijven bestaan. Heeft men bijvoorbeeld een groot beschrijfbaar geheugen nodig, dan zal de prijs, afmeting en warmteontwikkeling een belangrijke rol spelen. Het verlies van informatie bij het afzetten van de spanning, de gematigde snelheid en de iets moeilijkere besturing (refreshen) zal men dan voor lief moeten nemen. Om de afweging te kunnen maken moeten de eigenschappen bekend zijn.

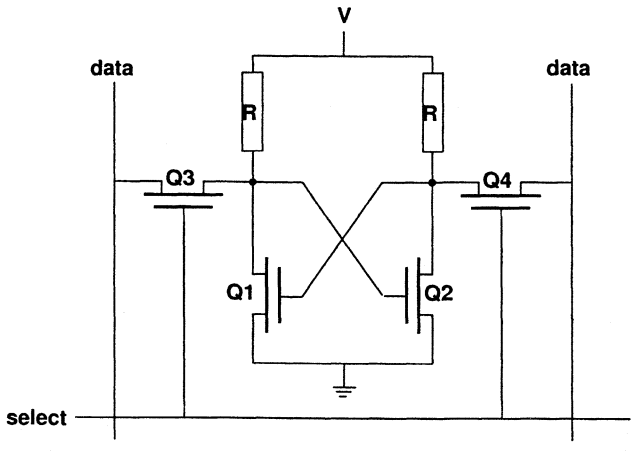
6.2 Random Access Memory (RAM)

Dit type geheugen is met hoge snelheid (10 tot 100 nsec als toegangstijd) te lezen en te beschrijven. Bij het afzetten van de spanning zal alle informatie verloren gaan.
Er worden hier 3 typen bekeken:

.Statische RAM
.Dynamische RAM
.Bipolaire RAM

6.2.1 Statische RAM

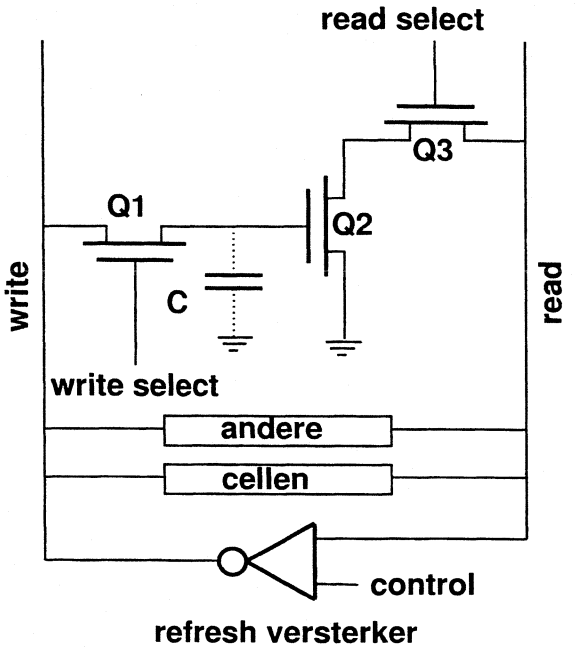
De MOS-uitvoering van een statische RAM cel is in Afbeelding 65 getekend. Het is een bi-stabiele schakeling gevormd door de transistoren Q1 en Q2. Als Q1 geleidend wordt gemaakt, dan zal de spanning op de gate van Q2 tot bijna 0 Volt dalen en Q2 sperren. Het gevolg is een hoge spanning op de gate van Q1, die dus geleidend zal blijven. De andere stabiele toestand is dat Q2 in geleiding is en Q1 gesperd. De schakeling kan dus 1 bit informatie bevatten. Om een geheugen samen te stellen zijn een groot aantal bits in een matrix gerangschikt. Om een bit te aktiveren wordt met een selectsignaal van een rij bits de transistoren Q3 en Q4 open gezet. Op de data draden van de gewenste kolom is de waarde dan uit te lezen. Om het bit te schrijven wordt via de datalijn op de geselecteerde plaats een lage spanning geforceerd. Omdat het in deze MOS-techniek eenvoudiger is om een transistor dan een weerstand te maken, worden de weerstanden uitgevoerd als een speciaal gemodificeerde transistor. De tijd voor het lezen of schrijven van een bit is in de orde van 50 nsec.



Afbeelding 65 Statische MOS RAM-cel

6.2.2 Dynamische RAM

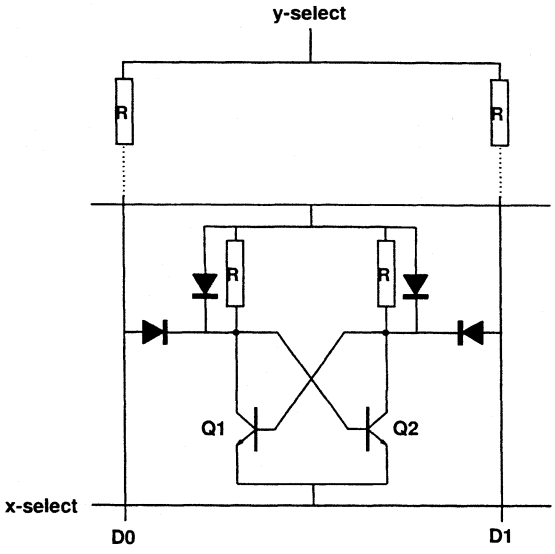
De dynamische RAM cel maakt voor het opslaan van de informatie gebruik van het al of niet aanwezig zijn van een lading op een condensator. Dit is getekend in Afbeelding 66.
De condensator zal zijn lading door lekstromen verliezen, waardoor ook de informatie verloren gaat. De opgeslagen informatie moet daarom regelmatig worden ververst, minimaal 1 keer in de 2 msec. De bitschakelingen zijn in een matrix geplaatst. Om een bit te schrijven wordt via het write select signaal, van een hele rij, de transistor Q1 open gezet. Met het, kolomsgewijs georganiseerde, write signaal kan de condensator al of niet van lading worden voorzien. Om te lezen wordt met het read select signaal van een hele rij de transistor Q3 open gezet. Op de gewenste kolom kan dit via de draad read worden uitgelezen. Bij het uitlezen wordt ook de informatie via de refresh amplifier op de betreffende plaats teruggeschreven. Omdat natuurlijk niet gegarandeerd kan worden dat elk bit binnen 2 msec een keer gelezen zal worden, moeten extra refreshcycles worden ingelast. Het verversen van de bits wordt per rij gedaan. In elke kolom wordt op de geselecteerde rij via de refreshamplifier de gelezen waarde teruggeschreven. Dat betekent dat voor de refresh organisatie dat cyclisch alle rij-adressen binnen 2 msec. eenmaal geaktiveerd moeten worden. Dit refreshen moet tussen het lezen en schrijven door gebeuren.
De toegangstijd van deze geheugens ligt in de orde van 70 nsec.



Afbeelding 66 Dynamische MOS RAM cel

6.2.3 Bipolaire RAM

De bipolaire RAM is opgebouwd uit bistabiele schakelingen met transistoren. Zie Afbeelding 67. Als Q1 geleidend is zal de basis van Q2 een lage spanning hebben, en gesperd zijn. Daardoor wordt de collectorspanning van Q2 hoog en kan de transistor Q1 open gestuurd worden. Als met een hoge spanning op de basis van Q2, Q2 in geleiding wordt gebracht, zal de schakeling omklappen.
De geheugencellen zijn weer in een matrix geplaatst. Om een cel te selecteren wordt het y-select signaal van de gekozen kolom op een hogere spanning gebracht, en aan het xselect signaal van de gekozen rij op een lagere spanning aangeboden. De geleidende transistor in de gekozen cel zal ervoor zorgen dat één van de, bij de kolom horende, draden D0 of D1 laag zal worden. Het verschil van D0 en D1 zal dus de waarde van het geselecteerde bit aangeven. Moet er niet gelezen, maar geschreven worden, dan kan naar keuze, op D0 of D1 een hogere spanning (een 1) worden gezet om de geselecteerde transistor in geleiding te brengen.

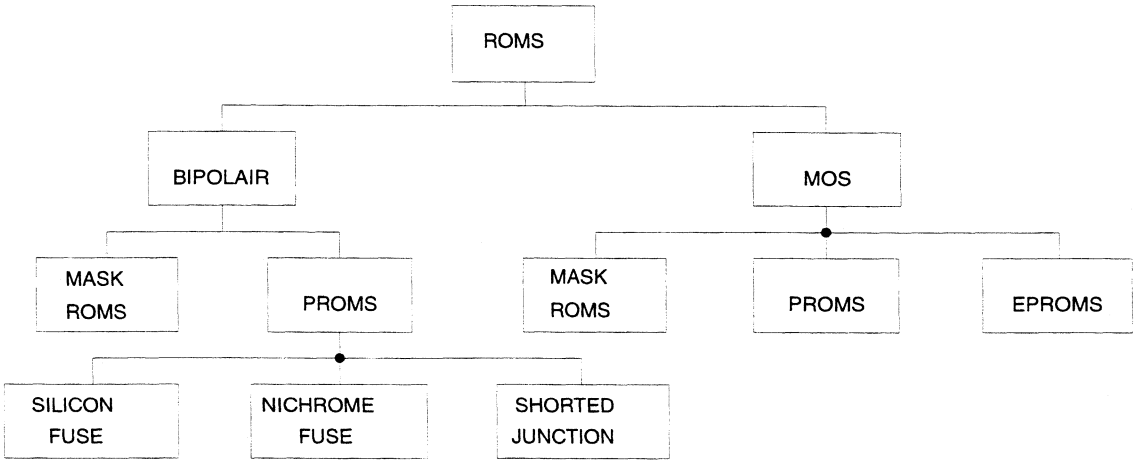


Afbeelding 67 Bipolaire RAM cel

De toegangstijd van dit geheugen is 15 a 20 nsec. Het energieverbruik en dus de warmteontwikkeling is hoog. De integratiedichtheid is kleiner dan bij de andere ramtypen. De hoeveelheid geheugen per I.C. is kleiner en de kostprijs veel hoger.

6.3 Read Only Memory (ROM)

Er zijn toepassingen waarbij het verliezen van informatie door het uitschakelen van de spanning niet toelaatbaar is. Dan maakt men gebruik van niet vluchtige geheugens (non volatile memory) of Read Only Memories, geheugens waaruit tijdens normaal bedrijf alleen gelezen kan worden. Het "beschrijven" van deze geheugens heeft al eerder plaats gevonden. Apparaten met een aantal vaste instellingen gebruiken vaak ROM's voor het opslaan van die constanten. Bij computers zit bijvoorbeeld het opstartprogramma (bootstrapprogramma) in ROM. En de programma's voor spelcomputers zijn ook vaak in ROM's ondergebracht.
Dit kan dus alleen voor gegevens die tijdens gebruik niet veranderen. Als de gegevens wel kunnen veranderen neemt men zijn toevlucht tot zuinige, beschrijfbare geheugens die d.m.v. een oplaadbare batterij blijvend van spanning worden voorzien. (kanaalinstelling T.V.,systeemconfiguratie PC enz.)
De ROM's kunnen worden ingedeeld naar technologie en manier waarop ze beschreven worden. Een overzicht is te zien in Afbeelding 68.



Afbeelding 68 Typen ROMS

Of men een Bipolair of een MOS type kiest hangt van grootte en snelheid van de toepassing af. Welke fabricageproces voor de ROM gekozen wordt hangt vooral van het aantal identieke ROM I.C.'s men nodig heeft.

6.3.1 Mask ROM's

De informatie van een mask ROM wordt aangebracht tijdens het fabricageproces. De klant gaat met de gewenste ROM-inhoud naar de fabrikant, die de informatie omzet in een masker waarmee de chip gefabriceerd wordt. Deze masker-kosten zijn hoog, maar éénmalig. Daarom is deze keuze alleen lonend bij het maken van een groot aantal (100.000) identieke produkten.
De Bipolaire hebben een kleine capaciteit maar zijn snel, de MOS-uitvoeringen paren een zeer grote capaciteit aan een klein volume en laag energieverbruik.
Een mask-ROM bestaat in feite uit een aantal kolomdraden en een aantal rijdraden waarbij op elk kruispunt wel of geen kontakt gemaakt wordt. Via de adressering wordt een kolom en een rij gekozen en er wordt op de betreffende plaats getest op een verbinding.

6.3.2 PROM's

Naast de genoemde ROM's zijn er nog de P(rogrammable)ROM's. Deze PROM's kunnen door de gebruiker zelf worden voorzien van informatie. Er is hiervoor wel speciale apparatuur voor nodig. Om de PROM's te programmeren zijn twee methoden beschikbaar. De eerste is de fusible link techniek. Hierbij zijn op alle plaatsen verbindingen aanwezig die tijdens het programmeren selectief worden "opgeblazen". Dit wegbranden van de verbindingen gebeurt met een stroomstoot. Een andere methode is het aanbrengen van een verbinding. Hierbij is op elke plaats een diode in sperrichting (geen verbinding) aangebracht. Door een grote stroom door de diode te voeren gaat de diodewerking over in een kortsluiting, dus een verbinding. Beide processen zijn onomkeerbaar, veranderingen kunnen niet meer worden aangebracht.

6.3.3 EPROM's

Een ander type PROM is de EPROM wat de afkorting is van Erasable PROM.
Dit type ROM is te programmeren, maar ook weer te wissen om daarna weer van gewijzigde informatie te voorzien. De verbindingen in de matrix bestaan nu uit een MOS-transistor waarvan de gate zwevend is. Met behulp van een spanning (12,5 Volt- 25 Volt) die hoger is dan bedrijfsspanning van het I.C. (5 Volt), kan een lading op de gate van de transistor worden gebracht, waardoor op die plaats weer een verbinding ontstaat. De tijdsduur en grootte van deze programmeerspanning is kritisch en afhankelijk van het typenummer. De tijdsduur van de lading wordt door de fabrikant gegarandeerd voor 100 jaar. Om het geprogrammeerde I.C. te wissen moet het gedurende ongeveer 20 minuten worden bestraald met U.V.licht van een bepaalde golflengte. Het I.C. is daarom uitgerust met een kwartsvenster om het U.V.licht door te laten. Als apparatuur in de zon staat moet dit venster worden afgeplakt om ongewenst wissen te voorkomen.

Er is ook een type dat electrisch gewist kan worden, de EEPROM.
Dit staat voor Electrical Erasable Programmable Read Only Memory. Het wissen gebeurt met een afwijkende spanning. Het schrijven kan in de applicatie gebeuren. Het schrijven van een bit duurt langer dan het lezen van een bit. De informatie blijft behouden bij het uitschakelen van de spanning.

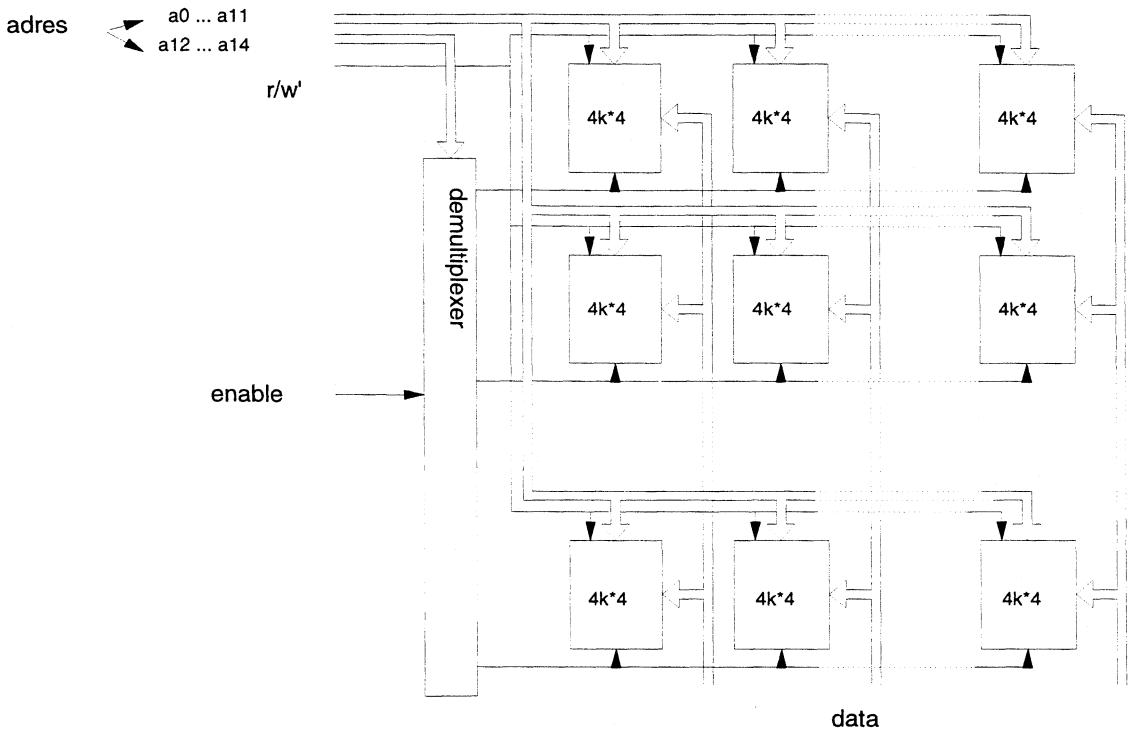
6.3.4 Seriële geheugens

Een laatste type geheugen is het seriegeheugen, gebaseerd op het schuifregisterprincipe. De informatie wordt sequentieel ingeklokt. Er zijn statische en dynamische schuifregisters. Bij de dynamische registers wordt de informatie weergegeven door het al of niet aanwezig zijn van een hoeveelheid lading. De lading moet regelmatig opgefrist worden. Bij dit soort geheugens wordt dit bereikt door de ingang van het register met de uitgang te verbinden en de informatie voortdurend rond te pompen. Door de klokpulsen te tellen weet men wat het adres is van het bit aan de uitgang van het register. Moet een bit op een bepaald adres worden gelezen of geschreven, dan moet gewacht worden tot de teller het juiste adres aangeeft. De toegangstijd is dus lang en variabel.
Bij het statische seriegeheugen wordt het bit opgeslagen in een bistabiele schakeling, en is geen refresh nodig. De organisatie is gelijk aan die van het dynamische seriële geheugen.
Een bekend type dynamisch serieel geheugen is het "Charge Coupled Device" of CCD geheugen. De bits worden dus weergegeven door het al of niet aanwezig zijn van lading. Men kan het device opgebouwd denken uit één grote MOS transistor met vele gates. Door de gates periodiek en cyclisch aan te sturen kan de lading door het substraat worden voortbewogen.

Voor echte random access toepassingen zijn de seriële geheugens minder geschikt, maar als de toepassing de informatie altijd in dezelfde volgorde gebruikt, dan geldt het bezwaar van de lange toegangstijd niet meer.
Bij de CCD-camera wordt de lading aangebracht d.m.v. licht, en de uitlezing geschiedt in een vaste volgorde. Als nu een lading niet 1 maar een aantal bits representeert, dan kan een heel videobeeld worden opgeslagen.

6.4 Geheugenmodule

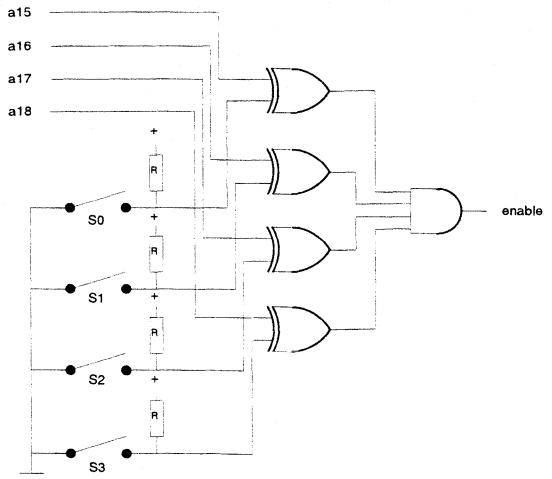
Een computersysteem heeft een maximum wat betreft de hoeveelheid geheugen dat kan worden aangesloten. In de meeste gevallen is slechts een gedeelte van deze mogelijke hoeveelheid, de "geheugenruimte", gebruikt. Maar zelfs de toegepaste hoeveelheden geheugen passen niet in één I.C. Daarom worden met geheugen I.C.'s modulen met een grotere capaciteit samengesteld. Met geheugenmodulen als eenheden kan vervolgens naar keuze de geheugenruimte worden ingevuld.
Hier wordt uitgegaan van het voorbeeld dat de maximaal mogelijke hoeveelheid aansluitbaar geheugen van 512k woorden van 16 bits of 2^{19} woorden van 16 bits. Als grootte van de geheugenmodulen wordt 32k woorden van 16 bits gekozen. De beschikbare I.C.'s hebben een capaciteit van 4k nibbles (= woorden van 4 bits).
In Afbeelding 69 is aan gegeven hoe de module kan worden opgebouwd.



Afbeelding 69 Organisatie geheugenmodule

Om het vereiste aantal databits van 16 te krijgen worden 4 I.C.s met 4 databits, in een rij, parallele geschakeld. De besturingssignalen r/w' enable en adres is voor elk I.C. ineen rij gelijk, omdat per woord van 16 bits wordt uitgelezen. Om de modulegrootte van 32K (= 5 + 10 = 15 adreslijnen) te krijgen worden 8 I.C.'s van 4k (= 2 + 10 = 12 adreslijnen) per kolom aangesloten. Met een demultiplexer of decoder wordt het enablesignaal op één van de 8 rijen doorgegeven. Er zijn 3 adresbits op de demultiplexer nodig om de keuze van 1 uit 8 te kunnen maken. Het r/w' signaal is op alle I.C.s aangesloten omdat dit signaal voor de gehele module geldt. Om de module aan te sluiten worden de 16 datalijnen aangesloten aan de 16 bits databus van de computer en de 15 adreslijnen aan de 15 laagstwaardige adreslijnen van het computersysteem.

De 4 hoogstwaardige adreslijnen, a15 tot a18, bepalen welke geheugenmodule aktief is. Daarom is een geheugenmodule uitgerust met een adresselectie die de plaats van de module in de geheugenruimte bepaalt. Omdat modulen later toegevoegd kunnen worden moet deze plaats instelbaar zijn. In Afbeelding 70 is aangegeven hoe met 4 schakelaars de plaats kan worden ingesteld. Een gesloten schakelaar zet een 0 op de exclusive-or poort en met een geopende schakelaar wordt de ingang van de poort m.b.v. een weerstand naar een 1 gebracht. Als de standen van de schakelaars overeenkomen met de signalen op de adreslijnen, wordt het enablesignaal 1.

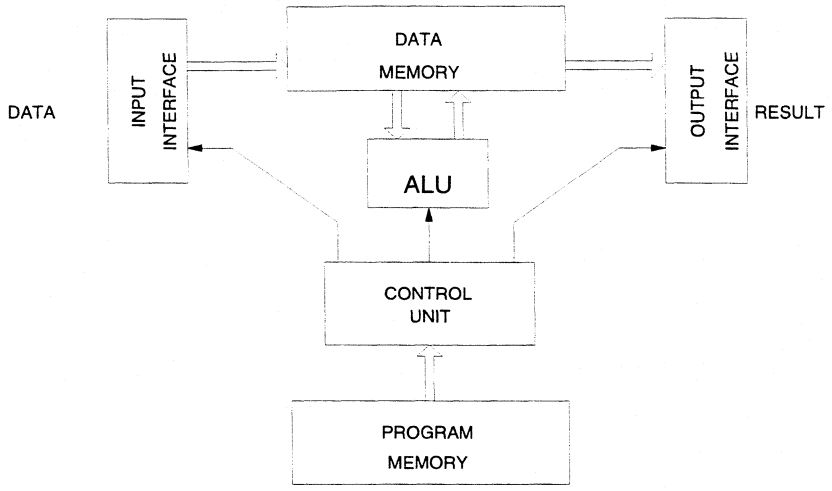


Afbeelding 70 adresselectie

7 De Computer

7.1 Inleiding

Computers zijn er in vele soorten. De grootte varieert sterk zowel in afmetingen als in verwerkingscapaciteit en prijs. Maar het systeem van informatieverwerking komt altijd op hetzelfde neer. Er zijn een aantal functionele blokken die in elk systeem zijn terug te vinden.



Afbeelding 71 computersysteem

In Afbeelding 71 zijn deze blokken aangegeven.

De input interface transformeert de data uit de buitenwereld in de in de computer gebruikte data. Dit gebeurt vaak in 2 fasen. In de eerste fase wordt de data, bijvoorbeeld druk, vochtigheid, temperatuur, het indrukken van een toets enz. via sensoren of opnemers omgezet in een electrisch signaal en in de tweede fase wordt dit electrisch signaal gecodeerd in nullen en enen. Binnen de computer worden alleen getallen, gecodeerd in het tweetallig stelsel, gebruikt met een maximale grootte afhankelijk van woordbreedte van het systeem. De data uit de buitenwereld moet zodanig worden omgezet dat het te schrijven is in rijtjes bits (getallen) van een bepaalde lengte.

De zo verkregen getallen worden opgeslagen in een geheugen, het data memory of data geheugen. Dit data memory kan, afhankelijk van het systeem, uit verschillende componenten zijn opgebouwd.
Mogelijke componenten zijn interne flipflop-registers van de computer, exterm halfgeleidergeheugen of extern achtergrondgeheugen (via magnetische of optische registratie).

Dan is er een rekeneenheid, bijvoorbeeld een ALU, die de bewerkingen op de getallen uitvoert en de verkregen tussenresultaten wegzet in het data memory en de uiteindelijke antwoorden doorgeeft naar de outputinterface.

Deze outputinterface transformeert de getallen tot de gewenste signalen. Ook hier zijn vaak 2 fasen te onderscheiden. Eerste fase is het vertalen van de getallen in spanningen of stromen die na versterking in de tweede fase door aktuatoren omgezet worden in de gewenste signalen of acties.

De control unit zet globale opdrachten uit het program memory om in besturingssignalen voor ALU, inputinterface, outputinterface en verbindingen. Deze unit levert elke klokpuls van het systeem de instelgegevens en controlesignalen van de dataverwerkende hardware.

Het program memory of programma geheugen bevat een lijst van opdrachten of instrukties, het programma. Ook deze instrukties hebben de vorm van getallen met een maximale grootte die wordt bepaald door de woordlengte van het program memory. Het vullen van het programmageheugen met instrukties, het loaden van het programma, kan op verschillende tijdstippen plaatsvinden. Als voor het program memory een ROM gebruikt wordt, en de inhoud dus tijdens de fabricage al wordt vastgelegd, dan hebben we te maken met een computer die slechts voor één doel is te gebruiken. Dit wordt een special purpose computer genoemd. Bestaat het program memory uit RAM geheugen, dan kan een programma eenvoudig worden vervangen door een ander. Dit is het geval bij de general purpose computer.
Het onveranderlijke deel van de computer dat uit componenten met vaste verbindingen bestaat, wordt de hardware genoemd. De programmatuur, die variabel is, wordt de software genoemd.
Het gedeelte dat de control unit, de ALU en de interne registers omvat wordt de Central Processing Unit of kortweg CPU genoemd. Deze CPU voor kleine computers of processoren, kan op één I(ntegrated)C(ircuit) worden ondergebracht. Het levert de microprocessorchip op.

Omdat zowel datageheugen als programmageheugen met getallen gevuld worden, kunnen voor programma en data hetzelfde geheugen, het main memory gebruikt worden.
John von Neumann was de eerste die in 1946 dit concept van in geheugen opgeslagen program ma's, het "stored program" concept, lanceerde.

De architectuur van een computer zegt iets over de opbouw van een computer. Alleen de hardware-eigenschappen die de gebruiker moet kennen om programma's te kunnen maken, behoren tot de architectuur van het systeem. Belangrijke architectuureigenschappen van computers zijn

. Het aantal registers in de CPU

. De woordlengte van het geheugen en de ALU

. De verzameling van alle instructies die uitgevoerd kunnen worden, de instructieset.

Een aantal architecturen zijn:

Meervoudige register architectuur:

De CPU bevat een aantal registers waarin tussenresultaten kunnen worden gezet. Het voordeel van het gebruik van een register boven dat van het geheugen is vooral snelheid. Interne registers hebben een kortere toegangstijd en kunnen tegelijkertijd (= parallel) werken.
Bij sommige machines zijn de registers volkomen identiek, terwijl bij anderen elk register specifieke eigenschappen heeft.

Enkelvoudige register architectuur:

De CPU bevat één dataregister, vaak ACCU genoemd. De instructies zijn eenvoudiger dan bij een meervoudige registerarchitectuur, omdat er in een instructie geen register behoeft te worden aangewezen.

Registerloze architectuur:

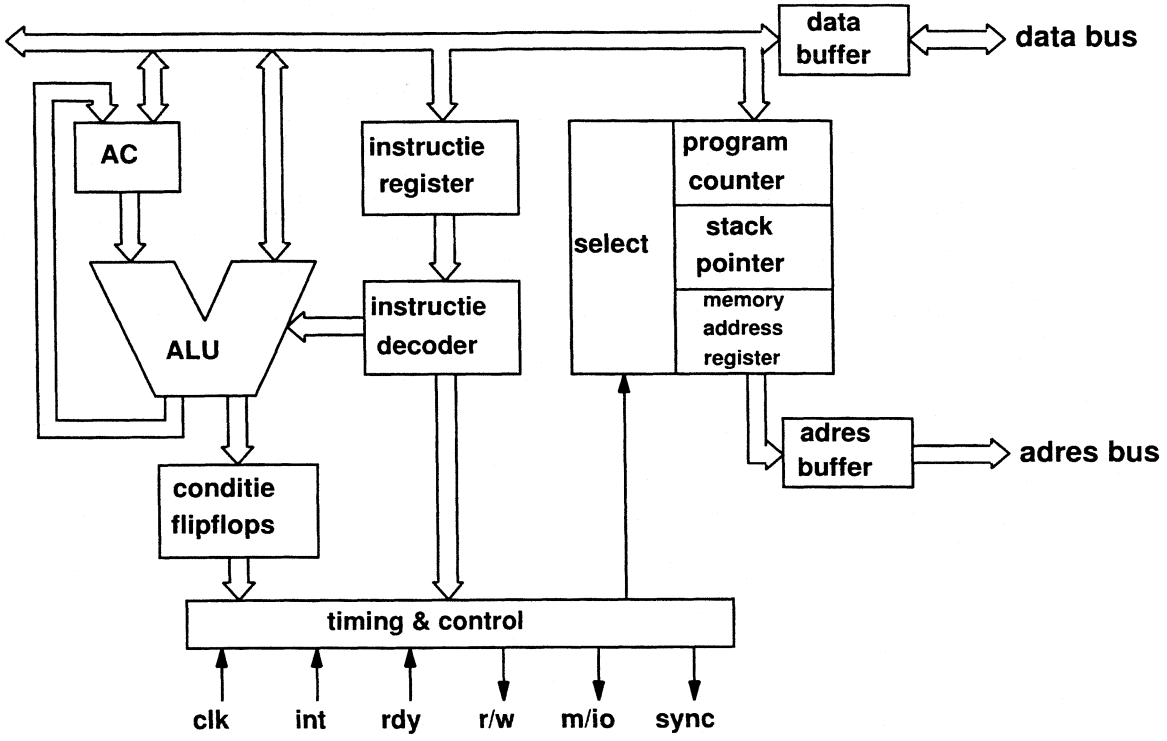
De bewerkingen moeten via het geheugen worden afgehandeld.
Een bijzondere vorm van een registerloze architectuur is de Stack architectuur. In deze archtectuur wordt een speciaal register (geen dataregister), de stackpointer, gebruikt om de plaats in het geheugen bij te houden waar de te bewerken getallen staan. De instrukties zelf passen de waarde van de stackpointer aan. Deze architectuur sluit mooi aan bij de structuur van een aantal programmeertalen.

De goedkope microprocessoren 8085 van Intel, 6800 van Motorola, Z80 van Zilog en ook de

voorbeeldcomputer, de microcontroller 80C51 van Intel zijn voorbeelden van een meervoudige registerarchitectuur met een woordlengte van 1 byte (= 8 bits) een adresruimte van 64K bytes. De instructiesets van al deze computers zijn verschillend qua codering maar komen wat betreft het soort en aantal bewerkingen overeen. De nieuwere typen microprocessoren 80x86 van Intel. 680xx van Motorola en de Z8000(0) van Zilog hebben een 16 of 32 bits architectuur, een groter adresbereik en zijn een factor duurder.

7.2 Opbouw van de C.P.U.

Als voorbeeld zal nu een enkelvoudige registerarchitectuur bekeken worden.



Afbeelding 72 computer met ACCU

In Afbeelding 72 zijn globaal de onderdelen van een C.P.U. met één register, de ACCU, weergegeven. Centraal staat de ALU die de databewerkingen voor zijn rekekening neemt. Zoals in het hoofdstuk combinatorische schakelingen al behandeld is, kan de ALU onder andere optellen, aftrekken, negeren, en een aantal logische bewerkingen uitvoeren op getallen met een lengte gelijk aan de woordlengte van de computer. Soms is er ook nog de mogelijkheid ingebouwd om alle bits 1 plaats naar links, of alle bits 1 plaats naar rechts te schuiven.

Als resultaat wordt niet alleen een antwoord, maar ook een aantal conditievlaggen bepaald. Deze conditievlaggen, die elk één bit groot zijn, worden opgeslagen in de conditieflipflops, die samen het conditiecode register genoemd worden. Een andere naam voor dit register is status register, (Program Status Word) met de statusvlaggen. De meest bekende vlaggen zijn CARRY, OVERFLOW, ZERO of EQUAL, MINUS of LESS THAN. Deze vlaggen worden bij rekenkundige en logische bewerkingen tegelijkertijd met het antwoord geleverd. Daarnaast komen er vergelijk en testinstructies voor die uitsluitend als antwoord een statusvlag opleveren. De statusvlaggen worden o.a gebruikt in de nog te behandelen conditionele sprong en subroutine instructies.

Het enige echte dataregister, de ACCU, heeft een directe verbinding met de ALU. Het resultaat van een bewerking komt altijd in de ACCU terecht.

Bewerkingen van het type $ACCU^{n+1}$ <= ($ACCU^{n}$) operatie (2^{e} operand) zijn dus mogelijk. Als voor de operatie de optelling gebruikt wordt kan in de ACCU dus de som van een aantal operanden worden gevormd, anders gezegd de operanden worden geaccumuleerd. Hiermee is de naam van het register verklaard. De ACCU verbinding met de interne bus verzorgt het ophalen van operanden en het wegschrijven van antwoorden.

Het uitvoeren van een programma betekent dat de instrukties van dat programma, één voor één, uit het geheugen worden gehaald om vervolgens te worden afgehandeld. Het register dat bijhoudt welke instructie aan de beurt is om opgehaald te worden is de Program Counter of kortweg PC. De PC bevat de plaatsaanduiding in het (programma)geheugen, het adres, van de eerstvolgende instructie.

Een ander register dat een plaats in het geheugen bijhoudt is de stackpointer of stapelwijzer. In een opbergsysteem dat als stack of stapel georganiseerd is kunnen items slechts bovenop de top van de stapel gelegd worden of van de top van de stapel worden afgehaald. Dat betekent dat wat er het eerste opkomt er als laatste kan worden afgehaald. Daarom wordt een stack-organisatie ook wel LIFO (= last in first out) genoemd. Zo'n organisatie kan er ook in een computergeheugen op na gehouden worden. Geheugenwoorden worden van stack gehaald of op stack gezet. De gebruiker hoeft het adres niet op te geven, omdat dit door de stackpointer wordt bijgehouden. Buiten het op stapel zetten en terughalen van data door de gebruiker wordt de stack ook door de computer zelf gebruikt bij het uitvoeren van de nog te behandelen instructie subroutinesprong of call.
De stackpointer bevat dus een adres voor het (data)geheugen.

Het derde getekende adresregister is het memory address register. In dit register wordt het adres gevormd dat nodig is voor de adressering van de operanden. Het register bevat dus een adres in het (data)geheugen. Bij de adresseringsmethoden zal duidelijk worden dat voor het tot stand komen van het adres van een operand soms nog wat rekenwerk moet worden verricht. Het resultaat van dit rekenwerk is een adres dat wordt opgeslagen in het memory address register.

De C.P.U. is met 3 bussen, de databus, de adresbus en de controlbus, met het combinatie van data- en programmemory, het main memory, verbonden. Het geheugen zelf heeft geen klokaansluiting en de timing moet dus elders geregeld worden. De data buffer verbindt de databus van het geheugen met de interne bus van de C.P.U. Soms is de databuffer voorzien van een register maar dit is niet noodzakelijk. Wel vindt er versterking in de vorm van bi-directional tristatebuffers plaats.

De adresbuffer verbind één van de adresrgisters door met de adresbus van het geheugen. Ook hier is meestal geen register in opgenomen. Versterking via tristate buffers in één richting zijn voldoende.

In het instructieregister wordt de, uit het geheugen gehaalde, instructie bewaard gedurende de tijd die nodig is voor de decodering.

De decoder ontrafelt de instructie. Bij bewerkingsopdrachten zal de functie van e ALU worden ingesteld. Uit de instructie decoder komen de gegevens voor de timing & control.

De timing & control die hier als blok getekend is is in werkelijkheid verdeeld over de hele schakeling. De timing en modecontrol van alle registers en het instellen van de datapaden wordt van hieruit geregeld. Daarnaast zijn er nog signalen die de timing met de buitenwereld verzorgen. Zo kan via het signaal clk een uitwendige klok worden aangesloten. In veel gevallen is dit niet nodig als de timing & control een eigen klokpulsgenerator heeft. Met het signaal int kan van buiten af een interrupt gegeven worden. Als de computer de interrupt heeft aangezet zal bij het optreden van een interrupt de normale programma-afhandeling worden stopgezet, en zal de zogenaamde interruptafhandeling gestart worden. De programmatuur hiervoor bevindt zich op een bepaalde, van te voren afgesproken, plaats in het geheugen. Met het ready-signaal rdy kan het geheugen aangeven dat het gevraagde datawoord klaar staat op de databus. Als het geheugen snel genoeg is kan dit signaal altijd op 1 gehouden worden. Gebruikt men echter langzaam geheugen dan kan de computer opgehouden worden tot het moment dat het geheugen klaar is. Men spreekt dan van waitcycles.

Het r/w signaal levert het gelijknamige signaal voor het geheugen.
Het signaal m/io is een soort extra adresinformatie. Het geeft aan of het adres op de adresbus bedoeld is voor het geheugen of voor de keuze van een randapparaat.
Het sync signaal loopt gelijk met de inwendige cycli van de machine. Het kan gebruikt worden om externe apparatuur in de pas te laten lopen met de computer.

7.3 Opbouw van de instructie

De term "instructie" is al een aantal malen genoemd. Het is een opdracht gecodeerd als getal in het tweetallig stelsel. De instructie heeft een tweetal ingrediënten nl "wat moet ik doen" , de functie en "waarmee moet ik het doen", de operanden.

FUNCTIE	ADRES OPERAND 1	ADRES OPERAND 2	ADRES RESULTAAT

INSTRUCTIE

Afbeelding 73 laat een 3-adres instructie zien. Voor elk deel van de instructie zijn een aantal bits gereserveerd. Het aantal bits dat voor de functie is gereserveerd bepaalt de grootte van de instructieset. Als er k bits voor de funktie gereserveerd zijn (en alle functiecodes zijn even lang) zijn er maximaal 2^k verschillende instructies mogelijk. De grootte van het direct te besturen geheugen wordt bepaald door het aantal bits in de adresvelden. Een voorbeeld van zo'n instructie kan zijn:

| telop | inhoud 300 | inhoud 400 | plaats 500 |

Om ruimte te sparen kan een 2-adres instruktie gemaakt worden.

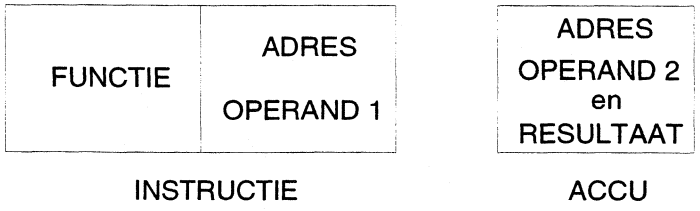
FUNCTIE	ADRES OPERAND 1	ADRES OPERAND 2 en RESULTAAT

INSTRUCTIE

Hierbij wordt voor de operand 2 dezelfde plaats gebruikt als voor het antwoord. Bij het uitvoeren van deze instructie gaat dus de waarde van operand 2 verloren. Dit is in veel gevallen toelaatbaar.
Zo niet dan zijn nu twee instructies nodig.

| copieer | inhoud 400 | plaats 500 |
| telop | inhoud 300 | plaats 500 |

Een verdere vereenvoudiging is mogelijk als men voor één van de operanden een register in de machine neemt. Heeft de machine maar 1 register, de ACCU, dan is voor operand 2 geen adresveld nodig.

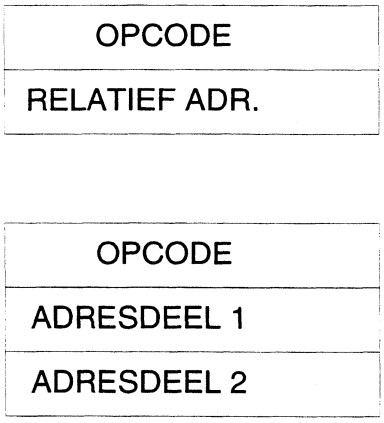


Om de som van adres_300 en adres_400 in adres_500 te krijgen zijn nu 3 instructies nodig:

haal	inhoud 300
telop	inhoud 400
breng	plaats 500

Om twee redenen wordt er naar korte instructies gestreefd.
1. Kleine instructies vragen weinig geheugenwoorden. Dat betekent tijdwinst.
   Dus meer instructies per seconde.
2. Korte instructies vragen minder geheugen. Dat betekent geheugenwinst. Grotere programma's voor hetzelfde geld.

De kleine microprocessoren gebruiken 1-adres instructies en hebben een databus van 8 bits.
Voor de functie of "operatiecode" of kortweg "opcode" worden meestal 8 bits gereserveerd.
Dus ondanks het feit dat de instructielengte tot een 1-adres formaat is ingekort, past een instructie nog niet in één geheugenwoord. Daarom kan een instructie bestaan uit 1,2,3 en soms zelfs 4 bytes (= 8 bits woord). Dit zijn de meerwoords instructies.
Voorbeelden van een 2-woords en een 3-woords instructie:

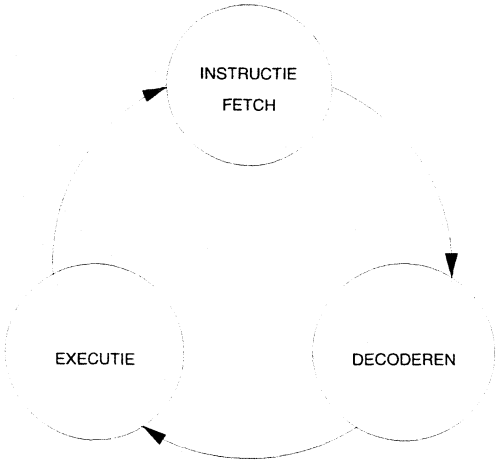


Afbeelding 76 instructieformaat

In het eerste voorbeeld worden 8 bits voor de adressering gebruikt, maar dit is relatief t.o.v. een 16 bits register, meestal de PC. Door dit 8 bits, two's complement gecodeerd, getal op te tellen bij de waarde van de 16-bits PC wordt een 16 bits adres gevormd.
In het 2^{e} voorbeeld is een 16 bits adres direct in 2 bytes gestopt. De maximale grootte van het te gebruiken adres is dan dus 2^{16} of 64K bytes.

7.4 Instructieafhandeling

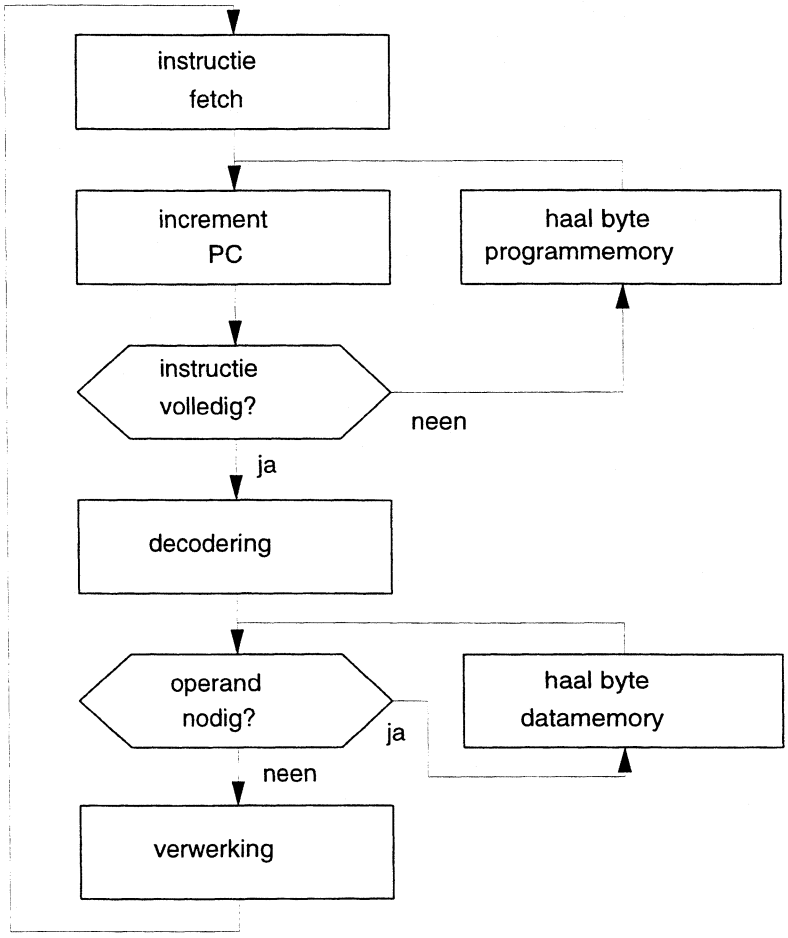
De uitvoering van een instructie is in 3 fasen te verdelen:



Afbeelding 77 instructie fasen

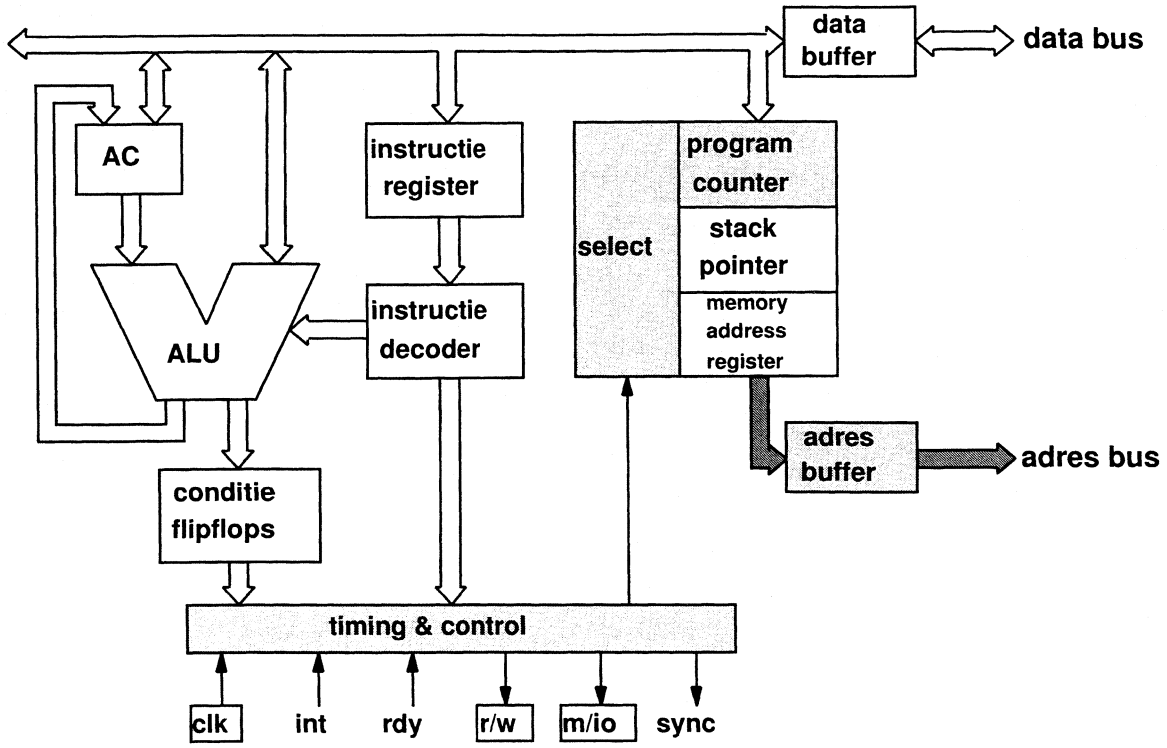
Eerst wordt de instructie opgehaald uit het (programma)geheugen en naar het instructieregister gebracht. Dit wordt de instructiefetch genoemd. Daarna wordt de instructie gedecodeerd en omgezet in besturingssignalen. In de laatste fase, de executie, wordt de instructie uitgevoerd of geexecuteerd.
Daarna vindt weer de instructiefetch van de volgende instructie plaats. Dit proces gaat door tot het uitzetten van de computer. Er zijn computers die een HALT instructie hebben, of die door controlesignalen de instructie kunnen uitrekken in tijd, maar in principe is dit een eindeloos proces. Er bestaat voor de computer geen einde van een programma, er is altijd weer een volgend getal in het geheugen dat als instructie kan worden geinterpreteerd. Zelfs als er gelezen wordt op een plaats waar geen geheugen aanwezig is, zal een "hoogohmige" bus, afhankelijk van de technologie, FF_{16} lezen en dit mooi als instructie uitvoeren.
Als de computer aangezet wordt of gereset wordt start dit proces met een instructiefetch op een vast adres, meestal het adres 0000.

Als de instructie niet in 1 geheugenwoord past zal de instructiefetch tot een aantal geheugenaanvragen aanleiding geven. Als er een fijnere verdeling, op het niveau van de geheugencycli, gemaakt wordt, ziet het er als volgt uit:

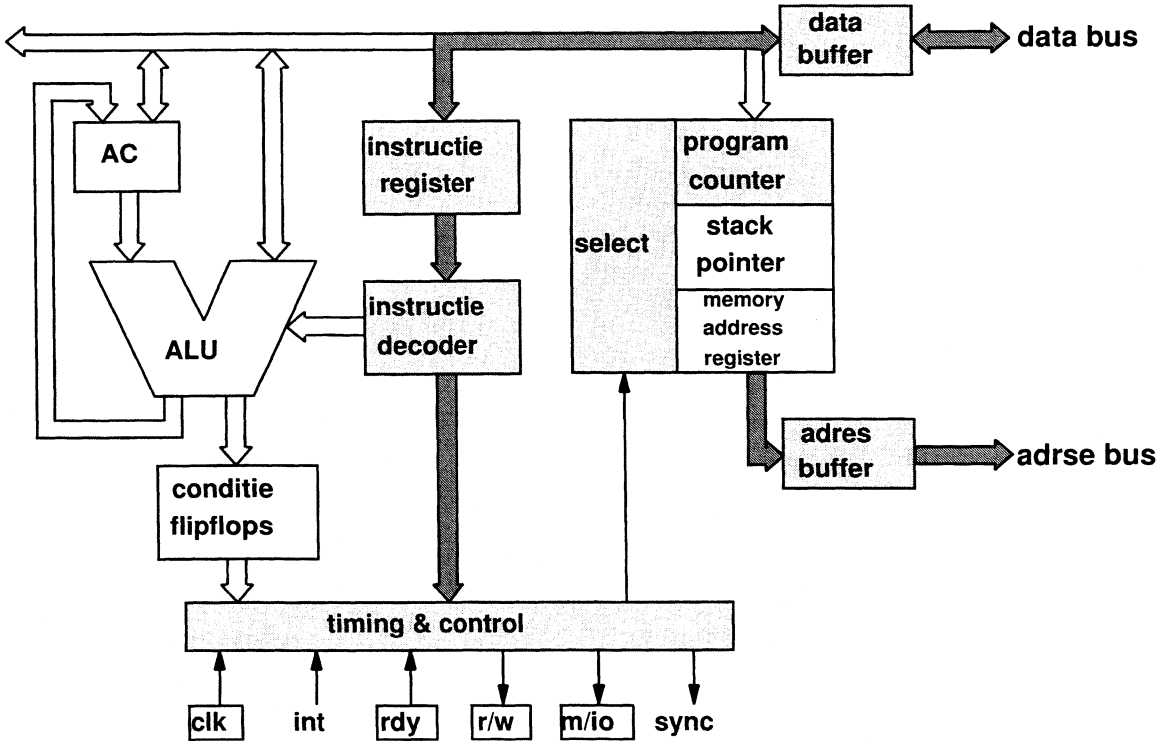


Afbeelding 78 instructieafhandeling

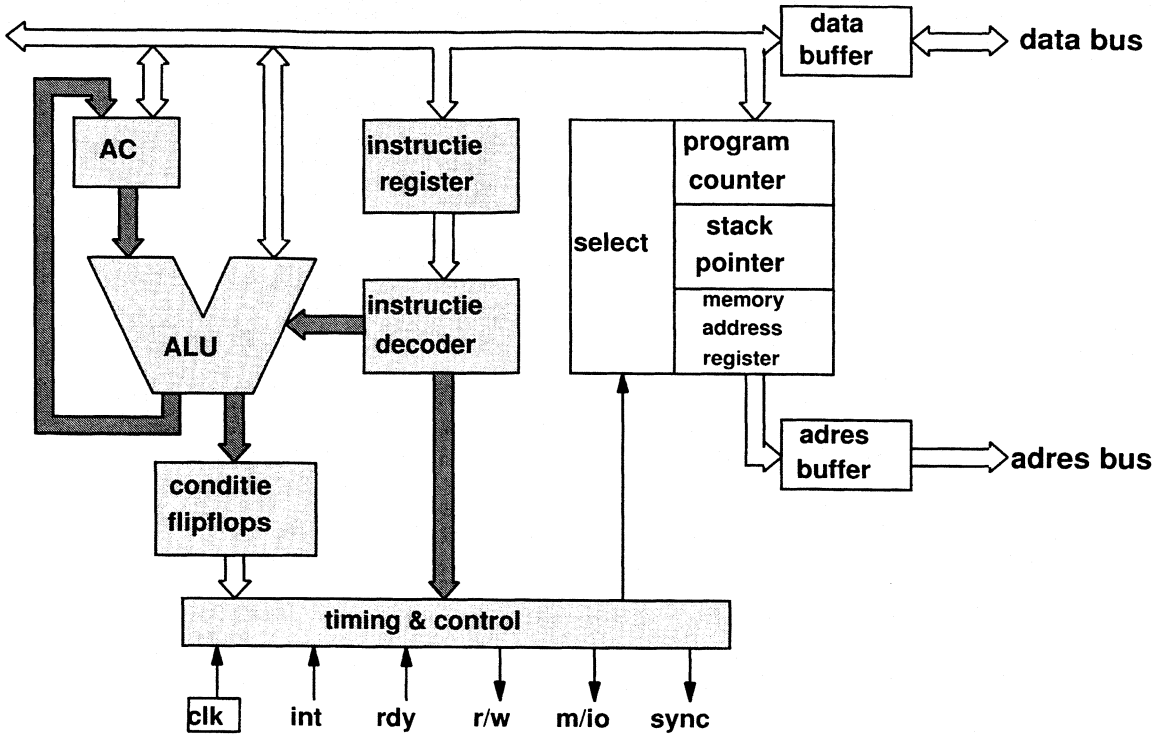
Het ophalen van de instructie vergt nu een aantal geheugenslagen. Er worden 1,2,3 of 4 bytes uit het geheugen gehaald om de instruktie volledig binnen te halen. Tijdens de uitvoering van de instructie kunnen geheugenaccessen nodig zijn om de operanden te halen of weg te schrijven of de stack te gebruiken.



Afbeelding 79 fetch 1

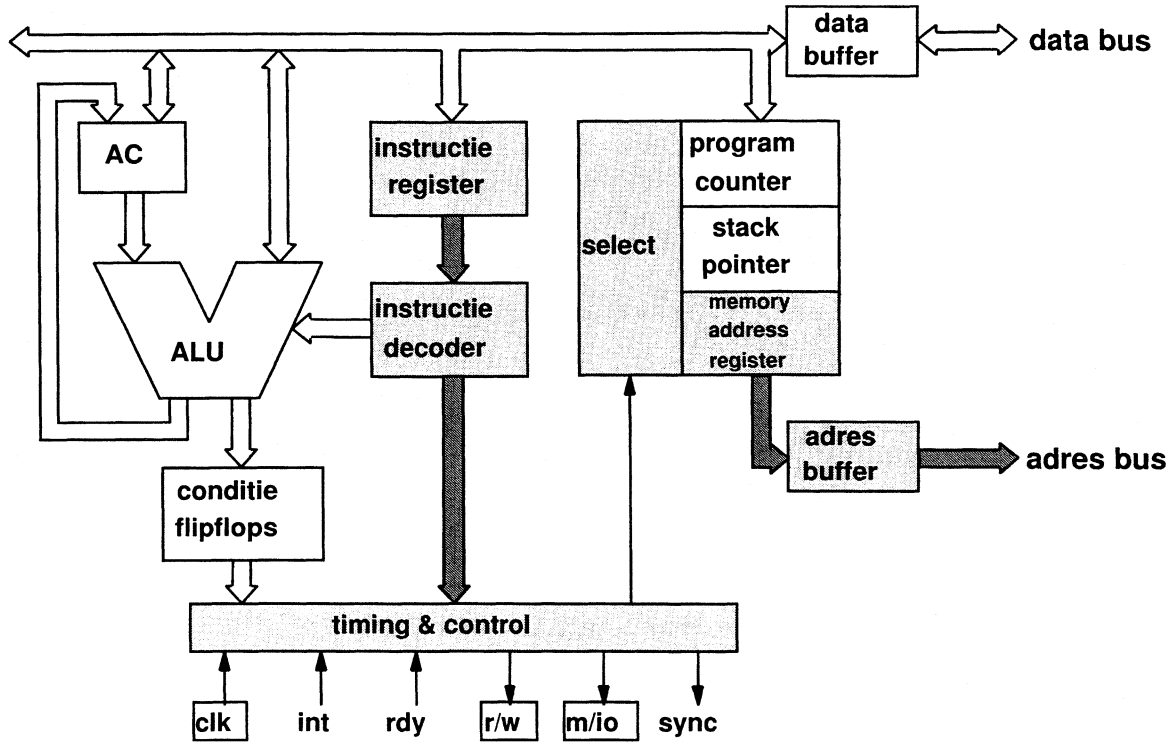


Afbeelding 80 fetch 2

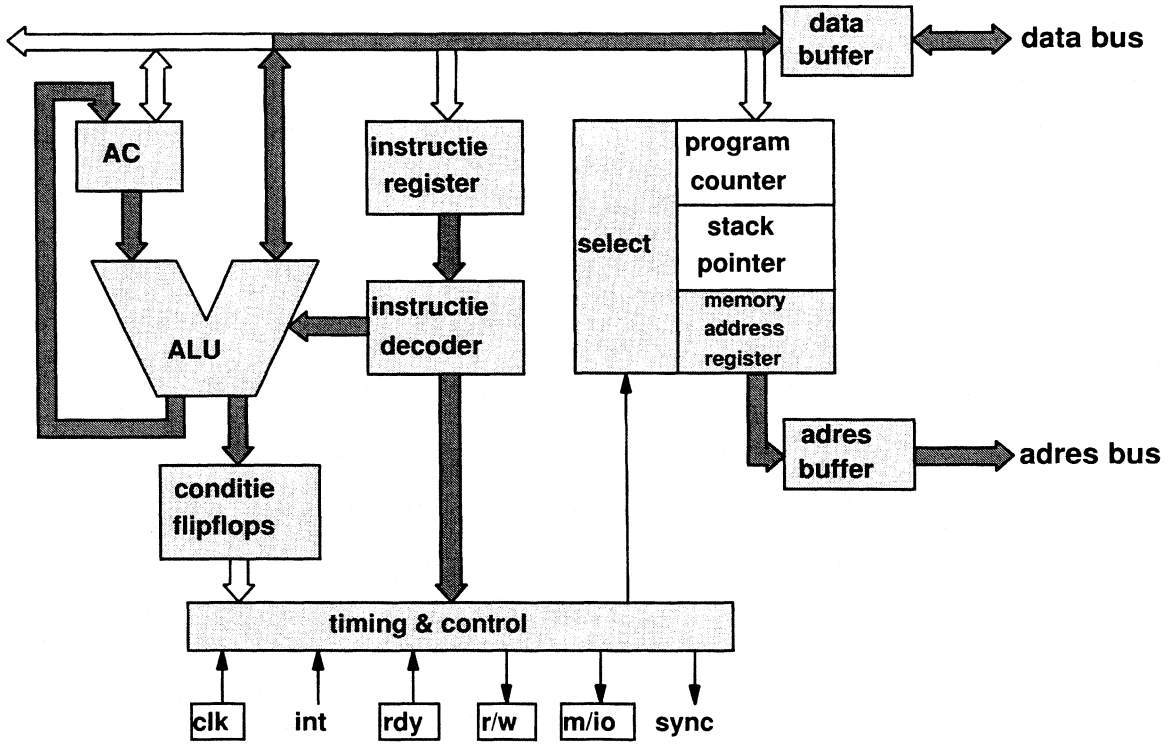


Afbeelding 81 executie INC A

In Afbeelding 79, Afbeelding 80 en Afbeelding 81 is aangegeven welke blokken binnen de computer worden geactiveerd, en welke verbindingen worden open gezet om een bepaalde instructie uit te voeren. Als voorbeeld wordt de instructie INC A gebruikt. Dit is in assemblertaal de schrijfwijze van de instructie die de ACCU met 1 verhoogt (incrementeert).
De instructie moet eerst worden opgehaald. Daarvoor wordt de PC geselecteerd om het adres te leveren. Dit adres wordt via de adresbuffer aangeboden aan het geheugen. Bovendien wordt via de control aan het geheugen duidelijk gemaakt dat het om lezen gaat (r/w signaal) en dat het uit het geheugengebied moet komen en niet uit de input (m/io signaal). (Afbeelding 79)
Na de toegangstijd van het geheugen zal via het ready signaal (rdy signaal) gemeld worden dat de gevraagde data beschikbaar is op de databus. Via de databuffer wordt de instructie in het instructie-register gezet en de decoder en de timing & control worden aan het werk gezet.
Er wordt geconstateerd dat er geen operanden nodig zijn en er kan met de executie van de instructie worden begonnen. Bovendien wordt in deze fase ook de PC met 1 verhoogd. Hiermee is de fetchfase van de uitvoering van de instructie afgesloten.(Afbeelding 80).
Voor de executie van INC A wordt de accu geselecteerd en als functie voor de ALU wordt +1 ingesteld. Het antwoord wordt teruggescreven naar de ACCU. Bij sommige machines worden ook de conditievlaggen beinvloed. Bij het 0 worden, de ZERO-flag, en bij het negatief worden, de MINUS-flag. Bij de voorbeeldmachine de 80C51 verandert geen der vlaggen bij het uitvoeren van deze instructie.(Afbeelding 81).
In alle fasen van afhandeling is ook de klok actief. Deze klok wordt via het signaal clk doorgegeven.



Afbeelding 82 operand fetch



Afbeelding 83 executie ADD A,M

In het vorige voorbeeld had de instructie geen operand uit het geheugen nodig bij het volgend voorbeeld is dit wel het geval. De volgende instructie zal de inhoud van adresplaats M optellen bij de inhoud van de accu. In assemblertaal kan dit geschreven worden als ADD A,M
De fetchfase is identiek met die van het vorige voorbeeld. Zie Afbeelding 79 en Afbeelding 80.
Het vervolg is aan gegeven in Afbeelding 82 en Afbeelding 83.
Bij de decodering is het memory address register gevuld met het adres dat door M wordt aangegeven. Voor de operand fetch kiest de besturing dit memory address register om het adres te leveren voor het ophalen van de operand. Via de adresbuffer wordt dit adres doorgegeven naar het geheugen, weer vergezeld van de besturingssignalen r/w en m/io voor respectivelijk de instellingen "lezen" en "geheugen".(Afbeelding 82).
Er wordt gewacht op het signaal rdy uit het (data)geheugen om er zeker van te zijn dat de gevraagde informatie klaar staat op de databus. Vervolgens wordt via de databuffer het getal op de interne databus gezet en wordt één der ingangen van de ALU met de interne databus verbonden. De andere ingang van de ALU is met de ACCU verbonden en als funtie voor de ACCU zal ADD worden ingesteld. Dit zal resulteren in een antwoord aan de uitgang van de ALU.
Het antwoord zal worden teruggeschreven naar de ACCU, terwijl een aantal conditievlaggen of statusvlaggen naar het conditiecode register worden weggeschreven. Meestal worden vlaggen als CARRY, OVERFLOW, MINUS en ZERO geproduceerd.(Afbeelding 83).

7.5 De instructie set

De verzameling van instructies die bij een bepaald type computer kan worden gebruikt wordt instructie set genoemd. In feite is het een lijst getallen met aan elk getal een operatie gekoppeld. De verzameling getallen noemt men de machinecode van de computer. Elk type computer heeft zijn eigen machinecode die door de ontwerper is bedacht. De operaties van de verschillend typen computers zullen op elkaar lijken, de machinecodes zijn specifiek voor een bepaalde computer, en zijn verschillend van die van andere computers. Een tijd lang deden computerfabrikanten hun best om de instructieset zo groot mogelijk te maken. Het aantal instrukties was een verkoopargument. Het is echter gebleken dat de meest gebruikte instrukties slechts een klein gedeelte van de instruktie set uitmaken. De meest "exotische" instrukties maken wel dat de computer langzamer wordt. Daarom is er nu een streven om juist minder (maar wel snel uit te voeren) instrukties in de instructie set onder te brengen. Men gaat van een CISC (= Complex Instruction Set Computer) architectuur over op een RISC (= Reduced Instruction Set Computer) architectuur.
De mogelijke operaties van een instructie set kunnen worden ingedeeld in een aantal groepen.

7.5.1 Copieer operaties

De copieer operatie dupliceert de informatie van de ene plaats binnen het systeem (source) naar een andere plaats (destination).

MOV	DST-OPR	SRC-OPR

LOAD	SRC-OPR

STORE	DST-OPR

De operanden kunnen registers of geheugenplaatsen zijn. Binnen de instructie kunnen de source operand en de destination operand van plaats verwisseld zijn. Als één van de operanden de ACCU is, dan betekent LOAD naar de ACCU halen van de OPERAND en STORE de OPERAND-plaats brengen van de inhoud van de ACCU. Sommige computers hebben nog de mogelijkheid om binnen de opcode aan te geven wat de afmeting (aantal bits) is van het te copieren getal.

7.5.2 Data bewerkings operaties

Data bewerkingen hebben 1 of 2 argumenten nodig, voor optellen zijn er 2 nodig, maar voor het incrementeren is er maar 1 nodig.

OPC	OPERAND 1	OPERAND 2

Voorbeelden van data bewerkings opdrachten:

	OPERAND 1 ERVOOR	OPERAND 2	OPERAND 1 ERNA	
ADD	01101100	01000111	10110011	geen carry
SUB	01101100	01000111	00100101	geen carry
AND	01101100	01000111	01000100	
OR	01101100	01000111	01101111	
XOR	01101100	01000111	00101011	
NEG(ate)	01101100	N.V.T.	10010100	
INC(rement)	01101100	N.V.T.	01101101	
DEC(rement)	01101100	N.V.T.	01101011	
NOT	01101100	N.V.T.	10010011	
SET	01101100	N.V.T.	11111111	
CLR	01101100	N.V.T	00000000	
ROR	01101100	N.V.T.	00110110	
ROL	01101100	N.V.T.	11011000	
ASR	11011000	N.V.T.	11101100	

Met de optelinstructie ADD kunnen 2 8-bits getallen opgeteld worden. Om grotere getallen te kunnen optellen is de ADC (Add with Carry) instructie toegevoegd:

```
  ADC          ADD
       1 ┐
00100100 │ 11101011
00111011 │ 10111010
──────── │ ──────── +
01100000 └ 10100101
```

Dit geldt ook voor het aftrekken naast de instructie SUB(tract) komt ook de instructie SUBB (SUBtract with Borrow) voor. bij het optellen van getallen groter dan 1 byte, zullen er een aantal optelinstructies worden uitgevoerd. Bij elke (sub) optelling zal ook het overflow bit een waarde krijgen, maar alleen bij de laatste optelling heeft dit betekenis.
De RO(tate)R(ight) is een "kop aan staart" verschuiving van 8 bits over 1 plaats naar rechts. Een variant is RRC (= Rotate Right with Carry), waarbij de lus met het carrybit is uitgebreid tot 9 bits. Bij de SHIFT instructies worden de bits ook 1 plaats verschoven, maar het uiterste bit gaat verloren terwijl het ontbrekende bit wordt aangevuld met een 0.
Al deze instructies komen zowel links als rechtsschuivend voor.
Een bijzondere schuifinstructie is nog de ASR (= Arithmetic Shift Right). Het getal dat dat naar rechts geschoven wordt, wordt als getal met teken beschouwd. Als het getal naar rechts geschoven wordt, gaat dit met behoud van teken. Dus het ontbrekende bit wordt aangevuld door het tekenbit. In het voorbeeld wordt -40 door de ASR instructie in -20 veranderd. Het naar rechts schuiven betekent arithmetisch door 2 delen met weggooien van 0.5.

7.5.3 Vergelijk en test operaties

Vergelijk en test operaties geven als antwoord bits in het statusregister (= Program Status Word).

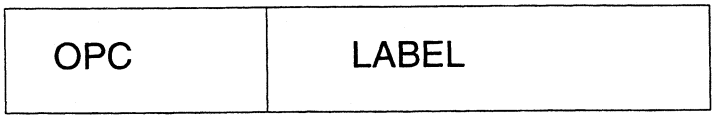
TEST	OPERAND

COMP	OPERAND 1	OPERAND 2

TEST is een vergelijking met 0. De COMP(are) instructie vergelijkt 2 operanden. Het antwoord kan = < of > zijn. Dit kan in 2 bits van het statusregister worden opgeborgen nl het ZERO of EQUAL bit en het MINUS of LESS-THAN bit. Ook bij de rekenkundige instructies worden deze bits gevormd. Maar deze instructies bepalen de statusbits zonder de operanden te veranderen.
Sommige computers hebben verschillende compare instructies voor two's complement gecodeerde getallen, unsigned getallen en floating point getallen.

7.5.4 "Control transfer" operaties

Control transfer instructies maken het mogelijk om in een programma, na een bepaalde instructie, niet de volgende, maar een instructie op een willekeurige andere plaats uit te gaan uitvoeren.



Als opcode kunt u tegenkomen JMP (jump), GOTO, BRANCH enz.
Als OPERAND hoort er een LABEL bij. Dit LABEL is ook een adres, maar niet in de dataruimte van het geheugen, maar in de instructieruimte van het geheugen. Het adres geeft de plaats aan waar de volgende instructie kan worden gevonden. En omdat het de taak van de PC is om de volgende instructie aan te wijzen, kan de werking van zo'n control transfer instructie symbolisch worden aangegeven met PC <= LABEL.
Dus:

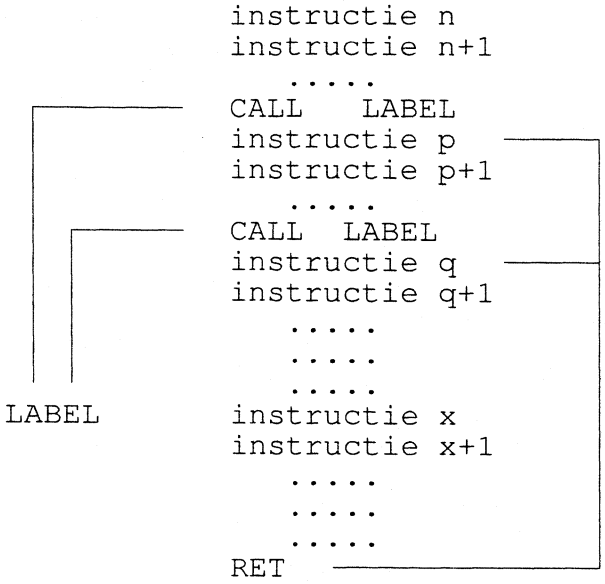
```
LABEL   instructie n
        instructie n+1
        instruktie n+2
            .....
            .....
        instructie n+x
        JMP  LABEL
```

Hiemee is het bijvoorbeeld mogelijk om een programmadeel een aantal malen te herhalen. In het voorbeeld zou dit herhalen zelfs eindeloos zijn. Daarom zijn er nog de "conditional control transfer" instructies. Deze instructies hebben de keuze of om de volgende instructie uit te voeren of naar een andere plaats te springen. Als beslissingscriterium kunnen de bits uit het statusregister worden gebruikt. Voorbeelden van instructies zijn JMPZ (spring als ZERO = 1), JMPNZ (spring als ZERO = 0), JMPGT (spring als LESS-THAN = 0 dus greater than), JMPLT , JMPC (spring als CARRY=1) enz.

```
        instructie n
        instructie n+1
            .....
        COMP    A,3
        JMPGT   LABEL
        instructie p
        instructie p+1
            .....
LABEL   instructie q
        instructie q+1
            .....
```

In bovenstaand voorbeeld wordt afhankelijk van de voorwaarde A > 3 een keuze gemaakt uit een tweetal stukken programma. Eerst wordt met een vergelijk instructie de statusbit geset, en in de volgende instructie wordt afhankelijk van de uitkomst van de vorige instructie, naar de volgende of een andere gewenste plaats gesprongen. De combinatie van testen en conditioneel springen komt

veel voor. De 80C51 microcontroller heeft een tweetal instructies waarin dit testen en selectief springen in een instructie is ondergebracht.
Dit zijn DJNZ (Decrement and Jump on Non Zero) en CJNE (Compare and Jump on Not Equal).
De CJNE instructie heeft 3 operanden, de te vergelijken getallen en de plaats waar naartoe gesprongen kan worden.
Een ander type instructie in de control transfer groep is de subroutine aanroep. Deze instructie zorgt er ook voor dat de executie op een andere plaats kan worden voortgezet, maar met de bedoeling om na het uitvoeren van een stukje programmatuur elders, weer terug te keren naar de plaats waarvandaan gesprongen werd:



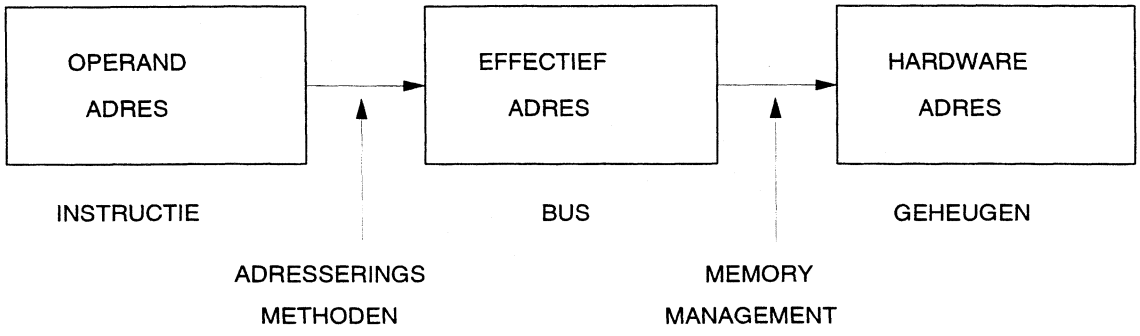
Op twee plaatsen in het voorbeeld wordt naar een stukje programmatuur gesprongen aangegeven met het adres LABEL. Aan het eind van dit stukje programmatuur wordt weer naar het (hoofd)programma teruggesprongen. Maar na de eerste aanroep is dit de instructie p en na de tweede aanroep is dat de instructie q. De RET instructie is dus een spronginstructie naar een adres wat ergens opgeborgen is. Dit opbergen vindt plaats bij de uitvoering van de CALL instruktie, immers op dat moment is het adres van het "terugkeeradres" bekend. De meestgebruikte manier om dit adres op te bergen is het op de stack te plaatsen. Naast CALL komt ook de term J(ump to)S(ubroutine)R(outine) voor. Bij sommige computers heet de RET(urn) instructie R(e)T(urn from)S(ubroutine).
Ook de CALL instructie heeft conditionele varianten.

7.5.5 Overige operaties

Sommige computers hebben speciale instructies voor input en output van gegevens, anderen zien de register van de input en de output apparaten gewoon als geheugenplaatsen. De instructies POP en PUSH geven de mogelijkheid om respectivelijk getallen van de stack te halen of getallen naar de stack te schrijven. Een ander type instructie verzorgt het aan en afzetten van de interrupt. De interrrupt is een signaal waarop de computer de normale programmaafhandeling stopt en verder gaat met een programma op een afgesproken plaats in het geheugen. Deze interruptmogelijkheid kan programmatisch worden afgezet. Een zeer bijzondere instructie is de HALT instructie, die de executie van instructies stopt. De START instructie bestaat dan ook niet.

7.6 Adresseringsmethoden

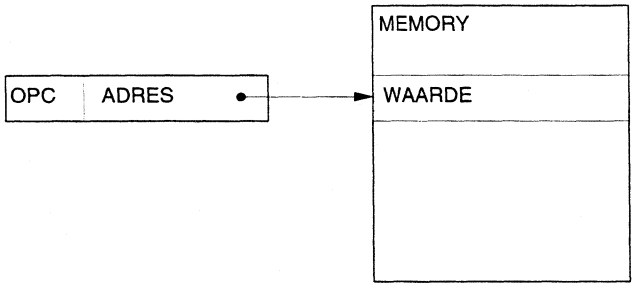
Adresseren is het aanwijzen van een plaats binnen het computersysteem. Dit kan een register, een randapparaat of een plaats in het geheugen zijn. Het aantal registers is meestal klein, ze kunnen met een beperkt aantal bits (3 à 4) worden aangegeven. Ook het aantal randapparaten is klein. Het probleem ligt bij het geheugen. Geheugens van 1 Gigabyte (= 2^{30} byte) komen voor. Maar het is niet uitvoerbaar om voor elke operand, binnen een instructie, 30 bits te reserveren. Er is binnen de instructie set al voor gezorgd het aantal operanden klein te houden, maar dit is niet voldoende. Een tweetal adres bewerkingen kan er voor zorgen dat het aantal benodigde adresseringsbits drastisch wordt verminderd.



Afbeelding 88 Adres transformaties

In Afbeelding 88 is aangegeven hoe in 2 adresmodificatiestappen het adresbereik kan worden vergroot. In stap 1 worden binnen de C.P.U. Het operand adres omgezet in een effectief adres op de bus van de computer. Dit effectieve adres kan gebruikt worden om een "hardware" geheugen aan te sturen. Maar vind men het adresbereik van dit geheugen te klein dan kan bijvoorbeeld met behulp van paginering dit bereik sterk vergroot worden. Men kan dan een "hardware" geheugen aansluiten dat vele malen groter is dan het adres bereik van het effectieve adres. In dat geval adresseert de programmeur met het effectieve adres een virtueel geheugen, dat een gedeelte is van het werkelijke geheugen. Het mechanisme dat ervoor zorgt dat het virtueel geheugen ergens een plaats krijgt in het werkelijke geheugen heet "memory-management". Dit memory-management wordt bestuurd door het Operating system, een stuurprogramma van het computersysteem. Een 2^e taak van het memory-management is om dit, mogelijk zeer grote, geheugen te verdelen over halfgeleidergeheugen en geheugen op magnetisch disk.
De adresseringsmethoden nemen stap 1 voor hun rekening.

7.6.1 Direkte adressering



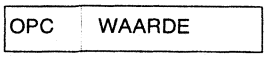
Afbeelding 89 Directe adressering

Directe adressering wordt ook wel absolute adressering genoemd. Bij deze vorm van adressering vindt geen modificatie van het adres plaats. Het effectieve adres is opgenomen in de instructie.

Meestal zal dit 2 bytes ruimte vergen. Omdat bij de 80C51 microcontroller het direct toegankelijk datageheugen maar 128 (256) bytes groot is, wordt daar, volgend op het opcode-byte, 1 adresbyte gebruikt. Met dit adres is de waarde van de operand binnen het geheugen aan te wijzen.

7.6.2 Onmiddellijke adressering

Bij onmiddelijke adressering of "immediate addressing" bevat de instructie helemaal geen adres, maar de waarde van de operand zoals in Afbeelding 90 is aangegeven.

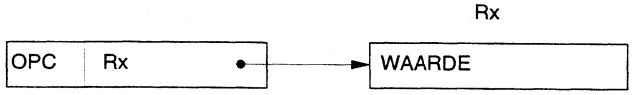


Afbeelding 90 Onmiddelijke adressering

Deze vorm van adressering kan alleen voor constanten gebruikt worden. Immers bij het gebruik als variabele zou de inhoud moeten kunnen worden veranderd, dus zou de instructie zelf moeten worden gewijzigd. Maar programma's die zichzelf modificeren zijn onbruikbaar. Bovendien staat de programmatuur van microprocessoren vaak in niet vluchtig geheugen, en dan is het zelfs niet mogelijk om de instructies in "run time" te wijzigen.
Bij de 80C51 vergt onmiddelijke adressering 1 byte extra.

7.6.3 Registeradressering

Bij registeradressering bevindt de waarde van de operand zich in een register.

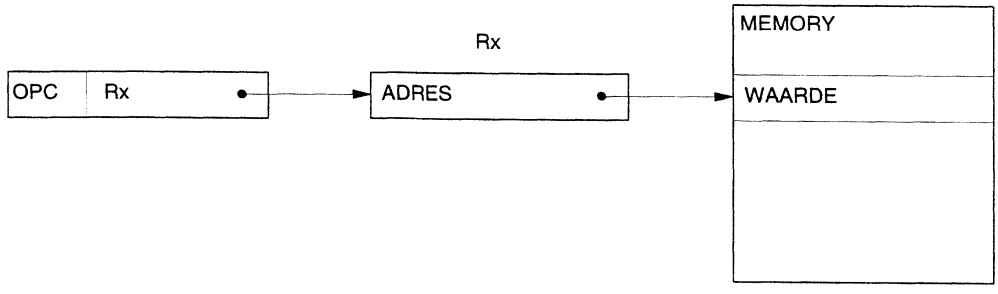


Afbeelding 91 Register adressering

De 80C51 heeft 4 sets van 8 registers. Welke set van registers in gebruik is wordt bepaald door 2 bits in het Program Status Word. In de instructie wordt 1 van de 8 registers gekozen. De 3 bits die voor deze adressering nodig zijn, zijn opgenomen in het opcode-byte. Dit geeft 1 bytes instructies.

7.6.4 Indirekte adressering

Bij indirecte adressering wordt op de aangewezen plaats geen waarde, maar opnieuw een adres gevonden. Dit 2^e adres geeft de plaats van de waarde aan. Afhankelijk van de plaats van het 2^e adres, register of geheugen, worden de termen register-indirect of indirect gebruikt.

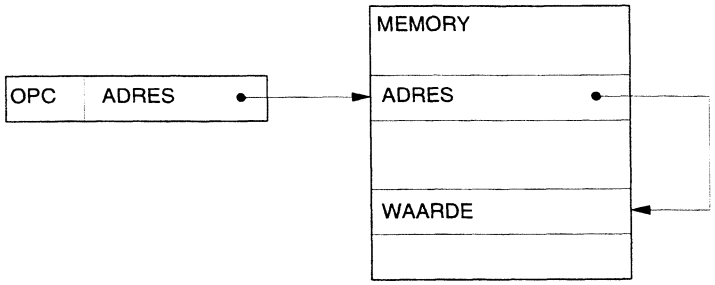


Afbeelding 92 Register indirecte adressering

Bij register-indirecte adressering is het afhankelijk van de grootte van het register hoe groot het adresbereik is.
Indirecte adressering heeft niet alleen tot doel om het aantal adresbits te beperken. Deze vorm van adresseren wordt o.a. gebruikt bij het werken met tabellen. Stel men wil op de plaatsen 100 tot 130 de inhoud met 1 verhogen.

```
Direct                    Register-indirect
------                    -----------------
INC  100                      MOV   R0,#100
INC  101                LUS   INC   @R0
INC  102                      INC   R0
........                      CJNE  R0,#130,LUS
........                      ..........
INC  129
........
```

In het voorbeeld wordt register R0 gebruikt om het adres 100 in te zetten. Door R0 indirect (@) te verhogen wordt dus de inhoud van adres 100 verhoogd. Door R0 te verhogen komt er de verwijzing naar het volgende adres te staan. Met de C(ompare and)J(ump on)N(ot)E(qual) wordt het proces voortgezet totdat er in R0 130 staat.
De andere vorm van indirecte adressering via het geheugen is aangegeven in Afbeelding 93.

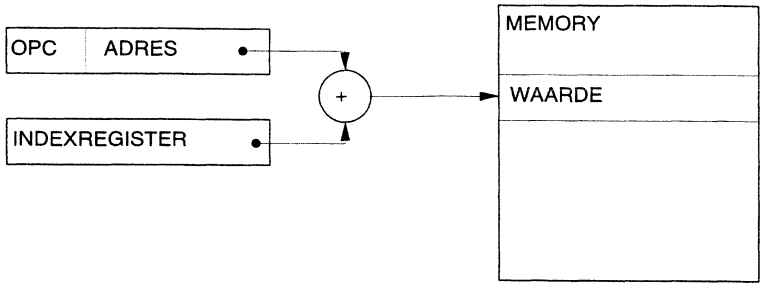


Afbeelding 93 Indirecte adressering

Hier wordt in de instructie een geheugenplaats aangewezen waar het adres van operand staat. Deze vorm van adressering komt bij de 80C51 niet voor.
Wel komt register indirecte adressering voor. De registers R0 en R1 kunnen via register-indirecte adressering een plaats in het interne datageheugen aanwijzen. Met een speciaal register DPTR kan ook het externe datageheugen geadresseerd worden. Het D(ata)P(oin)T(e)R register is 16 bits groot en met de < MOVX @DPTR,A > en < MOVX A,@DPTR > lezen en schrijven in 64K extern datageheugen. De adresseringsmethode is hier dus gekoppeld aan één opcode.

7.6.5 Geïndexeerde adressering

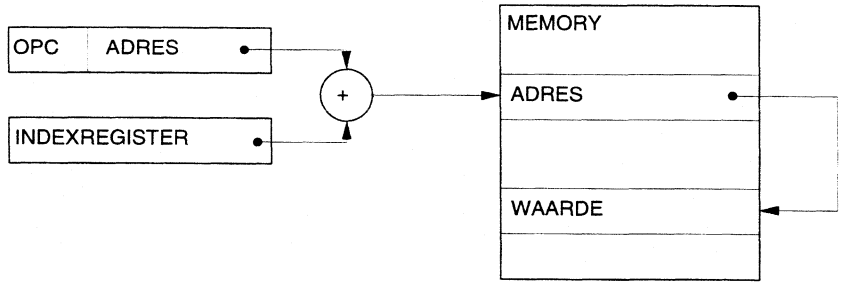
Bij indexering wordt bij een adres een getal opgeteld. Soms is hiervoor een speciaal register, het indexregister, beschikbaar. Een andere mogelijkheid is dat algemeen gebruikte registers als indexregister kunnen worden aangewezen. Deze vorm van adressering geeft evenals indirecte adressering de mogelijkheid om in tabellen te adresseren. Men adresseert m.b.v. de instructie het begin van de tabel, en zet in het indexregister een bepaalde off-set t.o.v. dit begin. Dit is aangegeven in Afbeelding 94



Afbeelding 94 Geindexeerde adressering

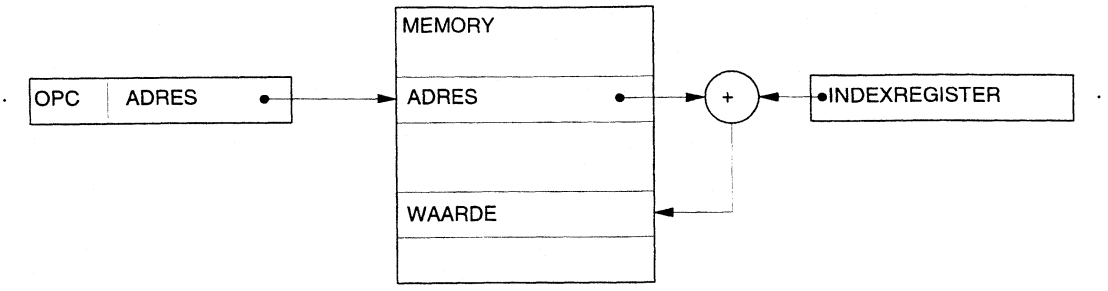
Zuiver geïndexeerde adressering heeft de 80C51 niet.

Er kunnen ook combinaties van adresseringen voorkomen. Als er zowel indirect als geïndexeerde adressering gebruikt worden. De 2 mogelijke volgordes leveren dan de geïndexeerde indirecte adressering in Afbeelding 95 en de indirect geïndexeerde adressering in Afbeelding 96 en Afbeelding 97 op.

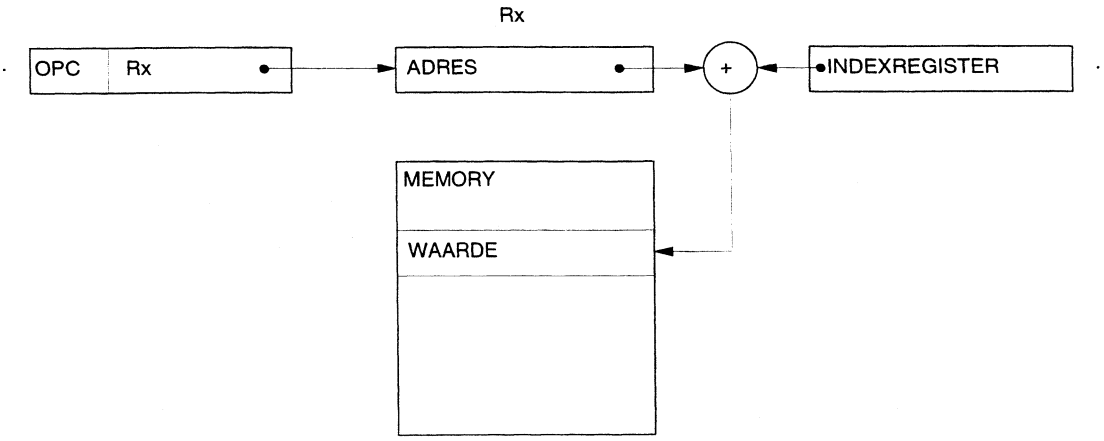


Afbeelding 95 Pre-indexing

Deze adressering behoort niet tot de mogelijkheden van de 80C51



Afbeelding 96 Post-indexing

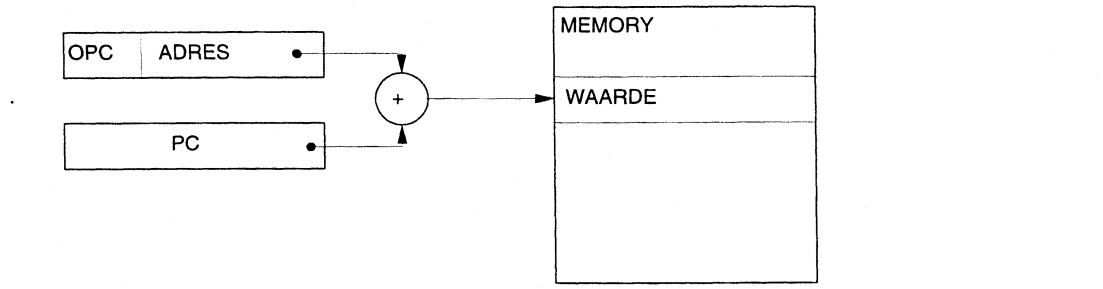


Afbeelding 97 Indirecte registeradressering met post-indexing

De register indirecte adressering met postindexing komt bij de 80C51 voor.
Het register Rx kan de datapointer DTPR of de PC zijn. Als indexregister wordt de accu gebruikt.
De instructies met dit adresseringstype zijn:
< JMP @A+DPTR >,< MOVC A,@A+DPTR > en < MOVC A,@A+PC >. De MOVC instructie kan data uit het programma geheugen halen.

7.6.6 Relatieve adressering

Relatieve adressering is een bijzondere vorm van geïndexeerde adressering. Als indexregister wordt de PC gebruikt. In Afbeelding 98 is dit aangegeven.



Afbeelding 98 Relatieve adressering

De PC wijst altijd naar de eerstvolgende uit te voeren instructie. Relatief adresseren t.o.v. de PC is

bijzonder aantrekkelijk voor spronginstructies. In programma's komen vaak korte stukjes programmatuur voor die een aantal malen moet moeten worden doorlopen. Er zal dan vanaf het eind van de lus relatief t.o.v. de PC een klein aantal adressen moeten worden teruggesprongen. Dit relatieve getal kan dus klein zijn. Deze relatieve waarde, die in de instructie is opgenomen, moet zowel positief als negatief kunnen zijn om sprongen voorwaarts en terug mogelijk te maken. Deze off-set is two's complement gecodeerd.
Bij de 80C51 wordt in een groot aantal instructies van dit type adressering gebruik gemaakt. Voor deze off-set wordt 1 byte in de instructie gereserveerd. Dit betekent dat er tot 128 plaatsen terug en tot 127 plaatsen voorwaarts gesprongen kan worden. Moet er een grotere sprong worden uitgevoerd dan zal dit met JMP AJMP of LJMP moeten.

7.6.7 Autoindexering

Met geïndexeerde adressering is het makkelijk om de inhoud van tabellen te bewerken. Met behulp van een indexregister kan een variabele off-set in de tabel worden gegeven. Als opeenvolgende adressen in de tabel moeten worden bewerkt zal het indexregister telkens met 1 moeten worden verhoogd om naar de volgende plaats in de tabel te wijzen. Deze combinatie van bewerken en verhogen van het indexregister komt zo vaak voor dat men deze twee mechanismen in één instructie heeft verenigd. Als deze adressering gebruikt wordt zal eerst de bewerking worden uitgevoerd en in dezelfde instructie het indexregister met 1 verhoogd worden. Men noemt deze vorm van adressering auto-increment. Bij de auto-decrement adressering wordt het indexregister met 1 verlaagd en daarna wordt de bewerking uitgevoerd. Het hangt van de computer af hoe de volgorde van de akties is, maar het komt wel altijd gepaard voor. Dus

	auto-increment: bewerking uitvoeren	daarna	indexregister verhogen
	auto-decrement: indexregister verlagen	daarna	bewerking uitvoeren
of			
	auto-increment: indexregister verhogen	daarna	bewerking uitvoeren
	auto-decrement: bewerking uitvoeren	daarna	indexregister verlagen

7.7 Instructieset van de 80C51

De 80C51 microcontroller heeft 255 instructies. Van de genoemde adresseringsmodes wordt slechts een klein aantal gebruikt. In de volgende tabel is het 1^{e} byte, het opcode byte, hexadecimaal aangegeven. Het aantal bytes wat nog volgt is afhankelijk van de adresseringsmode. Bij register-adressering volgt **geen** byte. Directe adressering vergt 1 byte extra relatieve adressering kost ook 1 byte. Ook de constante bij immediate adressing moet in 1 byte passen. Hierop vormt de instructie MOV DPTR,#constante een uitzondering, deze constante beslaat met zijn 16 bits waarde 2 bytes. De instructies LJMP en LCALL kunnen een sprong uitvoeren door het gehele mogelijke 64k programmageheugen, wat ook een adres van 2 bytes vergt. Daar waar in de tabel een adres van 11 bits staat aangegeven, is maar 1 byte extra nodig, omdat 3 bits in het opcodebyte verpakt zijn. Zo'n instructie komt dan ook op 8 plaatsen in de tabel voor.

00	NOP		40	JC	rel	80	SJMP	rel	C0	PUSH	dir
01	AJMP	adr11	41	AJMP	adr11	81	AJMP	adr11	C1	AJMP	adr11
02	LJMP	adr16	42	ORL	dir,A	82	ANL	C,bit	C2	CLR	bit
03	RR	A	43	ORL	dir,#c	83	MOVC	A,@A+PC	C3	CLR	C
04	INC	A	44	ORL	A,#c	84	DIV	AB	C4	SWAP	A
05	INC	dir	45	ORL	A,dir	85	MOV	dir,dir	C5	XCH	A,dir
06	INC	@R0	46	ORL	A,@R0	86	MOV	dir,@R0	C6	XCH	A,@R0
07	INC	@R1	47	ORL	A,@R1	87	MOV	dir,@R1	C7	XCH	A,@R1
08	INC	R0	48	ORL	A,R0	88	MOV	dir,R0	C8	XCH	A,R0
09	INC	R1	49	ORL	A,R1	89	MOV	dir,R1	C9	XCH	A,R1
0A	INC	R2	4A	ORL	A,R2	8A	MOV	dir,R2	CA	XCH	A,R2
0B	INC	R3	4B	ORL	A,R3	8B	MOV	dir,R3	CB	XCH	A,R3
0C	INC	R4	4C	ORL	A,R4	8C	MOV	dir,R4	CC	XCH	A,R4
0D	INC	R5	4D	ORL	A,R5	8D	MOV	dir,R5	CD	XCH	A,R5
0E	INC	R6	4E	ORL	A,R6	8E	MOV	dir,R6	CE	XCH	A,R6
0F	INC	R7	4F	ORL	A,R7	8F	MOV	dir,R7	CF	XCH	A,R7
10	JBC	bit,rel	50	JNC	rel	90	MOV	DPTR,#d16	D0	POP	dir
11	ACALL	adr11	51	ACALL	adr11	91	ACALL	adr11	D1	ACALL	adr11
12	LCALL	adr16	52	ANL	dir,A	92	MOV	bit,C	D2	SETB	bit
13	RRC	A	53	ANL	dir,#c	93	MOVC	A,@A+DPTR	D3	SETB	C
14	DEC	A	54	ANL	A,#c	94	SUBB	A,#c	D4	DA	A
15	DEC	dir	55	ANL	A,dir	95	SUBB	A,dir	D5	DJNZ	dir,rel
16	DEC	@R0	56	ANL	A,@R0	96	SUBB	A,@R0	D6	XCHD	A,@R0
17	DEC	@R1	57	ANL	A,@R1	97	SUBB	A,R1	D7	XCHD	A,@R1
18	DEC	R0	58	ANL	A,R0	98	SUBB	A,R0	D8	DJNZ	R0,rel
19	DEC	R1	59	ANL	A,R1	99	SUBB	A,R1	D9	DJNZ	R1,rel
1A	DEC	R2	5A	ANL	A,R2	9A	SUBB	A,R2	DA	DJNZ	R2,rel
1B	DEC	R3	5B	ANL	A,R3	9B	SUBB	A,R3	DB	DJNZ	R3,rel
1C	DEC	R4	5C	ANL	A,R4	9C	SUBB	A,R4	DC	DJNZ	R4,rel
1D	DEC	R5	5D	ANL	A,R5	9D	SUBB	A,R5	DD	DJNZ	R5,rel
1E	DEC	R6	5E	ANL	A,R6	9E	SUBB	A,R6	DE	DJNZ	R6,rel
1F	DEC	R7	5F	ANL	A,R7	9F	SUBB	A,R7	DF	DJNZ	R7,rel
20	JB	bit,rel	60	JZ	rel	A0	ORL	C,/BIT	E0	MOVX	A,@DPTR
21	AJMP	adr11	61	AJMP	adr11	A1	AJMP	adr11	E1	AJMP	adr11
22	RET		62	XRL	dir,A	A2	MOV	C,bit	E2	MOVX	A,@R0
23	RL	A	63	XRL	dir,#c	A3	INC	DPTR	E3	MOVX	A,@R1
24	ADD	A,#c	64	XRL	A,#c	A4	MUL	AB	E4	CLR	A
25	ADD	A,dir	65	XRL	A,dir	A5	onbekend		E5	MOV	A,dir
26	ADD	A,@R0	66	XRL	A,@R0	A6	MOV	@R0,dir	E6	MOV	A,@R0
27	ADD	A,@R1	67	XRL	A,@R1	A7	MOV	@R1,dir	E7	MOV	A,@R1
28	ADD	A,R0	68	XRL	A,R0	A8	MOV	R0,dir	E8	MOV	A,R0
29	ADD	A,R1	69	XRL	A,R1	A9	MOV	R1,dir	E9	MOV	A,R1
2A	ADD	A,R2	6A	XRL	A,R2	AA	MOV	R2,dir	EA	MOV	A,R2
2B	ADD	A,R3	6B	XRL	A,R3	AB	MOV	R3,dir	EB	MOV	A,R3
2C	ADD	A,R4	6C	XRL	A,R4	AC	MOV	R4,dir	EC	MOV	A,R4
2D	ADD	A,R5	6D	XRL	A,R5	AD	MOV	R5,dir	ED	MOV	A,R5
2E	ADD	A,R6	6E	XRL	A,R6	AE	MOV	R6,dir	EE	MOV	A,R6
2F	ADD	A,R7	7F	XRL	A,R7	AF	MOV	R7,dir	EF	MOV	A,R7
30	JNB	bit,rel	70	JNZ	rel	B0	ANL	C,/bit	F0	MOVX	@DPTR,A
31	ACALL	adr11	71	ACALL	adr11	B1	ACALL	adr11	F1	ACALL	adr11
32	RETI		72	ORL	C,bit	B2	CPL	bit	F2	MOVX	@R0,A
33	RLC	A	73	JMP	@A+DPTR	B3	CPL	C	F3	MOVX	@R1,A
34	ADDC	A,#c	74	MOV	A,#c	B4	CJNE	A,#c,rel	F4	CPL	A
35	ADDC	A,dir	75	MOV	dir,#c	B5	CJNE	A,dir,rel	F5	MOV	dir,A
36	ADDC	A,@R0	76	MOV	@R0,#c	B6	CJNE	@R0,#c,rel	F6	MOV	@R0,A
37	ADDC	A,@R1	77	MOV	@R1,#c	B7	CJNE	@R1,#c,rel	F7	MOV	@R1,A
38	ADDC	A,R0	78	MOV	R0,#c	B8	CJNE	R0,#c,rel	F8	MOV	R0,A
39	ADDC	A,R1	79	MOV	R1,#c	B9	CJNE	R1,#c,rel	F9	MOV	R1,A
3A	ADDC	A,R2	7A	MOV	R2,#c	BA	CJNE	R2,#c,rel	FA	MOV	R2,A
3B	ADDC	A,R3	7B	MOV	R3,#c	BB	CJNE	R3,#c,rel	FB	MOV	R3,A
3C	ADDC	A,R4	7C	MOV	R4,#c	BC	CJNE	R4,#c,rel	FC	MOV	R4,A
3D	ADDC	A,R5	7D	MOV	R5,#c	BD	CJNE	R5,#c,rel	FD	MOV	R5,A
3E	ADDC	A,R6	7E	MOV	R6,#c	BE	CJNE	R6,#c,rel	FE	MOV	R6,A
3F	ADDC	A,R7	7F	MOV	R7,#c	BF	CJNE	R7,#c,rel	FF	MOV	R7,A

8 Assemblerprogrammering

8.1 Symbolische operatiecodes en adressen

Eerder is genoemd hoe de instrukties voor een digitale computer eruit zien en worden verwerkt. Instrukties kunnen worden weergegeven in numerieke vorm, dus als getallen. Meestal gebeurt dit octaal of hexadecimaal, maar ook de binaire of decimale vorm is mogelijk.
Het werken met instrukties in deze vorm is erg lastig voor de programmeur, immers men moet de numerieke waarde van de instrukties kennen en zelf de adressen voor de operanden bepalen. Daarom bedenkt men bij zo'n instruktieset in getalvorm een assembler(taal). In deze assemblertaal worden de operatiecodes weergegeven als makkelijk te onthouden lettercombinaties en voor de operanden kunnen, suggestieve, namen worden gebruikt. Om deze redenen wordt deze assemblercode ook wel mnemonische code genoemd. Bij b.v. optellen is het eenvoudiger de code ADD dan de code 2x te onthouden. Zeker als men bedenkt dat het cijfer aangeduid als x nog van de adresseringsmode afhangt. Een label BEREKEN heeft voor de programmeur meer betekenis dan het adres 1022. Wel is voor het omzetten van een programma geschreven in assembler(taal) een programma nodig dat deze formele omschrijving omzet in een uit cijfers (= bits) bestaand programma (de objectcode). Dit vertaalprogramma wordt, (ter verwarring) evenals de te vertalen code, assembler genoemd.

8.2 Assemblerprogramma's

Een assemblerprogramma bestaat uit een lijst assemblerstatements. Een assemblerstatement kan een intruktie of een pseudo-instruktie, ook wel direktief genaamd, zijn. Één assemblerinstruktie wordt vertaald naar één machine-instruktie, een pseudo-instruktie of direktief levert geen machine-instruktie op, maar is slechts een aanwijzing voor de vertaler. Een voorbeeld van zo'n direktief is de opdracht om een hoeveelheid geheugen te reserveren. (Pseudo-)Instrukties bestaan uit vier delen of velden:

. het labeldeel
. het operatiedeel
. het operandendeel
. het commentaardeel

Het labeldeel wordt gebruikt als plaatsaanduiding in het programma. Het geeft bijvoorbeeld aan waar in het programma ruimte voor data moet worden gereserveerd. Ook kan het de plaats aangeven waar naartoe gesprongen wordt vanuit een ander programmadeel. De assembler vertaalt een label(naam) in een adres(=getal).

Het operatiedeel bevat de symbolische operatiecode van de instruktie, bijv. ADD. Bij de pseudo-instruktie staat er een code voor de aanwijzing, bijv. DS (= Define Storage).

Het operandendeel van de instruktie bevat 0,1 of meer aanduidingen voor de operanden van de instruktie. Het aantal operanden wordt bepaald door het type instruktie. Waarin een operand vertaald wordt hangt af van de soort adressering die wordt gebruikt. Bij een pseudo-instruktie hangt dit operandendeel af van het type pseudo-instruktie.

Commentaar wordt opgenomen om een toelichting te geven bij het betreffende programmaonderdeel. In het commentaar wordt aangegeven wat de werking op funktioneel niveau is van één of meer instrukties. Het commentaar is noodzakelijk om later veranderingen in de programmatuur aan te kunnen brengen. Dit is zeker nodig als men een verandering aan moet brengen in programmatuur die men niet zelf geschreven heeft. De bedoeling van een bepaald stuk programmatuur is dan slechts met behulp van goed commentaar te ontrafelen.

Om een globale indruk van assemblerprogramma's te krijgen, volgt een gedeelte van zo'n programma voor de 80C51 microcontroller. Hierin wordt de som van de variabelen TERM1 en TERM2 berekend in de variabele SUM. In de linker kolom staat de ermee overeenkomende machinecode in hexadecimale vorm.

```
LINE LOC  OBJ          T          SOURCE
   1 0000                TERM1      EQU  40H      ; adres TERM1
                                                   ; in data-RAM
   2 0000                TERM2      EQU  41H      ; adres TERM2
                                                   ; in data-RAM
   3 0000                SUM        EQU  42H      ; adres SUM
                                                   ; in data-RAM
   4 0000 E5 40      [1] NEXTSUM    MOV  A,TERM1  ; A:=TERM1
   5 0002 25 41      [1]            ADD  A,TERM2  ; A:=A+TERM2
   6 0004 F5 42      [1]            MOV  SUM,A    ; SUM:=A
```

De laatste drie assemblerstatements zijn instrukties, waarvan alleen de eerste een label heeft. MOV en ADD zijn operaties, TERM1, TERM2 en SUM zijn operanden. De scheiding tussen label en operatieveld is een spatie. Dus als een regel met een spatie begint wordt verondersteld dat er geen label is. Operaties en operanden worden van elkaar gescheiden door één of meer spaties. De eerste drie statements zijn pseudo-instrukties waarmee geheugen, al of niet met inhoud, wordt gereserveerd. De operanden worden van elkaar gescheiden met komma's. De puntkomma geeft aan dat het commentaarveld begint.

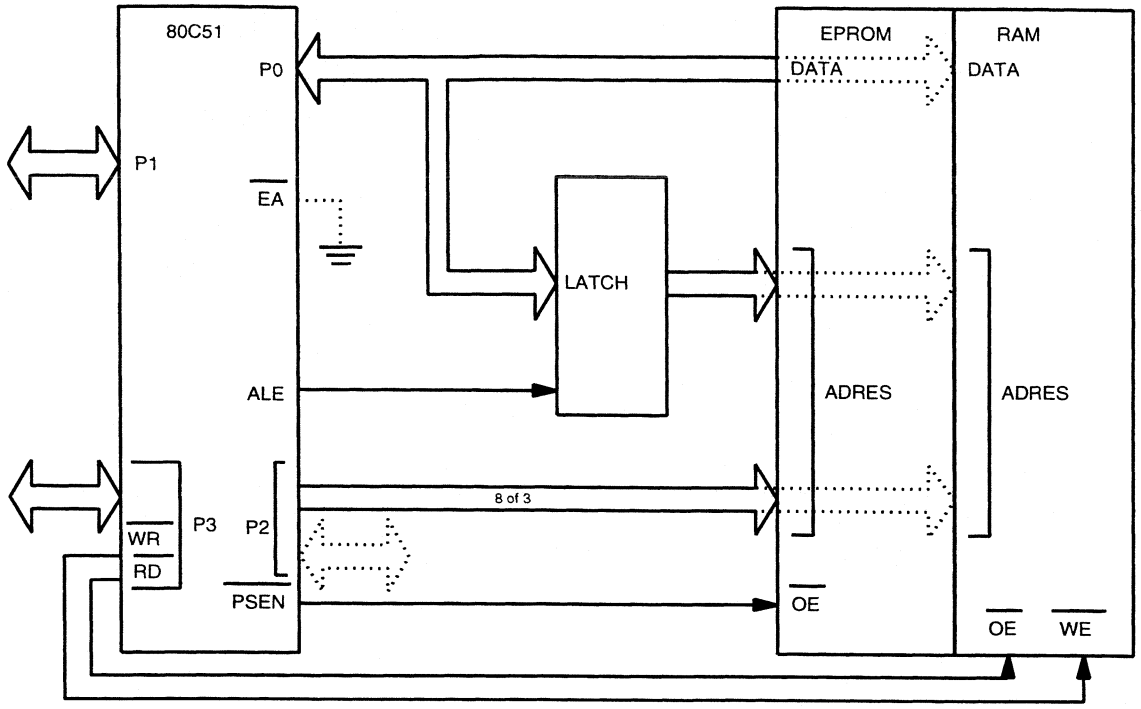
8.3 De funktie van de assembler

Omdat computers slechts uit bitreeksen bestaande machinecodeprogramma's kunnen verwerken moet een programma dat geschreven is in assemblerstatements omgezet worden in een lijst bits. Dit is een eenvoudig proces omdat één assemblerstatement omgezet wordt in één machinecodeinstruktie. Dit omzetten of vertalen kan met de hand gebeuren, maar wordt bijna altijd door een assembler of vertaalprogramma uitgevoerd. Wel moet men bedenken dat elk type computer zijn eigen machinecodeinstrukties heeft, wat betekent dat bij elk type computer een eigen assemblerprogramma hoort. Hogere programmeertalen, zoals bijv. PASCAL, zijn onafhankelijk van het type computer, het daarbij behorende vertaalprogramma, de compiler, is echter veel complexer dan een assembler. Bij een compiler wordt één statement omgezet in een aantal machinecodeinstrukties. Het codegenererende deel van een compiler is weer afhankelijk van het type computer.
Het laten uitvoeren van een programma bestaat uit drie delen:

. Het bedenken van het programma en het intypen m.b.v een gewone teksteditor. Men beschikt dan over het bronprogramma (source program).
. Het vertalen van het bronprogramma in het objektprogramma (object program) m.b.v het assemblerprogramma. Men spreekt ook wel van het omzetten van de brontaal (source language) in de doeltaal (target language).
  Het in het geheugen brengen van het objectprogramma (laden) en laten uitvoeren.

8.4 De struktuur van de 80C51 microcontroller

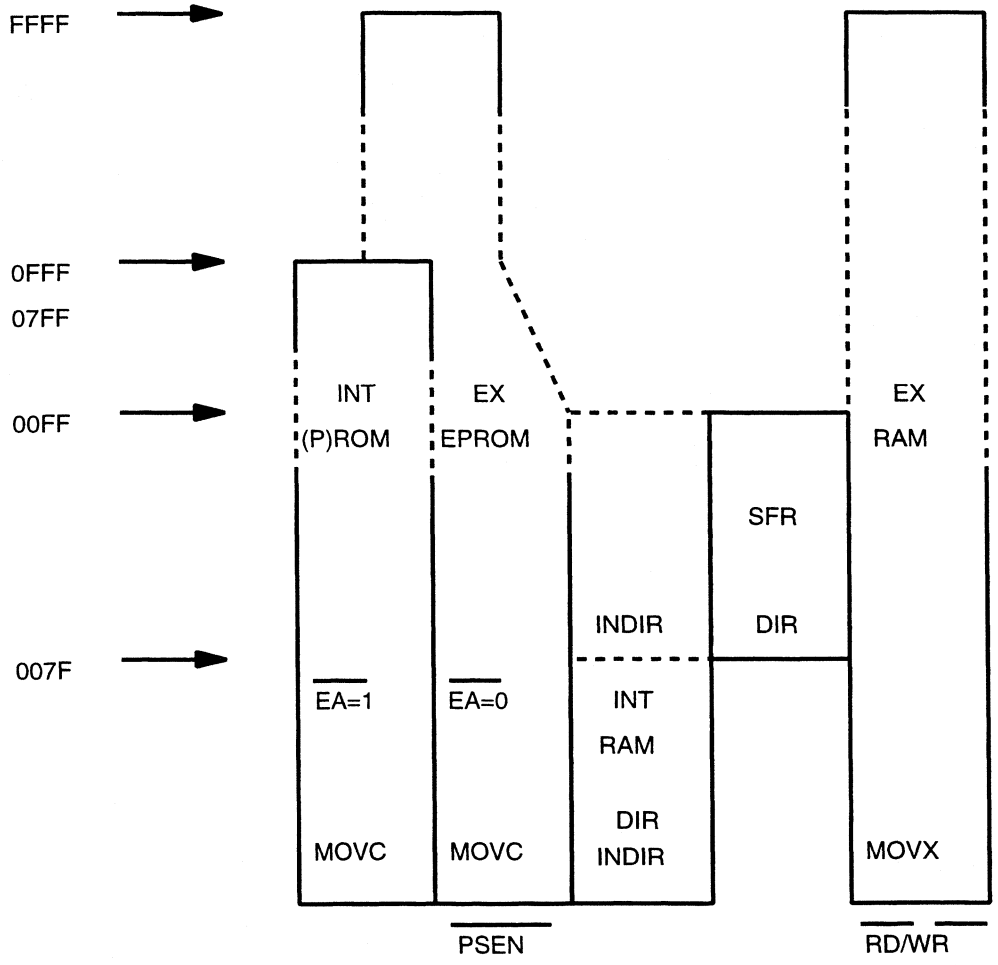
In dit college wordt als voorbeeld de assemblercode voor de 80C51 microcontroller behandeld. Om dit te kunnen begrijpen moet een globale opbouw van de 80C51 worden bekeken.
In het volgende schema zijn een aantal delen van het systeem terug te vinden:



Afbeelding 99 schema van het systeem

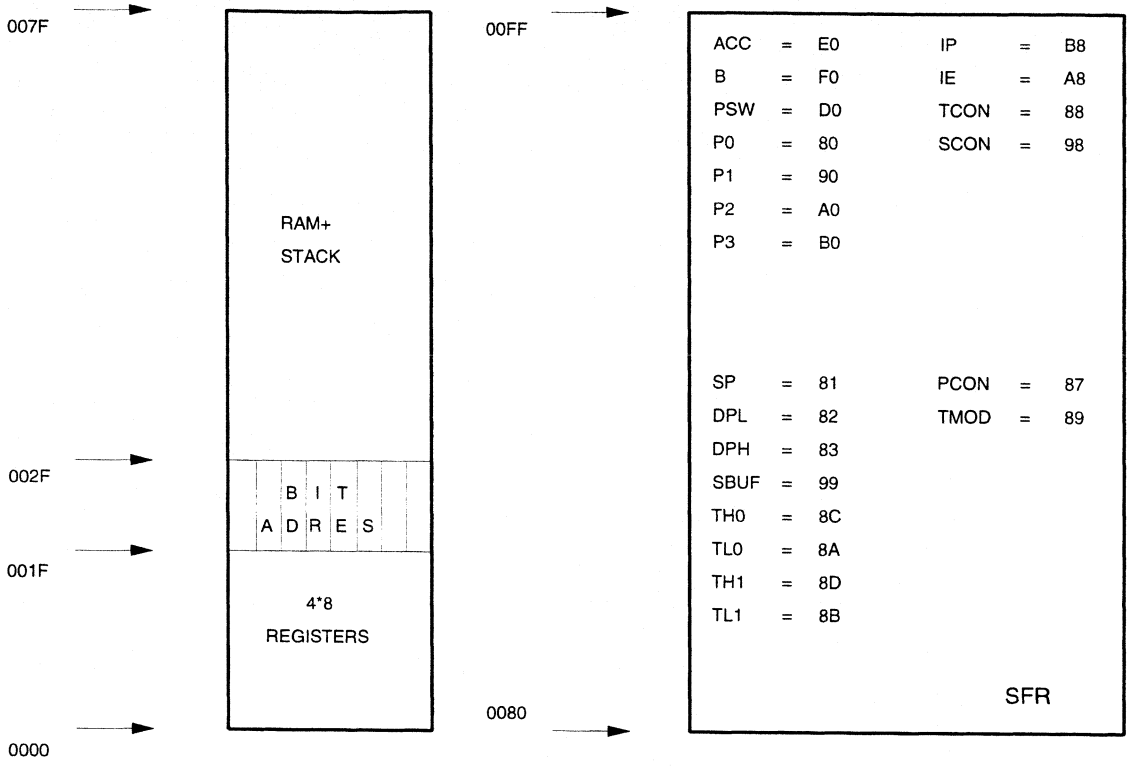
De controller is voorzien van een 4-tal 8-bits verbindingen met de buitenwereld, de poorten P0, P1, P2 en P3. In de configuratie met uitwendig geheugen wordt P0 als databus en als als adresbus gebruikt. Eerst wordt het
8-bits low-adresdeel naar buiten gestuurd, en met het commando ALE opgeslagen in de adreslatch. Vervolgens kan P0 als 8-bits databus fungeren. Het
8-bits high-adresdeel is beschikbaar via P2. Bij gebruik van extern RAM-geheugen zijn er nog 2 bits van P3 nodig om het lees en het schrijfsignaal naar het RAM-geheugen te leveren. De resterende poortsignalen kunnen voor de specifieke Input/Output van de gebruiker worden benut.
Het signaal PSEN aktiveert het uitwendige EPROM-geheugen. Het wordt opgewekt tijdens een FETCH-cycle of d.m.v de instruktie MOVC. De RD en WR signalen voor het externe RAM treden op bij uitvoering van de instuktie MOVX.

De 80C51 maakt onderscheid tussen programmageheugen en datageheugen. In de minimale configuratie is er 4K programmageheugen beschikbaar, dat is uitgevoerd als ROM. Het datageheugen bestaat uit 128 bytes RAM-geheugen (256 bytes bij de 80C52). Bovendien zijn er 128 byte-plaatsen gereserveerd voor de S(pecial) F(unction) R(egisters). Heeft men aan deze minimale configuratie (in geld en oppervlakte) niet voldoende, dan bestaat de mogelijkheid een groter type controller te kiezen, of het programmageheugen uit te breiden tot 64K ROM en het toevoegen van een DATA-(RAM)geheugen van 64k. Deze maximale configuratie is hier geschetst.



Afbeelding 100 geheugen mapping

De keuze welk programmageheugen wordt gebruikt wordt bepaald door de aansluiting EA. EA = 0 ==> externe geheugen en EA = 1 ==> interne geheugen.
Het externe RAM geheugen kan met de instruktie MOVX bereikt worden.
De 80C51 beschikt over een tweetal inwendig te adresseren stukken:
het inwendige RAM geheugen van 128 bytes en een stuk van 128 bytes dat
S(pecial) F(unction) R(egister) wordt genoemd. In deze twee delen zijn de registers van de machine ondergebracht. Het volgende plaatje laat de indeling zien:



Afbeelding 101 RAM-indeling

Er zijn 4 sets van 8 registers beschikbaar. Welke set van 8 registers in gebruik is wordt bepaald door 2 bits in het P(rogram) S(tatus) W(ord), een register dat zelf een plaatsje vind op adres D0 van het SFR. De 4*8 machineregisters liggen in de eerste 32 bytes van het interne RAM-geheugen. De volgende 16 bytes (= 128 bits) zijn ook afzonderlijk als bits te adresseren. De belangrijkste registers in het SFR zijn de accumulator A (E0), het register B (F0) dat bij vermenigvuldigen en delen wordt gebruikt, het P(rogram)S(tatus)W(ord) (D0), de poorten P0 (80), P1 (90), P2 (A0), P3 (B0), S(tack)P(ointer) (81) en D(ata)P(ointer)H(igh)+D(ata)P(ointer)L(ow) (83+82) dat als 16bits adresregister dienst doet. Ook in dit SFR gebied zijn 16 bytes (= 128 bits) afzonderlijk als bits te adresseren, en wel die waarvan het adres eindigt op 0 of 8. Zo kunnen bijv. de afzonderlijke bits van het P(rogram)S(tatus)W(ord) worden bekeken:
PSW.7: carryflag van de accu, PSW.6: Auxiliary carry (= carry van bit3 naar bit4) van de accu, PSW.5: general purpose flag, PSW.4 en PSW.3: pointer naar de in gebruik zijnde registerbank, PSW.2: overflowflag, PSW.1: te definieren flag en PSW.0: pariteitsbit van de accumulator. Omdat bij andere typen controllers naast het SFR ook intern RAM op dezelfde adressen kan voorkomen, kan de SFR alleen met directe adressering worden bereikt, terwijl het overeenkomstige interne RAM geheugen alleen met indirecte adressering te gebruiken is.

8.5 De instrukties van de 80C51

Bij de 80C51 kunnen een aantal groepen instrukties worden onderscheiden:
1. De aritmetische instrukties.
2. De logische instrukties.
3. De copieer of verplaatsinstrukties intern.
4. De copieer of verplaatsinstrukties extern.
5. De Boolean instrukties.
6. De onvoorwaardelijke en de voorwaardelijke spronginstrukties.

De bewerkinsopdrachten die op twee operanden werken hebben de algemene vorm: OPERATIE OPERAND1,OPERAND2 waarbij OPERAND1 tevens de plaats van het antwoord is, waardoor de oorspronkelijke inhoud verloren gaat.
Van de in hoofdstuk 5 behandelde adresseringsmethoden worden er in de 80C51 slechts een klein aantal gebruikt. Ook zijn de mogelijke adresseringsvormen sterk afhankelijk van de operatiecode. Vergelijkt men de operaties "optellen" en "and" met elkaar dan ziet men dat de laatste meer adresseringsmogelijkheden voor de destination-operand heeft. Als allerlei combinaties van bewerkingen en adresseringen kunnen voorkomen spreekt men van een orthogonale instruktieset. Bij de 80C51 (evenals de meeste Intel processoren) is dit zeker niet het geval. Daarom zullen de verschillende combinatisch apart worden bekeken.
. Registeradressering: bij een aantal opcodes is registeradressering mogelijk, de operand kan worden aangegeven met R0,R1,...,R7. Binnen de instruktie worden hiervoor 3 bits gebruikt. In de volgende instruktielijstjes wordt registeradressering aangegeven met "reg".
. Directe adressering: kan worden gebruikt om een plaats in de 128 bytes interne RAM of in het 128 bytes SFR aan te geven. In de instruktie kost zo,n adres 1 byte. In de lijstjes wordt het aangegeven met "dir".
. Indirecte adressering komt slechts voor als registerindirecte adressering. De registers R0 en R1 kunnen een 8 bits adres bevatten, en op die manier een plaats in het interne RAM aangeven. Het SFR kan **niet** op deze manier worden geadresseerd om onderscheid met intern RAM op de plaatsen 80 tot FF mogelijk te maken. In de lijstjes wordt het aangegeven met "indir".
Het register DPTR kan een 16 bits adres bevatten, en wordt gebruikt bij de MOVX instruktie om het externe RAM te kunnen adresseren.
. Immediate adressing maakt gebruik van 8 bits constanten en wordt aangegeven met "imm". Slechts in de instruktie MOV DPTR,constante kan van een 16 bits constante gebruik worden gemaakt.
. Indexed adressing wordt uitsluitend gebruikt om programmageheugen te adresseren. Als indexregister kan alleen de accu worden gebruikt, terwijl het basis adres zich in de DPTR of in de PC bevindt. In het laatste geval kan men van relatieve adressering spreken.
Nu volgen de mogelijke instrukties van de 80C51 gesorteerd naar soort bewerking.

1. De aritmetische instrukties:

SOORT	MNEMONIC	WERKING
increment	INC A|reg|dir|indir|DPTR	oper:=opr+1
decrement	DEC A|reg|dir|indir	oper:=opr-1
add	ADD A,reg|dir|indir|imm	A:=A+oper
add met carry	ADDC A,reg|dir|indir|imm	A:=A+oper+C
subtract	SUBB A,reg|dir|indir|imm	A:=A-oper-C
multiply	MUL AB	B,A:=A*B
divide	DIV AB	A:=int(A/B) B:=mod(A/B)
decimal adjust	DA A	A(hex):=A(decimaal)

De ADDC wordt gebruikt voor optellingen van getallen die groter zijn dan 8 bits. Een Carry die is ontstaan uit de optelling van de low-order bytes van het getal kan met deze instruktie meteen worden verwerkt in de high-order bytes van het getal.

2. De logische instrukties:

SOORT	MNEMONIC	WERKING
and op bytes	ANL A,reg|dir|indir|imm ANL dir,A|imm	A:= A and oper dir:= dir and oper
or op bytes	ORL A,reg|dir|indir|imm ORL dir,A|imm	A:= A or oper dir:= dir or oper
exclusive or op bytes	XRL A,reg|dir|indir|imm XRL dir,A|imm	A:= A exor oper dir:= dir exor oper
clear	CRL A	A:= 0
complement	CPL A	A:= not A
rotate left	RL A	roteer A linksom
rotate left w Carry	RLC A	roteer A+C linksom
rotate right	RR A	roteer A rechtsom
rotate right w Carry	RRC A	roteer A+C rechtsom
swap	SWAP A	A[0,3] <--> A[4,7]

3. De copieer of verplaatsinstrukties intern:

SOORT	MNEMONIC	WERKING

move	MOV A,reg|dir|indir|imm
	MOV reg,A|dir|imm
	MOV dir,A|reg|dir|indir|imm
	MOV indir,A|dir|imm
	MOV DPTR,imm16
		A:=oper
		reg:=oper
		dir:=oper
		indir:=oper
		DPTR:=oper

push	PUSH dir	oper op stack
pop	POP dir	oper van stack
exchange	XCH A,reg|dir|indir XCHD A,indir	A <--> oper A[0,3] <--> oper[0,3]

4. De copieer of verplaatsinstrukties extern:

SOORT	MNEMONIC	WERKING

move data	MOVX A,@reg|@DPTR
	MOVX @reg,A
	MOVX @DPTR,A
		A:= oper
		dir:= oper
		ext:= oper

move constant	MOVC A,@A+DPTR|@A+PC	A:= oper

5. De Boolean instrukties:

De Boolean instrukties werken allen met operanden van één bit. Het carry-flagbit bij de Boolean instrukties heeft ongeveer dezelfde functie als het accu-register bij de byte-instrukties.

SOORT	MNEMONIC	WERKING
and op bits	ANL C,bit|/bit	C:= C and oper
or op bits	ORL C,bit|/bit	C:= C or oper
move bits	MOV C,bit MOV bit,C	C:= bit bit:= C
clear bit	CLR C CLR bit	C:= 0 bit:= 0
set bit	SETB C SETB bit	C:= 1 bit:= 1
complement bit	CPL C CPL bit	C:= not C bit:= not bit

6. De onvoorwaardelijke en de voorwaardelijke spronginstrukties:

In de nu volgende spronginstrukties komen de instrukties AJMP en ACALL voor. Deze instrukties voorzien maar in 11 adresbits. Deze 11 bits doen dienst als de 11 laagwaardige bits van een 16 bits adres. De ontbrekende 5 bits zijn de 5 hoogstwaardige bits van de PC. Zo'n sprong blijft dus altijd binnen het gebied van 2k vanwaar uit gesprongen wordt.

SOORT	MNEMONIC	WERKING
jump unc.	SJMP reladr AJMP addr11 LJMP addr16 JMP @A+DPTR	PC:= PC +/-127 PC:= PC[15,11],oper PC:= oper PC:= A+DPTR
call unc.	ACALL addr11 LCALL addr16	PC:= PC[15,11],oper PC:= oper
return	RET RETI	uit subroutine uit interuptr.
niets	NOP	PC:=PC+1
jump cond.	JC reladr JNC reladr JB bit,reladr JNB bit,reladr JBC bit,reladr JZ reladr JNZ reladr	PC:= PC+/-127 als C=1 PC:= PC+/-127 als C=0 PC:= PC+/-127 als bit=1 PC:= PC+/-127 als bit=0 PC:= PC+/-127 als bit=1 bit:=0 PC:= PC+/-127 als ACC=0 PC:= PC+/-127 als ACC<>0
decrement+jmp	DJNZ reg,reladr DJNZ dir,reladr	reg:=reg-1|dir:=dir-1 PC:=PC+/-127 als reg|dir<>0
compare+jump	CJNE A,dir|imm,reladr CJNE reg,imm,reladr CJNE indir,imm,reladr	PC:= PC+/-127 als A<>opr PC:= PC+/-127 als reg<>opr PC:= PC+/-127 als indir<>opr

8.6 De vormen van adressering van de 80C51

Onmiddellijke adressering:
De operand is een constante. Dit wordt aangegeven door er "#" voor te zetten. De constante kan, afhankelijk van de instruktie, een getal van maximaal 255 (8 bits) of 65535 (16 bits) zijn.
Voorbeeld:

```
#123
#-7
```

Symbolische namen met het voorvoegsel "#" kunnen ook gebruikt worden. Als de symbolische naam als een constante is gedeclareerd, wordt de waarde van de constante genomen. Een constante kan gedeclareerd worden met de pseudoinstruktie

```
        naam   EQU  constante
of      naam   EQU  expressie
```

Hierin staat naam in het labelveld van de instruktie. De expressie geeft een berekenbaar getal aan.

Voorbeeld:

```
MAX     EQU 4          ;declaratie van de constante MAX
        ....
        ....
        #MAX           ;geeft de waarde 4
of
MIN     EQU 3+7*2
        ....
        ....
        #MIN           ;geeft de waarde 17
```

Als een programmalabel als constante gebruikt wordt zal dit meestal een 16 bits waarde opleveren. De waarde wordt immers bepaald door de plaats in het programma, wat een adres van 0 tot 65535 kan zijn.
Voorbeeld:

```
NEXTSUM  MOV......
         .........
         #NEXTSUM  ;geeft de waarde, het adres van NEXTSUM.
```

Het gebruik van een symbolische naam i.p.v. de constante zelf geeft meer inzicht in de bedoeling van de constante. Als de constante een aantal malen voorkomt in het programma, en men wil deze constante later veranderen heeft het bovendien het voordeel dat slechts de declaratie van de constante behoeft te worden veranderd en niet de constante zelf op alle plaatsen in het programma waar de constante gebruikt wordt.

Directe adressering:
Voorbeeld:

```
        MOV  COUNT,#11
```

Hierin geeft COUNT een adres aan dat bij de 80C51 onder de 256 moet liggen. Het kan een label zijn die in het interne RAM of SFR ligt, of de waarde is bepaald door een EQU statement.

Registeradressering:

Registers worden aangegeven door hun naam:
R0, R1, R2, R3, R4, R5, R6, R7, R8
A, SP, B, DPTR, PC enz.
Voorbeeld:

```
DIFFER   EQU   5           ;declareer constante 5
                ....
RESULT   EQU   30H         ;adres van 1 byte ruimte
                ....       ;in het data-RAM
         MOV   A,R4        ;A:=R4
         SUBB  A,#DIFFER   ;A:=A-DIFFER
         MOV   RESULT,A    ;RESULT:=A
```

Indirekte adressering komt alleen voor als registerindirect. Als registers zijn te gebruiken de 8-bits registers R0 en R1 en het 16 bits DPTR register. Indirecte adressering wordt aangegeven door "@" voor het betreffende register te zetten. Er wordt dan niet de inhoud van het register, maar de inhoud van het adres dat in het register staat als operand genomen.

Voorbeeld:

```
 EVENTS   EQU     30H          ;adres van variabele EVENTS
          ....                 ;in data-RAM
          MOV     R1,#EVENTS   ;R1:= adres van EVENTS
          INC     @R1          ;EVENTS:=EVENTS+1
of
 TABEL    EQU     4000H        ;declareer constante TABEL
          ....                 ;als 4000 hexadecimaal
          MOV     DPTR,#TABEL  ;DPTR:= adres 4000H
          MOVX    @DPTR,A      ;adres 4000:=A
```

Indexering:

Geïndexeerde adressering komt alleen voor bij de MOVC en de JMP instruktie, de MOVC kan gegevens uit het code (= programmageheugen) halen, terwijl de JMP de volledige 64k programmageheugen omvat. Er is geen instruktie die naar het programmageheugen kan schrijven, omdat het programma verondersteld wordt in ((E)P)ROM te staan. De 3 vormen die voor kunnen komen zijn:

```
MOVC   A,@A+DPTR
MOVC   A,@A+PC
JMP    @A+DPTR
```

De inhoud van het 16 bits DPTR-register of het 16 bits PC-register wordt opgeteld bij de inhoud van het 8 bits accumulator-register. Het resultaat van de optelling levert het adres van de operand. De gelezen waarde keert terug in de accu, waardoor dus de oorspronkelijke offset verloren gaat. Voorbeeld het lezen uit een tabel met constanten:

```
MOV    DPTR,#TABEL       ;DPTR:=adres van TABEL
MOV    A,#OFFSET         ;A:=offset in de tabel
MOVC   A,@A+DPTR         ;A:=tabelwaarde
```

Een wat complexer voorbeeld is het lezen van een waarde uit een 2-dimensionaal array op plaats [index1,index2]:

```
MOV   DPTR,#VERWIJZING  ;DPTR:=beginadres
MOV   A,#INDEX1         ;A:=index1
SUBB  A,#DIM1_LOW       ;A:=index1-low1
MOV   B,#DIM1_MUL       ;B:=mul1
MUL   AB                ;A:=(index1-low1)*mul1
ADD   A,#INDEX2         ;A:=(index1-low1)*mul1+index2
SUBB  A,#DIM2_LOW       ;A:=(index1-low1)*mul1+index2-low2
MOVC  A,@A+DPTR         ;A:=ARRAY[index1,index2]
```

De constanten VERWIJZING, DIM1_LOW, DIM2_LOW komen uit de Dope-vector die het array beschrijft. De instruktie MOV B,#constante maakt gebruik van directe adressering. Want hoewel B een register is wordt dit door de assembler omgezet in een direct adres dus MOV 0F0H,#constante, want register B is in het SFR op adres F0 terug te vinden. Verder is verondersteld dat het gevormde produkt kleiner is dan 256, zodat het antwoord geheel binnen register A past, en register B dus 0 wordt.

Het voorbeeld met een geïndexeerde jump verwerkt een case-statement, een variabele sprong op grond van een rangnummer:

```
        MOV   DPTR,#TABLE     ;DPTR:=adres van sprongtabel
        MOV   A,#INDEX        ;A:=rangnummer
        RL    A               ;A:=2*A
        JMP   @A+DPTR         ;de sprong zelf
        ....
TABLE   AJMP  GEVAL0
        AJMP  GEVAL1
        AJMP  GEVAL2
        AJMP  GEVAL3
        AJMP  GEVAL4
```

De index moet met 2 vermenigvuldigd worden omdat de instruktie AJMP addr11 uit 2 bytes bestaat.

8.7 Conventies en pseudoinstrukties van de assembler

De code die een assembler moet afleveren ligt vast door de computer waarvoor de assembler gemaakt is, maar de inputconventies voor de assembler kunnen door de maker van de assembler, binnen bepaalde grenzen , gekozen worden. Daarom zijn de conventies altijd gebonden aan één bepaalde assembler.
Labels en namen bestaan uit rijtjes lettertekens van maximaal 8 karakters, waarvan de eerste geen cijfer mag zijn. Elke naam moet uniek zijn en mag dus maar éénmaal gedefinieerd worden (als label of als constante). De naam moet zo gekozen worden dat de betekenis in het programmadeel duidelijk wordt.
Voorbeelden:

```
FOUT
TERM1
FACTOR
SOM
RESULT6
```

Getallen kunnen decimaal, hexadecimaal of binair zijn.
Decimale getallen bestaan uit een rijtje cijfers al of niet met een + of - voorafgegaan.
Hexadecimale getallen onderscheiden zich door een H erachter te plaatsen.
Bovendien moet er als ze met een letter beginnen nog een 0 voor gezet worden om ze van labels en namen te kunnen onderscheiden.
Binaire getallen worden gekenmerkt door er een letter B achter te zetten.
Een karakter wordt aangegeven door het teken tussen enkele quotes te zetten. Rijtjes karakters of strings worden gevormd door het rijtje karakters tussen enkele quotes te zetten, maar in het laatste geval kunnen ze alleen gebruikt worden in de nog komende DB pseudo-instruktie.
Bitadresseringen kunnen alleen worden toegepast op de adressen 20H t/m 2FH in het interne RAM en op de adressen in het SFR die eindigen op 0 of 8. In assembler wordt bitadressering aangegeven door na het adres een punt en het bitnummer in het byte te zetten, dus bijv. 39.5 of 27H.5 of 00100111B.5 wat in alle drie gevallen 3DH als bitadres oplevert nl de laatste 5 bits van het byteadres gevolgd door het bitnummer. Hier levert dit 00111-101 -> 3DH
op. In het SFR kunnen alleen adressen die op 0 of 8 eindigen bitgeadresseerd worden. In dat geval worden de drie laatste nullen van het adres vervangen door het bitnummer. Dus 0C0H.3 levert 0C3H als bitadres.

Voorbeelden:

```
       decimaal:   1234
                   -55
                   +12
   hexadecimaal:   333H
                   -3AH
                   +0AAH
         binair:   10101110B
                   10B
       karakter:   'z'
                   'tekst'
                   '1234'
bitadressering:    0E0H.7   (het voorste bit van de accu)
                   0D0H.4   (bit 4 van het Program Status Word)
```

Aritmetische expressies:
I.p.v. getallen kunnen ook aritmetische expressies worden gebruikt. Deze worden door de assembler uitgerekend. Een expressie is

. een getal
. de naam van een constante (bijv een label)
. een karakterconstante
. $ (dat is de waarde van de PC op de plaats waar de $ staat)
. - expressie
. expressie + expressie
. expressie - expressie
. expressie * expressie
. expressie / expressie (het integerdeel van de deling)

Pseudoinstrukties:

```
ORG  expressie            (origin)
```

De waarde van de expressie geeft aan op welk adres het op ORG volgende programma moet beginnen.

```
DS   expressie            (define storage)
```

De waarde van de expressie geeft aan hoeveel bytes geheugen moeten worden gereserveerd vanaf het punt waar DS staat.

```
DB  expr,expr,expr,...    (define byte)
```

De expressies worden geëvalueerd en de waarden als inhoud van de te reserveren bytes gebruikt.

```
LABEL  EQU  expressie     (equate)
```

De waarde van de expressie wordt verbonden met de labelnaam.

```
END
```

Deze instruktie markeert het einde van de te vertalen assemblertekst.
Pas op het heeft niets te maken met het beëidigen van het programma.

De assembler heeft 2 soorten output:

1. De listing, het orginele programma met daarnaast de geheugennummers met de hexadecimaal weergegeven inhouden.
2. De objektcode-file, die geschikt is om het computerprogramma te vullen met bitpatronen.

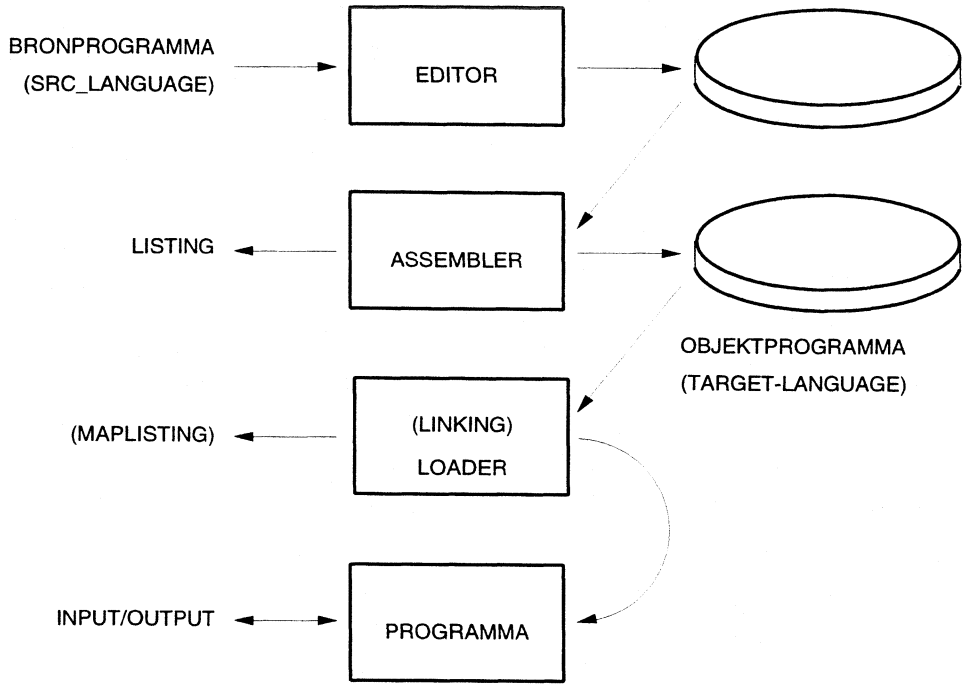
Voorbeeld van een listing:

```
 LINE LOC  OBJ         T                SOURCE
    1 0000                  ;             VOORBEELD
    2 0000                  ;
    3 0000                          ORG    1000H
    4 1000 E8          [1] START    MOV    A,R0
    5 1001 22          [2]          RET
    6 1002 31 32 33       TEXT     DB     '1234',0
           34 00
    7 1007                  ;
    8 1007                          END
symboltable (2 symbols):
      START :1000         TEXT :1002
```

Onder LOC staan de geheugenadressen in hexadecimale vorm. Onder OBJ staan de bijbehorende inhouden ook in hexadecimale vorm. T geeft aan het aantal machinecycli dat voor de uitvoering van d instruktie nodig is. Bij een klokpulsfrequentie van 12Mhz geeft het de tijd in microseconden aan. (één cyclus kost 12 klokpulsen).

8.8 Werken met de assembler

Om een werkend programma in het geheugen van de computer te krijgen, wordt naast de assembler nog van een tweetal programma's gebruik gemaakt, nl de EDITOR en de (LINKING) LOADER. Dit is in Afbeelding 102 aangegeven.



Afbeelding 102 Ontwikkelomgeving

Nadat het programma bedacht is wordt m.b.v. een EDITOR naar achtergrondgeheugen, bijvoorbeeld een schijf, gebracht. De EDITOR is een programma waarmee teksten kunnen worden ingetyped en verbeterd. Het in assembler geschreven programma wordt het bron- of sourceprogramma genoemd. De assembler is de source-language.
Na deze stap wordt het assemblerprogramma gestart om uit het source programma een objectprogramma samen te stellen. Dit objectprogramma bestaat dus uit lijsten getallen. De beschrijving is nu in de targetlanguage. Ook dit programma komt op de schijf te staan. De assembler kan ook een listing leveren. Dit is een afdruk van het sourceprogramma met de vertaling, meestal in hexadecimale vorm.
Dan komt de linking loader in aktie om de lijsten getallen op de schijf in nullen en enen naar het geheugen van de computer te brengen. Hierbij worden soms met het vertaalde programma andere, al kant en klare hulpprogramma's vanuit een bibliotheek meegeladen (gelinkt). Dit kunnen bijvoorbeeld rekenprogramma's zoals de logarithme, worteltrekking enz. zijn. Maar ook hulpprogramma's om karakters naar een beeldscherm te schrijven hoeven door de programmeur niet in zijn programma te worden opgenomen.
Pas nadat het programma geladen is kan het op de goede werking worden getest.
Bij het ontwikkelen van een programma voor een microprocessor zal bij het laden het objectprogramma niet naar het computergeheugen maar via een PROM-programmer in een EPROM gezet worden. Daarna wordt de EPROM in de applicatie aangebracht.

8.9 Voorbeelden van programma's voor de 80C51

Er volgen nu een aantal programma's geschreven in assembler voor de 80C51 microcontroller. Na de probleemstelling wordt meteen het programma in assembler gegeven. Dit is niet de juiste manier om zelf een probleem op te lossen. Voordat men aan het schrijven van het assemblerprogramma begint moet men eerst een oplossing maken in struktuurdiagrammen of pseudocode. Deze stap wordt hier meestal overgeslagen omdat het om de assembler gaat en niet om algemene programmeervaardigheden.
Omdat input/output nog niet behandeld is, komt dit in de programmavoorbeelden ook niet voor. In de praktijk zal een programma natuurlijk wel input en output hebben. In de voorbeelden staat in het commentaar vaak de werking van e betreffende instruktie, in werkelijkheid zal men in het commentaarveld aangeven wat het betreffende programmaonderdeel voor betekenis heeft in het totale probleem.

Voorbeeld 1: een berekening.
Berekend moet worden (P + Q - R + 1) - (S + T).
Omdat er geen input is zullen voor P,R en T variabelen met een beginwaarde worden genomen, terwijl Q en S als constanten worden gedeclareerd.

```
Q         EQU   14        ;declareer Q
S         EQU   22        ;declareer S
P         EQU   40H       ;adres P in RAM
R         EQU   41H       ;adres R in RAM
T         EQU   42H       ;adres T in RAM
RESULT    EQU   43H       ;adres RESULT in RAM
COMPUT    MOV   A,P       ;A:=P
          ADD   A,#Q      ;A:=P+Q
          CLR   C         ;C:=0
          SUBB  A,R       ;A:=P+Q-R
          INC   A         ;A:=P+Q-R+1
          XCH   A,R1      ;R1:=P+Q-R+1
          MOV   A,#S      ;A:=S
          ADD   A,T       ;A:=S+T
          XCH   A,R1      ;A:=P+Q-R+1 en R1:=S+T
          CLR   C         ;C:=0
          SUBB  A,R1      ;A:=(P+Q-R+1)-(T+S)
          MOV   RESULT,A  ;RESULT:=(P+Q-R+1)-(T+S)
          END
```

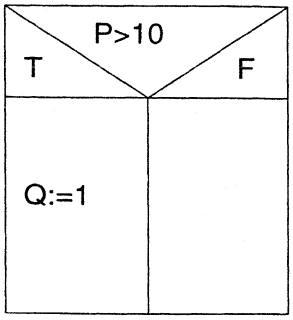
De registers A en R1 worden als hulpvariabelen gebruikt. Dit leidt tot kortere instrukties en snellere verwerking. Let op het verschil van constanten en variabelen, constanten worden met "#" ervoor gebruikt en de variabelen direct.
Als i.p.v van A en R1, R0 en R1 waren gebruikt dan was de 2^e aftrekking niet meteen mogelijk geweest omdat SUBB R0,R1 niet vertaald kan worden omdat bij de SUBB-instruktie de 1^e operand altijd de accu moet zijn.
Om de SUBB-instruktie hier goed uit te voeren moet de carry de waarde 0 hebben. Hier is dit ten overvloede gedaan door met CLR C de carry nul te maken. Dit is echter hier niet nodig omdat de voorgaande ADD-instruktie de carry ook al 0 maakt.

```
I.p.v          INC   A
kan ook        ADD   A,#1
```

De eerste vorm bestaat echter uit 1 byte, terwijl de 2e instruktie omgezet wordt in 2 bytes. De verwerkingstijd is voor beide instrukties gelijk, nl 1 cycle (= 12 klokpulsen). De eerste vorm verdient dus de voorkeur.

De nu volgende voorbeelden maken gebruik van de sprongopdrachten, al of niet voorwaardelijk. De voorbeelden laten voorwaardelijke, alternatieve en herhalingsopdrachten zien. Dit zijn konstrukties die ook in hogere talen, zoals PASCAL, voorkomen.

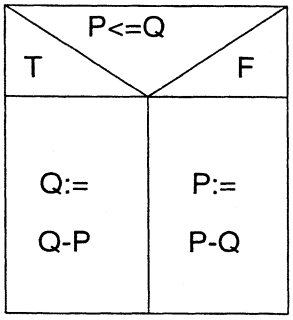
Voorwaardelijke opdracht: **IF P > 10 THEN Q:=1**




```
                ORG     0H
P               EQU     40H
Q               EQU     41H
                MOV     R2,P
                CJNE    R2,#10,ONGEL
                SJMP    ENDIF
ONGEL           JC      ENDIF
THEN            MOV     Q,#1
ENDIF           ...
                END
```

Hier wordt de CJNE instruktie gebruikt. Eerst wordt R2 vergeleken met de constante 10, en als het ongelijk is moet aan hand van de carry, die door de instruktie is beinvloed, uitgemaakt worden of het > of < is.

Alternatieve opdracht: **IF P <= Q THEN Q:=Q-P ELSE P:=P-Q**

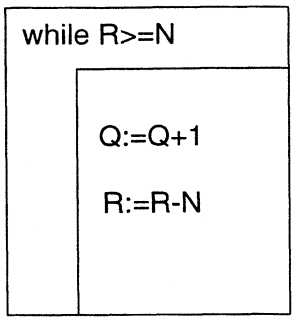



```
                ORG     0H
P               EQU     40H
Q               EQU     41H
                MOV     A,P
                CJNE    A,Q,ONGEL
                SJMP    THEN
ONGEL           JNC     ELSE
THEN            MOV     A,Q
                CLR     C
                SUBB    A,P
                MOV     Q,A
                SJMP    ENDIF
ELSE            SUBB    A,Q
                MOV     P,A
ENDIF           ...
                END
```

In de voorgaande construkties wordt een keuze gemaakt tussen twee mogelijkheden. Er komen alleen "voorwaartse" sprongen in voor. In de nu volgende drie construkties wordt een bepaald stuk programma herhaald uitgevoerd. Dat kan gebeuren onder een bepaalde conditie, of met behulp van een telling. In deze strukturen komen "terugwaartse" sprongen voor.

Herhalingsopdracht
"zolang"-versie

WHILE R >= N DO BEGIN Q:=Q+1;R:=R-N END

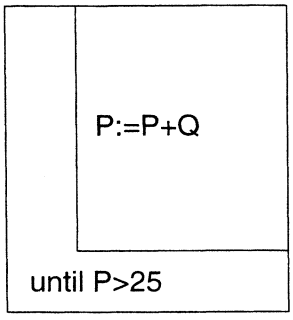



```
                ORG   0H
R               EQU   40H
Q               EQU   41H
N               EQU   42H
                MOV   A,R
WHILE           CJNE  A,N,ONGEL
                SJMP  BODY
ONGEL           JC    ENDWH
BODY            INC   Q
                SUBB  A,N
                SJMP  WHILE
ENDWH           MOV   R,A
                END
```

Bij de WHILE construktie wordt eerst getest, en dan pas al of niet de body uitgevoerd. Dit in tegenstelling tot de volgende UNTIL konstruktie, waarbij de body in elk geval één keer wordt uitgevoerd.

Herhalingsopdracht
"totdat"-versie

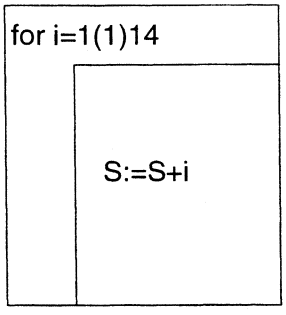
REPEAT P:=P+Q UNTIL P>25




```
                ORG   0H
P               EQU   40H
Q               EQU   41H
                MOV   A,P
REPEAT          ADD   A,Q
                CJNE  A,#25,ONGEL
                SJMP  REPEAT
ONGEL           JC    REPEAT
                MOV   P,A
                END
```

Herhalingsopdracht met teller:

FOR I:=1 TO 14 DO S:=S+I

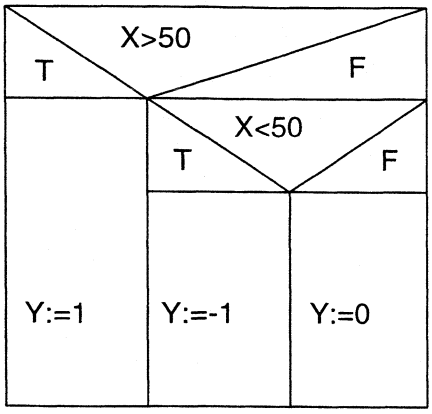



```
                ORG   0H
S               EQU   40H
                MOV   R0,#1
                MOV   A,S
FOR             ADD   A,R0
                INC   R0
                CJNE  R0,#14,ONGEL
                SJMP  FOR
ONGEL           JC    FOR
ENDFOR          MOV   S,A
                END
```

Bij de FOR-statement met constanten als grenzen vindt de test aan het einde plaats. Als de eindgrens < begingrens zal de compiler geen statements genereren. Als echter de eindgrens van een

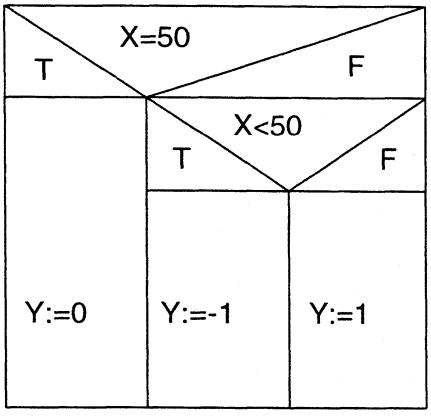
FOR-statement een berekende waarde is dan zal de test vóór de body worden uitgevoerd.
Als men alle construkties uit een hogere taal omgezet heeft in rijtjes assemblerstatements, dan zou men dus door te substitueren zelf het werk van de compiler kunnen overnemen. Maar in kombinaties kan dit tot een inefficiënte code leiden. Een goede compiler zal dan ook na het vertalen een optimalisatiestap inlassen. Deze optimalisatie is gedeeltelijk afhankelijk van de machinecode-instrukties van de doelmachine.

Voorbeeld:




```
X           EQU  40H
Y           EQU  41H
            MOV  A,X
            CJNE A,#50,ONG1
            SJMP ELSE1
ONG1        JC   ELSE1
THEN1       MOV  Y,#1
            SJMP ENDIF1
ELSE1       CJNE A,#50,ONG2
            SJMP ELSE2
ONG2        JNC  ELSE2
THEN2       MOV  Y,#-1
            SJMP ENDIF2
ELSE2       MOV  Y,#0
ENDIF2      NOP
ENDIF1      ...
```

Vanaf de label ELSE1 tot ENDIF1 is de 2e IF-statement ingevuld. Door het struktuurdiagram te veranderen, zonder dat de werking verandert, is een korter stuk programma te construeren.




```
X           EQU  40H
Y           EQU  41H
            MOV  A,X
            CJNE A,#50,ONGEL
GELIJK      MOV  Y,#0
            SJMP ENDIF
ONGEL       JNC  GROTER
KLEINER     MOV  Y,#-1
            SJMP ENDIF
GROTER      MOV  Y,#1
ENDIF       ...
```

Door het type instrukties van de 80C51 is een effectieve verdeling over de gevallen = > en < mogelijk. In deze volgorde toegepast levert dit een korter programma op dan tweemaal een verdeling over 2 gevallen te gebruiken.
Het volgende voorbeeld laat een toepassing van de MOVX-instuktie zien. Dit is de enige instruktie die werkt met het eventuele externe RAM-geheugen. Omdat het interne RAM slechts 128 bytes omvat, waar dan ook nog de stack in moet worden bijgehouden, bestaat er soms behoefte aan een extern RAM-geheugen, dat tot maximaal 64k bytes kan worden aangevuld. Het volgende stukje programmavoorbeeld maakt het mogelijk een aantal bytes (hier 32) van het interne RAM uit te wisselen met het externe RAM. Het is een voorbeeld met register-indirecte adressering. Het register

R1 wordt als 8-bits adrespointer gebruikt en het register DPTR wordt als 16-bits adrespointer gebruikt.

```
FRAME     EQU  30H            ;startadres intern RAM
PAGE      EQU  2000H          ;startadres extern RAM
          MOV  R1,#FRAME      ;init pointer intern
          MOV  DPTR,#PAGE     ;init pointer extern
          MOV  R3,20H         ;R3 is de teller
NEXTBYTE  MOVX A,@DPTR        ;haal 1 byte extern
          XCH  A,@R1          ;verwissel met intern
          MOVX @DPTR,A        ;breng byte extern
          INC  R1             ;verhoog pointer int.
          INC  DPTR           ;verhoog pointer ext.
          DJNZ R3,NEXTBYTE    ;genoeg?
          ....                ;ja het is klaar
          END
```

Als het externe RAM-geheugen programmatisch in stukjes (bladzijden) wordt verdeeld die verwisseld kunnen worden met een intern stuk geheugen (frame) dan is met behulp van wat "boekhoudkundige" routines er voor te zorgen dat het lijkt alsof men beschikt over een groot intern RAM-geheugen. Bij grotere systemen vind dit uitwisselen van RAM-geheugen plaats met DISK-geheugen. Men heeft dan te maken met een vorm van memory-management.
Het volgende voorbeeld laat een mogelijkheid zien om in het programmageheugen te lezen. Omdat bij controllers het programma in read-only memory staat, kan er wel gelezen worden, maar nooit geschreven worden. De behoefte om in het programmageheugen te lezen ontstaat als men tabellen met constanten in het programmageheugen wil opnemen. (variabelen kunnen natuurlijk niet worden opgenomen in het programmageheugen, omdat er niet geschreven kan worden). Om in programmageheugen te lezen is de instruktie MOVC te gebruiken. Stel men wil een sinustabel opnemen met 254 waarden volgens de formule:

$$n \rightarrow \sin \frac{n*\pi}{2*255} \quad met \quad (n = 1 \ tot \ 254)$$

```
          bepaling van de
          argumentwaarde n
          in de accu A.
          .....
          .....
          INC  A                ;de index wordt aangepast
          MOVC A,@A+PC          ;ophalen tabelwaarde
          AJMP OVERTAB          ;tabel zelf overslaan
TAB       DB   exp1,exp2,exp3..     ;tabelwaarden
          DB   exp9,exp10,exp11     ;      "
          .......                   ;      "
          DB   exp252,exp253,exp254 ;      "
OVERTAB   .....
          hier is de sinuswaarde
          in A beschikbaar.
```

De tabel bestaat uit 254 waarden, behorend bij n=1 tot n=254. De tabelindex is slechts te gebruiken van 2 tot 255, want de PC wijst bij uitvoering van de instruktie MOVC A,@A+PC naar de uit 2

bytes bestaande instruktie AJMP OVERTAB. De eerste twee tabelwaarden zijn dus niet te gebruiken. De tabelindex loopt van 2 tot 255. Daarom moet na de bepaling van n in A, A nog met 1 worden verhoogd om de juiste indexering in de tabel te geven.
De inhoud van de PC is de adrespointer, de inhoud van A de index, en de som van deze twee levert het adres van de tabelwaarde op.

In het volgende voorbeeld wordt indirecte registeradressering via R1 toegepast. Bovendien worden hier karakters als constanten gebruikt.
In het voorbeeld wordt gezocht in een string karakters naar de combinatie := op twee opvolgende plaatsen.
Het programma is een WHILE-construktie, immers zolang de pointer R1 nog niet naar de laatste plaats wijst en de combinatie nog niet gevonden is, dus als (R1<#STRING+LENGTE-1) AND (A=0) ga door met zoeken.
De karakters : en = staan tussen enkele quotes, en worden door de assembler door hun ASCII-waarde vervangen. Omdat het constanten zijn worden deze waarden vooraf gegaan door #,ten teken van immidiate addressing.
Als na het vinden van : het volgende teken geen = is wordt de pointer niet verhoogd, omdat diezelfde plaats nog op : getest moet worden. Anders kan de combinatie ::= niet goed verwerkt worden.
Het programma kan er als volgt uitzien:

```
RESULT    EQU  4FH                          ;variabele
STRING    EQU  50H                          ;string
LENGTE    EQU  20                           ;lengte 20
          CLR  A
          MOV  R1,#STRING-1                 ;R1 voor string
NEXTSYM   INC  R1                           ;R1 op string
          CJNE R1,#STRING+LENGTE-1,ONGEL;test op eind
          SJMP KLAAR
ONGEL     JNZ  KLAAR                        ;test op gevonden
          CJNE @R1,#':',NEXTSYM             ;gelijk : ?
          INC  R1                           ;volgende symbool
          CJNE @R1,#'=',NEXTSYM+1           ;gelijk = ?
          MOV  A,R1                         ;ja dan in A
          SUBB A,#STRING                    ;vorm rangnummer
          MOV  RESULT,A                     ;bewaar in RESULT
          DEC  RESULT                       ;corrigeer voor :
          SJMP NEXTSYM
KLAAR     ....
```

In het volgende voorbeeld wordt een binair getal dat op adres 40H of HEX staat naar het octale equivalent in ASCII vertaald. Voor de 3 cijfers van het octale getal wordt een buffer van 3 plaatsen gereserveerd vanaf adres 41H. R1 wordt als pointer in de cijferbuffer gebruikt en R2 als teller. De octale cijfers worden, van de laagwaardige kant af, gevormd door de 8 bits aangevuld met een Carry 0, in stukjes van 3 bits te knippen en de ASCII-waarde van het cijfer 0 erbij op te tellen. Omdat het knippen destruktief gebeurt wordt telkens een copie bewaard in de bufferplaats, die na bewerking verwisseld wordt. Het programma:

```
CODE0     EQU  40H              ;code voor 0
TAL       EQU  3                ;aantal cijfers
HEX       EQU  40H              ;te bewerken getal
OCT       EQU  41H              ;1e plaats
          MOV  R1,#OCT+TAL-1    ;laatste plaats
          MOV  R2,#TAL          ;init teller
          MOV  A,HEX            ;getal in accu
NEXTCYF   MOV  @R1,A            ;copie in buffer
          ANL  A,#7             ;knip 3 bits
          ADD  A,#CODE          ;maak ASCII
          XCH  A,@R1            ;zet in buffer
          RRC                   ;3 plaatsen
          RRC                   ;naar rechts
          RRC                   ;schuiven
          DEC  R1               ;pointer terug
          DJNZ R2,NEXTCYF       ;teller terug
          END
```

8.10 Het gebruik van subroutines

Bij het gebruik van een **subroutine** of procedure wordt een aantal instrukties bij elkaar gezet, los van het hoofdprogramma. Het hoofdprogramma is het gedeelte waarin de uitvoering of executie van het programma begint en eindigt. Om nu de apart gezette instrukties te kunnen uitvoeren moet in het hoofdprogramma een **subroutine-aanroep** instruktie, dit is een sprong naar de subroutine, worden uitgevoerd. Vervolgens worden de instrukties van de subroutine uitgevoerd. De laatste instruktie die in de subroutine wordt uitgevoerd is de **subroutine-terugkeer** of **return**-instruktie. Deze instruktie zorgt dat weer naar het hoofdprogramma wordt teruggesprongen, en wel naar de instruktie die volgt op de oorspronkelijke subroutine-aanroep instruktie.
Er zijn drie belangrijke redenen om in programma's subroutines te gebruiken:

. Een programma is eenvoudiger en beter te schrijven en te onderhouden als men het opsplitst in een aantal subroutines, die een duidelijk omschreven taak moet hebben,onafhankelijk schrijft en test. Men kan top-down ontwerpen, op de plaats van de subroutine-aanroep behoeft men slechts te weten **wat** de subroutine doet, zonder dat men zich al bekommert **hoe** dit wordt uitgevoerd. Pas bij het ontwerp van de subroutine zelf gaat men het **hoe** invullen. In de subroutine zelf kunnen weer aanroepen naar andere subroutines plaats vinden. Hierdoor wordt een stelselmatige verfijning van het te ontwerpen programma mogelijk. In het hoofdprogramma blijven de grote lijnen zichtbaar omdat het uit complexe blokken, de subroutines, en niet uit een veel groter aantal enkele instrukties is samengesteld.

. Als op een aantal plaatsen in het hoofdprogramma dezelfde reeks instrukties voorkomen bespaart het werk voor de programmeur en ruimte in het computergeheugen als deze reeks slechts éénmaal wordt toegevoegd aan het programma.

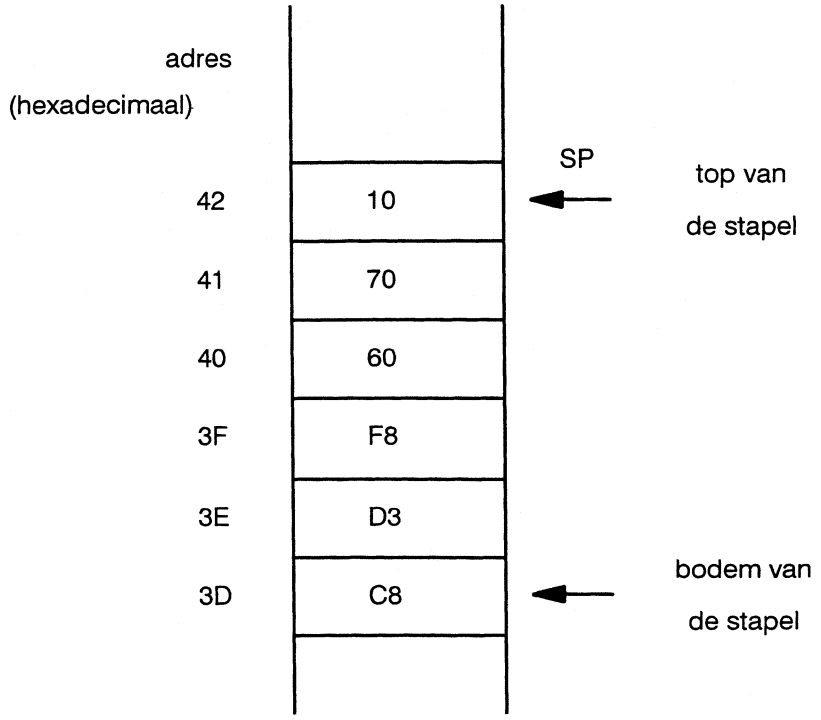
. Als een bepaald type bewerking op verschillende plaatsen in het programma voorkomt is het efficiënt om van één subroutine gebruik te maken. Om dit voor elkaar te krijgen wordt een subroutine met parameters gebruikt. Die parameters worden dan per aanroep ingevuld. Op die manier kan de programmeur meer struktuur in zijn programma's krijgen en ook een kleiner aantal typen subroutines gebruiken.

Het tweede en derde punt zijn vooral besparingen, en al bekend sinds het begin van het assemblerprogrammeren. Het eerste punt heeft pas aandacht gekregen toen het top-down ontwerpen in gebruik kwam. Het modulair programmeren is zo belangrijk dat niet alleen in assembler de subroutines worden gebruikt maar dit ook in hogere talen is ingebouwd, maar dan als procedures.

8.11 Aanroep en terugkeer van een subroutine bij de 80C51

In deze paragraaf wordt het subroutine mechanisme voor de 80C51 micro-controller beschreven. De voorbeelden zijn hierop afgestemd. De 80C51 heeft een **stackpointer** of **stapelwijzer** die is ondergebracht in het S(pecial) F(unction) R(egister) gebied. De S(tack) P(ointer) zit op adres 81H en is een register van één byte. Een **stack** of **stapel** is een gebied in het RAM geheugen voor tijdelijke opslag van gegevens, waarbij het gegeven dat er het laatst op gezet is er weer het eerst afgehaald wordt. (Last In First Out of LIFO ,in tegenstelling tot de FIFI = First In First Out organisatie die gebruikt wordt bij wachtrijen of queues). Het gegeven dat het eerste op de stapel gezet is staat op de **bodem van de stapel**, het gegeven dat er het laatst opgezet is staat op de **top van de stapel**. De Stack Pointer wijst steeds naar de top van de stapel. Bij de 80C51 groeit de stack in de richting van de hogere adressen. De bodem heeft dus het laagste adres, terwijl de top altijd een groter adres heeft.

Een stapel kan er als volgt uitzien:

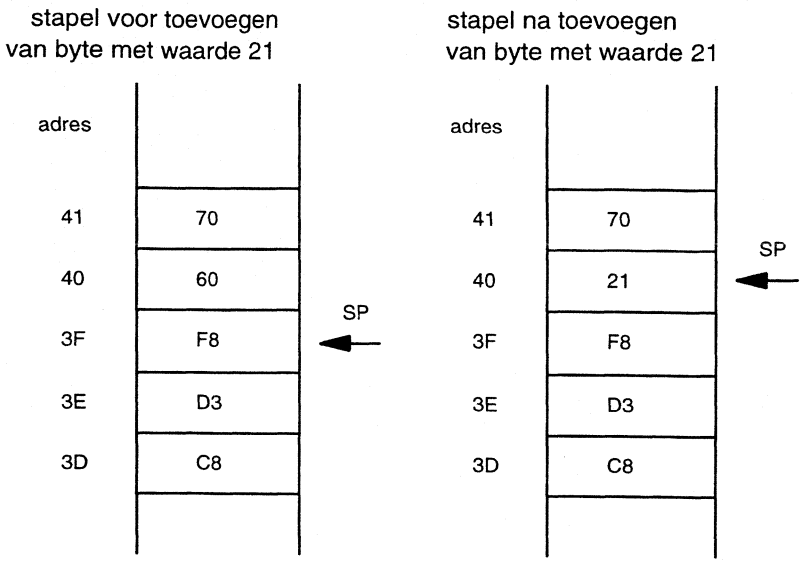


Afbeelding 110

De stapel bij de 80C51 ligt in het interne RAM, en kan dus niet verder groeien dan 7FH (bij andere typen micro-controllers van dezelfde familie kan dit 0FFH zijn.). De SP is ook maar een 8-bits register. Gegevens die op hogere adressen worden "weggezet" gaan verloren.
Een aantal van de nog volgende technieken, die gebruik maken van de stack, zijn algemeen van aard. De voorbeelden worden echter beschreven in 80C51 assemblercode, alhoewel ze daar door de kleine stack vaak niet erg realistisch zijn. De meeste (micro-)computers laten stacks tot minimaal 64K bytes toe en daar zijn de technieken zonder meer toepasbaar.

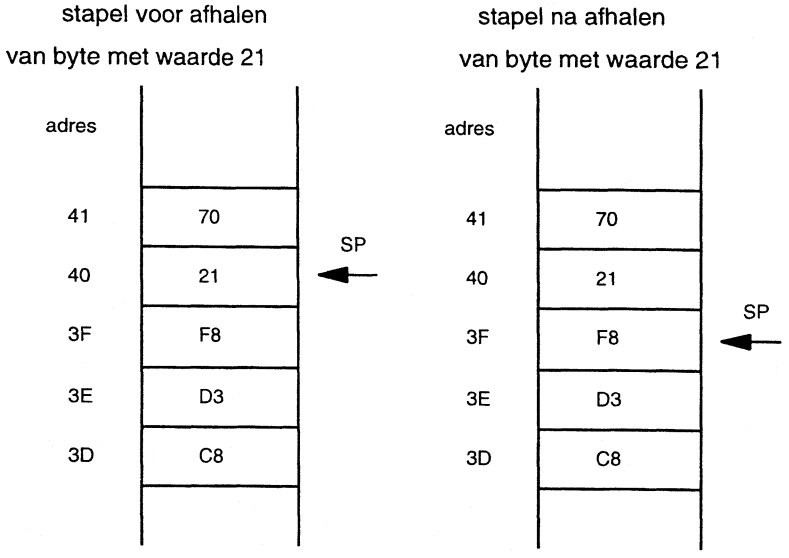
Het op de stapel zetten van een byte kan met de instruktie PUSH operand. Hierbij kan voor de operand alleen van directe adressering gebruik gemaakt worden. De inhoud van de geadresseerde plaats wordt gecopiëerd naar de top van de stack. De instruktie om een byte van de stack te halen is POP operand. Ook hier is de operand alleen direct adresseerbaar. De operand van de POP instruktie wordt van stack gehaald en in de direct geadresseerde plaats weggeschreven.

Dus als op adresplaats 25H de waarde 21H staat dan zal de instruktie
PUSH 25H het volgende beeld opleveren:



Afbeelding 111

De adressen die boven de top van de stack liggen worden dus overschreven en zijn niet "veilig" voor andere gegevensopslag.
Als de POP 25H instruktie wordt uitgevoerd zal de inhoud van de top van de stack gecopiëerd worden naar de geadresseerde plaats. Na afloop van de instruktie zal de inhoud van 25H dus 21H zijn. de stackmanipulatie kan als volgt worden geschetst:



Afbeelding 112

De sub-
routine is een stukje programma dat na uitvoering terug moet springen naar het programma op de plaats ná de instruktie waarvandaan de aanroep heeft plaatsgevonden. Dit **terugkeeradres** (de inhoud van de Program Counter op dat moment) moet dus onthouden worden. Dit wordt door de

uitvoerende instruktie gedaan door het terugkeeradres op de stack te zetten. Dit terugkeeradres bestaat uit 16 bits, en zal dus als 2 bytes, 2 plaatsen op de stack vergen. Als eerste wordt PC[0-7] en daarna PC[8-15] op stack gezet. De instrukties die deze subroutine aanroep op hun geweten kunnen hebben zijn ACALL adres en LCALL adres . Hierbij moet bij de ACALL- instruktie het adres binnen dezelfde 2K liggen als de aanroep, omdat de in de instruktie voorkomende 11 adresbits met de program-counter aangevuld worden tot 16 bits. De LCALL-instruktie heeft geen beperking want het volledige adres is in de instruktie gespecificeerd, maar het kost wel één byte meer binnen de instruktie.
De instruktie om uit de subroutine terug te keren is RET. Deze instruktie zal dus 2 bytes van stack halen en in de PC plaatsen waardoor de programma-executie weer doorgaat op het adres dat op d stack was opgeslagen.
Dit mechnisme laat toe dat binnen een subroutine opnieuw een andere of zelfs dezelfde subroutine opnieuw wordt aangeroepen. De adressen worden gewoon gestapeld, en de eerste terugkeer hoort bij de laatste aanroep. Wel moet de gebruiker binnen een subroutine ervoor zorgen dat bij toepassing van PUSH en POP instrukties de stack weer wordt opgeleverd zoals hij in gebruik genomen werd.

Een voorbeeld van het gebruik van een subroutine is het volgende. Stel dat een stukje programma er als volgt uitziet:

```
        .....
        MOV   A,82H     ;test of DPTR-low
        JNZ   NBOR      ;0 is
        DEC   83H       ;verlaag DPTR-high
NBOR    DEC   82H       ;verlaag DPTR-low
        .....
```

Dit stukje programma verlaagt het DPTR-register. Er bestaat nl wel de instruktie INC DPTR maar niet DEC DPTR. Het bovenstaande stukje heeft echter wel die werking. Komt dit nu op een aanta plaatsen in het programma voor dan kan men dit beter als de subroutine DECDPTR declareren. Dan kan het er als volgt uit gaan zien:

```
        .....
        .....
        ACALL DECDPTR   ;roep DECDPTR aan
        .....
        .....
        ACALL DECDPTR   ;roep DECDPTR aan
        .....
        .....
DECDPTR MOV   A,82H
        JNZ   NBOR
        DEC   83H
NBOR    DEC   82H
        RET             ;keer terug naar programma
```

Het rijtje instrukties is vervangen door één aanroep, terwijl het rijtje instrukties op een andere plaat in het programma, voorzien van een label en aan het eind afgesloten met een RET instruktie, worc opgenomen.
Het aantal instrukties dat wordt opgeschreven en dat in het geheugen moet worden gezet is kleine geworden. Het aantal instrukties dat moet worden uitgevoerd is echter toegenomen, immers de ACALL en RET instrukties zijn erbij gekomen.

8.12 Parameteroverdracht

In het geval van de DECDPTR subroutine wordt altijd de DPTR bewerkt, maar bijvoorbeeld een worteltrek-subroutine zal telkens op een andere waarde van een variabele worden toegepast. Op één of andere manier moet de waarde waaruit de wortel getrokken moet worden aan de subroutine worden doorgegeven. Hiervoor zijn verschillende mechanismen mogelijk. En vanzelfsprekend moet zowel in het hoofdprogramma als in de subroutine van hetzelfde mechanisme worden uitgegaan.

De eenvoudigste manier is om de parameter op een vaste plaats in het geheugen te zetten:

```
PAR      EQU          40H          ;parameterplaats
         .....
         .....
VAR      EQU          55H          ;variabele
         .....
         .....
         MOV          PAR,VAR      ;vul parameter in
         ACALL        ROUTINE      ;roep subroutine aan
         .....
         .....
ROUTINE  .....
         .....
         ANL          PAR,#1CH     ;gebruik waarde
         .....                     ;parameter
         RET
```

Een willekeurige waarde van een variabele wordt via een afgesproken parameterplaats, in het geheugen, doorgegeven aan de subroutine. De parameter fungeert als een doorgeefluik van een waarde. Bij de hogere programmeertaal PASCAL wordt zo'n parameter een value parameter genoemd. De waarde wordt bij het doorgeven gecopiëerd.
Een andere mogelijkheid is dat niet de waarde van de variabele, maar het adres van de variabele via de parameter wordt doorgegeven. Zo'n parameter wordt in PASCAL een variabele-parameter genoemd.

```
PAR      EQU          40H          ;parameterplaats
         .....
         .....
VAR      EQU          55H          ;variabele
         .....
         MOV          PAR,#VAR     ;vul parameter in
         ACALL        ROUTINE      ;roep subroutine aan
         .....
         .....
ROUTINE  .....
         .....
         MOV          R1,PAR       ;zet adres in R1
         MOV          A,@R1        ;gebruik waarde VAR
         ADD          A,#15        ;bewerk waarde
         MOV          @R1,A        ;zet antwoord in VAR
         .....
         RET
```

Hier wordt dus vanuit de subroutine de waarde van de variabele in het hoofdprogramma direct een berekende waarde gegeven. Vooral bij lijsten variabelen of array's heeft dit voordelen, i.p.v het copiëren van alle waarden kunnen de waarden direct worden ingevuld. Er hoeft alleen het beginadres van het array te worden doorgegeven. Door indexering op dit beginadres kunnen alle plaatsen worden bereikt.

In plaats van vaste plaatsen in het geheugen kunnen ook de registers als parameterplaatsen dienst doen. Bij de meeste computers zijn ze sneller toegankelijk dan het geheugen, maar men heeft er minder van. Bij de 80C51 is dit niet waar omdat de registers deel uitmaken van het kleine interne RAM. Het externe RAM is wel langzamer, maar niet geschikt voor het doorgeven van parameters, omdat het slechts de MOVX met adressering via DPTR toelaat. Deze algemene, veel gebruikte, technieken geven bij de 80C51 wat geforceerde voorbeelden. Grotere (en duurdere) controllers hebben meer adresseringsmogelijkheden, een grotere stack, en een groter geheugen.
Maar omdat op het praktikum de 80C51 controller is gekozen, worden de voorbeelden ook in de assemblertaal voor deze machine beschreven, en wordt niet een tweede assemblertaal geintroduceerd.

```
VAR1     EQU          55H          ;variabele
VAR2     EQU          56H          ;variabele
         .....
         .....
         MOV          R3,VAR1      ;vul parameter met waarde
         MOV          R1,#VAR2     ;vul parameter met adres
         ACALL        ROUTINE      ;roep subroutine aan
         .....
         .....
ROUTINE  .....
         .....
         ADD          A,R3         ;gebruik waarde
         MOV          @R1,A        ;antwoord in VAR2
         .....
         RET                       ;terugkeer hoofdprogramma
```

Hier wordt dus R3 als parameter voor een waarde gebruikt, terwijl via R1 een adres wordt doogegeven.
Zowel het doorgeven via een afgesproken adresplaats als het doorgeven via een register heeft het nadeel dat de afspraak voor elke subroutine opnieuw moet worden gemaakt. Een mogelijkheid waarbij dit niet nodig is, is het doorgeven via de stack.
Hogere programmeertalen maken vrijwel uitsluitend van dit mechanisme gebruik. Hieruit blijkt dat het een belangrijk mechanisme is waarvan behandeling niet achterwege kan blijven. Onze voorbeeldmachine, de 80C51 heeft een stack van maximaal 128 plaatsen, en de nu volgende voorbeelden zijn wel in 80C51 assembler geschreven, maar niet zo geschikt voor zo'n kleine controller.

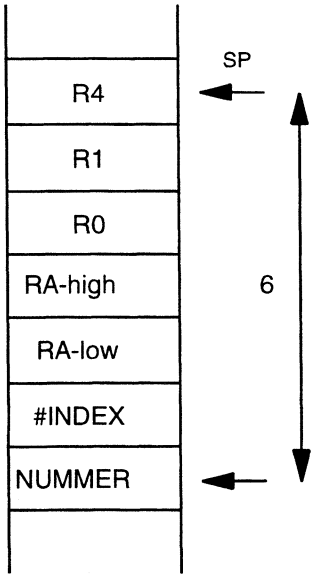
Allereerst een voorbeeld waarbij een waarde en een adres via de stack naar de subroutine worden doorgegeven:

```
VAR1     EQU        55H          ;variabele
VAR2     EQU        56H          ;variabele
ACC      EQU        0E0H         ;SFR-plaats van A
         .....
         .....
DPL      EQU        82H          :SFR-plaats DPTR-low
DPH      EQU        83H          ;SFR-plaats DPTR-high
         .....
         .....
         PUSH       VAR1         ;zet waarde VAR1 op stack
         MOV        A,#VAR2      ;maakt directe adressering
                                 ;mogelijk
         PUSH       ACC          ;zet adres op stack
         ACALL      ROUTINE      ;roep subroutine aan
         .....
         .....
ROUTINE  .....
         .....
         POP        DPH          ;haal adres-high van stack
         POP        DPL          ;haal adres-low van stack
         POP        ACC          ;haal VAR-adres van stack
         MOV        R1,A         ;en zet in R1
         POP        ACC          ;haal waarde van stack
         PUSH       DPL          ;zet adres-low en
         PUSH       DPH          ;adres-high terug op stack
         .....
         .....
         ADD        A,@R1        ;gebruik waarden
         MOV        @R1,A        ;antwoord in VAR2
         .....
         RET                     ;terugkeer hoofdprogramma
```

Achtereenvolgens wordt een waarde en een adres op de stack gezet, daarna wordt de subroutine aangeroepen waardoor het terugkeeradres op de stack gezet wordt. Het terugkeeradres neemt bij binnenkomst in de subroutine dus 2 bytes in beslag die boven op de 2 doorgegeven waarden staan. Daarom wordt eerst het adres eraf gehaald en dan worden de doorgegeven waarden van stack gehaald en in de **lokale variabelen** gezet. Daarna wordt het terugkeeradres weer op stack gezet om een "nette" RET instruktie mogelijk te maken.
Bij terugkeer uit een subroutine moet de stack er hetzelfde uitzien als voor de aanroep. Slechts als men hieraan voldoet is een ongecontroleerde groei van de stack te vermijden.
Omdat de PUSH en de POP instruktie, bij de 80C51, alleen directe adressering toelaten, wordt het adres van VAR2 eerst in A gezet, en met directe adressering via 0E0H op stack gezet. Het adres van A in het SFR gebied is 0E0H, en kan dus ook direct worden geadresseerd. Bij de meeste computers is deze omweg niet nodig, omdat de stack via allerlei adresseringen bereikbaar is.

8.13 Variabelen

Zowel in het hoofdprogramma als in subroutines kunnen variabelen gebruikt worden. Voor een variabele kan een geheugenlokatie of een register worden benut.
Men onderscheidt 2 soorten variabelen, de **globale variabelen** en de **lokale variabelen**. Als het programma in RAM staat is het verstandig de declaratie direct in het begin van het stuk programma te zetten waar ze gebruikt worden. Bij de 80C51 staat het programma in (EP)ROM en de variabelen in RAM.
De globale variabelen kunnen door het hele programma gebruikt worden, de lokale variabelen worden alleen gebruikt in de subroutine, en zijn maar tijdelijk nodig. Het is niet noodzakelijk deze tijdelijke variabelen in een register of in RAM te zetten. Men kan ook ruimte op de stack voor ze reserveren.
Om in een subroutine veilig registers te kunnen gebruiken, moet men of zeker weten dat de te gebruiken registers niet ook al in het hoofdprogramma in gebruik zijn, of men moet de waarden tijdelijk opbergen. In het nu volgende voorbeeld is dit aangegeven.
Vanuit het hoofdprogramma worden eerst NUMMER en #INDEX via de stack doorgegeven. Daarna volgt de aanroep van de subroutine. In de subroutine aangekomen worden eerst de inhouden van de te gebruiken registers R0, R1 en R4 op de stack gezet. Daardoor zijn ze weer beschikbaar om in de subroutine te gebruiken. Na deze handeling ziet de stack er als volgt uit:



Afbeelding 113

Om de parameters van stack te kunnen halen wordt een konstante NUMPAR gedefinieerd, die aangeeft hoe ver onder de top van de stack de parameters zijn opgeslagen. Bij grotere computers kunnen deze waarden, dan geindexeerd op de stackpointer, meteen bereikt worden. In het volgende voorbeeld is dit bereikt door een aparte pointer R0 te introduceren. Als voor deze pointer de SP zelf gebruikt zou worden, dan komt men in de situatie dat "boven" de stackpointer nog waarden staan die gebruikt moeten worden. Dit heeft het gevaar dat als er bijv. een interrupt komt en er gegevens, buiten het programma om, op stack gezet worden, nog te gebruiken waarden overschreven worden. Alleen waarden die "onder" de stackpointer staan zijn veilig.De ruimte die nodig is voor de tijdelijke lokale variabelen kan een voudig op stack worden gereserveerd door de SP met een bepaalde waarde te verhogen. Dit is in het voorbeeld niet aangegeven.

```
         ORG        3000H       ;beginadres
NUMMER   EQU        33H         ;variabele
INDEX    EQU        34H         ;variabele
ACC      EQU        0E0H        ;A direct
         PUSH       NUMMER      ;NUMMER naar stack
         MOV        A,#INDEX    ;adres INDEX
         PUSH       ACC         ;naar stack
         ACALL      BEWERK      ;aanroep subroutine
         .....
         .....
SP       EQU        81H         ;SP adres
DPL      EQU        82H         ;DPTR-low adres
DPH      EQU        83H         ;DPTR-high adres
NUMPAR   EQU        6           ;konstante
BEWERK   MOV        A,R0        ;breng R0 via
         PUSH       ACC         ;accu naar stack
         MOV        A,R1        ;breng R1 via
         PUSH       ACC         ;accu naar stack
         MOV        A,R4        ;breng R4 via
         PUSH       ACC         ;accu naar stack
         MOV        A,SP        ;SP voor bewerking naar A
         CLR        C           ;clear C voor aftrekken
         SUBB       A,#NUMPAR   ;vorm adres NUMMER
         MOV        R0,A        ;R0 wordt pointer
         MOV        A,@R0       ;haal NUMMER
         MOV        R4,A        ;breng in R4
         INC        R0          ;pointer op #INDEX
         MOV        A,@R0       ;haal #INDEX
         MOV        R1,A        ;breng naar R1
         .....
         bewerkingen met
         resultaat in A
         .....
         MOV        @R1,A       ;resultaat naar INDEX
         POP        ACC         ;haal R4 van stack
         MOV        R4,A        ;herstel R4
         POP        ACC         ;haal R1 van stack
         MOV        R1,A        ;herstel R1
         POP        ACC         ;haal R0 van stack
         MOV        R0,A        ;herstel R0
         POP        DPH         ;haal R.A. high
         POP        DPL         ;haal R.A. low
         DEC        SP          ;pas SP aan
         DEC        SP
         PUSH       DPL         ;R.A. terug op
         PUSH       DPH         ;juiste plaats stack
         RET                    ;terug naar programma
```

8.14 Herbetreedbaarheid

In moderne computerssystemen is het mogelijk dat een aantal programma's tegelijkertijd in het geheugen van de machine staan. De machine werkt om de beurt gedurende korte tijd aan één van de programma's zodat over een lange tijd bekeken alle programma's een deel van de beschikbare verwerkingstijd krijgen toegewezen.

De mogelijkheid voor verschillende gebruikers gezamenlijk één enkele kopie van een programma te gebruiken kan in zo'n systeem een aanzienlijke besparing van hetgeheugengebruik opleveren. Denk bijv. aaneen teksteditor in een systeem met verschillende terminals, waar vaak door verschillende mensen tegelijkertijd mee wordt gewerkt. Programma's waarbij dit mogelijk is, moeten echter wel **herbetreedbaar** (Engels :re-entrant) zijn. Herbetreedbare programma's onderscheiden zich op de volgende punten:

. De instrukties van het programma mogen tijdens de executie niet veranderd worden, dit wordt trouwens bij alle programma's ten stelligste afgeraden, omdat het opsporen van fouten er bijzonder ingewikkeld door wordt.

. De gebieden in het geheugen waar variabelen en parameters voor het programma staan,moeten duidelijk gescheiden zijn van het gebied waar de instrukties staan, en elke gebruiker van het programma moet een eigen gebied voor variabelen en parameters hebben.

Bij het omschakelen van de executie van een programma voor een bepaalde gebruiker naar de executie van een programma voor een andere gebruiker, zorgt het besturingsprogramma van de computer (het operating systeem) ervoor dat de waarde van alle registers, dus ook van de stapelwijzer, bewaard worden, en de waarden die de registers hadden op het moment dat het tweede programma de laatste keer onderbroken werd,worden teruggezet in de registers. Door iedere gebruiker een eigen stapel te geven en in een programma variabelen en parameters alleen in registers of op de stapel te plaatsen, kan voldaan worden aan de tweede eis betreffende herbetreedbaarheid van programma's.

In 8.3 is beschreven hoe parameters via de stapel kunnen worden doorgegeven. bij binnenkomst van een subroutine kan voor de benodigde lokale variabelen een gebied van vereiste grootte op de stapel worden gereserveerd, waarbij de stapelwijzer zodanig wordt aangepast dat deze naar de top van dit gebied wijst. Met indexereing t.o.v. de stapelwijzer van de vorm @constante+SP waarbij de constante het aantal bytes tussen de top van de stapel en de betreffende variabele is, kunnen de lokale variabelen worden geadresseerd. Bij het verlaten van de subroutine moet het gereserveerde gebied weer vrij gegeven worden door de stapelwijzer op de oude waarde terug te zetten. Bij de executie van programma's die in een hogere programmeertaal geschreven zijn, vindt de toewijzing van ruimte voor variabelen in de meeste gevallen op deze manier plaats. Bij de 80C51 is de stack erg klein, en is er geen adresseringsmode die geindexeerd op de SP werkt. daarom is dit soort technieken slechts moeizaam toepasbaar in de 80C51. De onbrekende adresseringsmodes moeten dan m.b.v. andere instrukties worden nagebootst.

Recursie is een speciale toepassing van herbetreedbaarheid. Bij recursie wordt in een subroutine diezelfde subroutine aangeroepen. Er is een aantal ingewikkelde problemen die met recursie eenvoudig en compact opgelost kunnen worden.

8.15 Voorbeelden van programma's met subroutines

Het volgende voorbeeld is een rekenvoorbeeld:

Bepaal 2 * (X>>3) + X + 2 * (Y>>4) + Y (met >> naar rechts schuiven)

```
        ORG     3000H       ;startadres
MONIT   EQU     00H         ;besturingssysteem
X       EQU     50H         ;variabele
Y       EQU     51H         ;variabele
SOM     EQU     52H         ;variabele
TOTAAL  EQU     52H         ;variabele
ACC     EQU     0E0H        ;invar
BREG    EQU     0F0H        ;invar
DPL     EQU     82H         ;invar
DPH     EQU     83H         ;invar
START   PUSH    X           ;breng x naar stack
        MOV     A,#3        ;breng waarde 3
        PUSH    ACC         ;naar stack
        MOV     A,#SOM      ;breng adres-
        PUSH    ACC         ;SOM naar stack
        ACALL   FORMUL      ;roep FORMUL aan
        MOV     TOTAAL,SOM  ;zet resultaat in TOTAAL
        PUSH    Y           ;breng y naar stack
        MOV     A,#4        ;breng waarde 4
        PUSH    ACC         ;naar stack
        MOV     A,#SOM      ;breng adres-
        PUSH    ACC         ;SOM naar stack
        ACALL   FORMUL      ;roep FORMUL aan
        MOV     A,TOTAAL    ;bepaal de som-
        ADD     A,SOM       ;van TOTAAL en SOM
        MOV     TOTAAL,A    ;zet in TOTAAL
        LJMP    MONIT       ;terug naar besturing
FORMUL  POP     DPH         ;haal adres-
        POP     DPL         ;van stack
        POP     ACC         ;haal adres van SOM-
        MOV     R1,A        ;van stack naar R1
        POP     ACC         ;haal waarde van stack
        MOV     R2,A        ;en breng in R2
        POP     ACC         ;haal variabele van stack-
        PUSH    DPL         ;zet het terugkeeradres-
        PUSH    DPH         ;weer op stack
SHIFT   CLR     C           ;maak Carry 0
        RRC     A           ;roteer rechts + C
        DJNZ    R2,SHIFT    ;aantal goed?
        ADD     A,ACC       ;A:=2*A
        ADD     A,R2        ;A:=A+R2
        MOV     @R1,A       ;resultaat naar SOM
        RET                 ;keer terug naar programma
        END                 ;eind voor de assembler
```

In dit voorbeeld zijn duidelijk twee stukken te onderscheiden die sterk op elkaar lijken. Door parameters in te voeren kunnen beide stukken met behulp van één subroutine worden uitgevoerd. De directief END, aan het eind van de tekst van het programma, is een aanwijzing voor de assembler dat de vertaler klaar is. Beeindiging van het programma vind plaats door de instruktie LJMP MONIT. Dit is een sprong naar het besturingsprogramma. Men kan programmatisch een programma niet echt stoppen. Men geeft het programma de vorm van een eindeloze lus, of draagt het bestuur over aan het besturingssysteem. Sommige computers kennen wel de HALT instruktie, maar vooral in systemen die door een aantal mensen tegelijk worden gebruikt is deze instruktie afgeschermd voor de "gewone" gebruiker.

Het volgende voorbeeld geeft aan hoe een subroutine gebruikt kan worden om waarden te lezen uit een tabel met constanten in het programmageheugen.

```
        MOV         A,ENTRY     ;plaats rangnummer in A
        ACALL       TABLE       ;roep de leesroutine aan
        .....
        .....
TABLE   MOVC        A,@A+PC     ;haal waarde uit tabel in A
        RET                     ;terugkeer naar programma
        tabwaarde 1
        tabwaarde 2
        tabwaarde 3
        ..........
        ..........
        tabwaarde 254
```

Hier vindt indexing op de PC plaats. Op het moment van uitvoering van de geïndexeerde adressering wijst de PC al naar de instruktie RET. Indexering met A=0 geeft geen tabelwaarde, maar het bitpatroon van de instruktie RET (= 22H). De tabelwaarden volgen direct op de subroutine, en elke tabel heeft zijn eigen subroutine-aanroep.

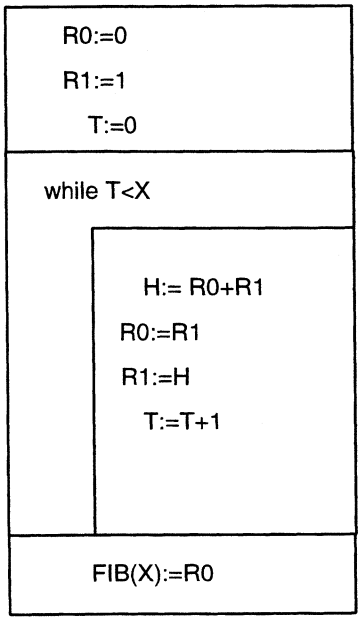
Het volgend voorbeeld is een programma met een recursieve subroutine, d.w.z dat de subroutine zichzelf, zelfs tweemaal, aanroept. Bij een recursieve construktie moet erop gelet worden dat deze aanroep conditioneel plaats vind. Zou de aanroep naar de eigen subroutine zonder test plaats vinden, dan stopt het proces nooit. (In werkelijkheid zal het programma wel stoppen, omdat de stackruimte altijd eindig is, en vroeg of laat zal het programma vast lopen).
Het voorbeeld is ontleend aan het genereren van de reeks van Fibonacci, met de regels:

```
Fib(0)=0
Fib(1)=1
Fib(x)=Fib(x-1)+Fib(x-2)        voor x > 1
```

Hierin is te zien dat elke term uit te reeks is te berekenen door de som van de twee voorgangers te nemen, die op hun beurt ook weer de som van hun voorgangers zijn,die .....
Het is een eindig proces want de argumenten worden altijd kleiner tot het moment dat x=1 of x=0, en het antwoord bekend is.

```
        ORG   3000H      ;plaats in memory
MONIT   EQU   0H         ;stuurprogramma
SP      EQU   81H        ;stackpointer
DPL     EQU   82H        ;datapointer-low
DPH     EQU   83H        ;datapointer-low
ACC     EQU   0E0H       ;accumulator
BREG    EQU   0F0H       ;B-register
ARG     EQU   48H        ;variabele arg
STACK   EQU   49H
BEGIN   MOV   SP,#STACK  ;init stack
        PUSH  ARG        ;arg op stack
        ACALL FIBON      ;bepaal Fib(arg)
        POP   ARG        ;haal antwoord
        LJMP  MONIT      ;terug naar besturing
FIBON   POP   DPH        ;haal adres op
        POP   DPL        ;en zet in DPTR
        POP   ACC        ;haal argument
        CJNE  A,#2,NEXT  ;arg > 1 ?
NEXT    JC    UIT        ;neen herstel
        PUSH  DPL        ;ja bewerk
        PUSH  DPH        ;zet adres terug
        DEC   A          ;arg-1
        PUSH  ACC        ;op stack
        DEC   A          ;arg-2
        PUSH  ACC        ;op stack
        ACALL FIBON      ;bepaal Fib(arg-2)
        POP   ACC        ;haal Fib(arg-2)
        POP   BREG       ;haal (arg-1)
        PUSH  ACC        ;zet verwisseld
        PUSH  BREG       ;terug op stack
        ACALL FIBON      ;bepaal Fib(arg-1)
        POP   ACC        ;haal Fib(arg-1)
        POP   BREG       ;haal Fib(arg-2)
        ADD   A,BREG     ;bepaal som
        POP   DPH        ;haal adres
        POP   DPL        ;van stack
UIT     PUSH  ACC        ;zet antwoord op stack
        PUSH  DPL        ;zet adres
        PUSH  DPH        ;terug op stack
        RET              ;keer terug
        END              ;aanwijzing assembler
```

In deze toepassing is het voordeel van de subroutine goed te zien. Zou men de subroutineaanroep proberen te vervangen door de instrukties van de subroutine komt men vanzelf achter dit voordeel. Een ander voordeel ziet men als de subroutine geschreven wordt. Komt men binnen de subroutine de instruktetie ACALL FIBON tegen dan bedenkt men alleen wat doet het en niet het hoe. Op dat moment is alleen de werking van belang, nl: aangeroepen met x op stack wordt teruggekeerd met FIB(x) op stack. Alleen op het hoogste level moet men toezien dat ook aan het "hoe gebeurt het" is voldaan.
Overigens moet men het voorbeeld zien als een programmeermethode. Als het uitsluitend om de termen van Fibonacci gaat kan het beter niet-recursief, maar iteratief worden opgelost volgens onderstaand PSD:



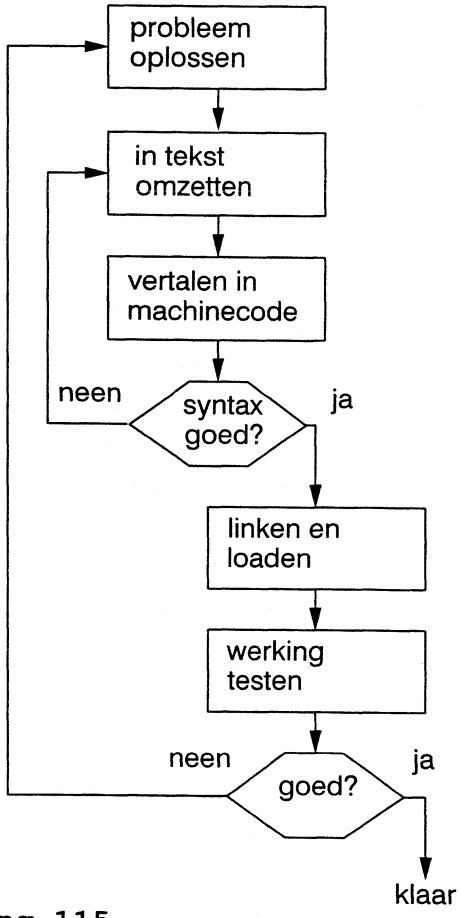
Afbeelding 114

Hier is de oplossing met een WHILE-statement beschreven. Met behulp van het voorgaande moet de lezer in staat zijn dit programma in assembler zelf te schrijven.

9 Programma ontwikkeling

9.1 Inleiding

In het hoofdstuk over Assemblerprogrammering is het werken met de assembler beschreven. Er wordt bedoeld het werken met de op het practicum gebruikte assembler. De assembler kan slechts gebruikt worden voor de 8051 familie van computers, immers de geproduceerde 8051 objectcode kan slechts door deze soort computers worden uitgevoerd. Anderzijds is het wel mogelijk dat er meerdere assemblers voor voor één type computer zijn. De werking van de verschillende instrukties worden bepaald door de archtectuur van de computer, maar de mnemonische code is naderhand bedacht. Zo zou voor de instruktie "add" ook "telop" gekozen kunnen zijn.
Ook werd er geen rekening gehouden met het maken van fouten tijdens de programmaontwikkeling. In afbeelding 115 is wèl rekening gehouden met een aantal fouten. Gebruikt men bijv. in de practicumassembler de instruktie "MOV R3,R5" dan zal de assembler u vertellen dat deze instruktie niet bestaat. Er is een ***syntaxfout*** of taalfout gemaakt. Pas als alle syntactische fouten verbeterd zijn kan er een objectmodule gevormd worden. Wordt het programma daarna voor de eerste keer "gerund", dan is het heel goed mogelijk dat de werking niet overeenstemt met de verwachtingen. Om te zien wat en waar het misgaat is het prettig om over ontwikkeltools voor het testen van een programma te beschikken. Om zo'n fout te herstellen moet op het oplossingsniveau worden ingegrepen. Voor het schrijven van assemblerprogramma's moet de programmeur naast het vermogen om zijn probleem te kunnen oplossen ook beschikken over een goede kennis van de gebruikte computer. Om aan dit laatste ongemak te ontkomen zijn programmeertalen bedacht die machine onafhankelijk zijn. Deze talen bestaan uit instrukties die gericht zijn op het beschrijven van oplossingen van problemen. Voor het vertalen of uitvoeren van zo'n programma geschreven in een machine onafhankelijke taal of H(igh) L(evel) L(anguage) taal is een meer complex vertaalprogramma nodig.



Afbeelding 115

9.2 Vertalen

Uit het voorgaande is al duidelijk geworden dat een programmeur nooit objectcode zal produceren door direct op bitniveau zijn machinecode statements op te schrijven. (In de begindagen van de computer gebeurde dit wel. Toen schreef, of verwerkte, door manipulatie met schakelaars, een programmeur zo'n 10 instrukties per dag. Zo,n stukje programma was dan wel getest en gedocumenteerd.). Hij zal of een machineafhankelijke assemblertaal of een probleemgeoriënteerde High Level Language gebruiken. Om toch de noodzakelijke objectprogramma's in machinecode te krijgen zal er een vertaling moeten plaatsvinden. Hier worden 3 vertaalmechanismen bekeken:

- de assembler
- de compiler
- de interpretator

9.2.1 Assembler

De assembler is een vertaler die één op één vertaling maakt, d.w. z. dat voor elke assemblerinstruktie één machinecodeinstruktie produceert. De bitpatronen van de instrukties in machinecode zijn door de computerontwerper bepaald. De lettercombinaties voor de assemblerinstrukties zijn zodanig gekozen dat ze makkelijk te onthouden zijn. Daarom spreekt men ook wel van mnemonische code. Een aantal assemblers kent nog de mogelijkheid om ***macro's*** te definiëren. In deze ***macro-assemblers*** kan een stuk tekst, dat op meerdere plaatsen voorkomt, apart gezet worden, van een naam worden voorzien en in het programma zelf worden vervangen door de naam van de macro. Bij de vertaling zal dan de naam van de macro worden vervangen door de tekst die bij die naam gedefinieerd is.

```
MACRO SAVE
MOV     A,R0
PUSH    ACC
MOV     A,R1
PUSH    ACC
MOV     A,R4
PUSH    ACC
ENDM
```

Afbeelding 116

```
.....          .....
.....          MOV     A,R0
.....          PUSH    ACC
SAVE           MOV     A,R1
.....          PUSH    ACC
.....          MOV     A,R4
.....          PUSH    ACC
.....          .....
```

Afbeelding 117

In afbeelding 116 is een voorbeeld gegeven van een macrodefinitie, in dit geval een stukje programma dat de registers R0, R1 en R4 op de stack zet. De tekst zelf, de body van de macro, staat tussen de regels met de woorden MACRO en ENDM. Achter het woord macro is de naam van de macro gedefinieerd. In afbeelding 117 is aangegeven hoe bij vertaling de macroinstruktie SAVE eerst wordt vervangen door de body van de gedefinieerde macro SAVE. Na deze tekstsubstitutie vindt de eigenlijke vertaling plaats. Ook bij macro's is er vaak de mogelijkheid om parameters mee te geven.

```
MACRO DISPLAY,CIJFER
MOV     A,CIJFER
MOV     DPTR,#TABEL
MOVC    A,@A+DPTR
MOV     P1,A
ENDM
```

Afbeelding 118

```
.....          .....
.....          MOV     A,#5
DISPLAY,#5     MOV     DPTR,#TABEL
.....          MOV     A,@A+DPTR
.....          MOV     P1,A
.....          .....
```

Afbeelding 119

In afbeelding 118 is een macro gedefinieerd die het 7-segmentspatroon van een cijfer zichtbaar moet maken op de display die met poort P1 verbonden is. De naam van de macro is DISPLAY. He cijferpatroon, CIJFER, is als parameter meegegeven. Ook hier betreft het weer een eenvoudig substitutiemechanisme. Bij expansie van de macro wordt op alle plaatsen binnen de body waar de parameter voorkomt, deze vervangen door de tekst van de aktuele parameter. In afbeelding 119 is DISPLAY vervangen door de body van de gelijknamig gedefinieerde macro, terwijl de tekst #5 gesubstitueerd is op de plaats van het woord cijfer binnen de body van de macro.
Met het macromechanisme beschikt de programmeur over middelen die te vergelijken zijn met het

gebruik van subroutines. Er zijn echter verschillen, de body van een subroutine komt slechts éénmaal voor terwijl de macrobody net zo vaak voorkomt als er aanroepen zijn. De subroutine vergt extra aanroep- en terugkeerinstrukties, de macro wordt net zo vaak geëxpandeerd als er aanroepen zijn. In de objectcode zal de subroutine nog steeds zichtbaar zijn terwijl de macro verdwenen is. De macro zal dan ook meer geheugenruimte vergen dan de subroutine, maar gebruikt minder executietijd dan de subroutine. (er behoeven geen extra sprong en terugkeerinstrukties te worden uitgevoerd.)
De meeste assemblers zijn two-pass assemblers, d.w.z dat de vertaling in twee fasen plaatsvindt, en dat in elke fase de orginele tekst wordt doorlopen. In de eerste fase wordt uitgevoerd:

- scanning van de invoertekst, d.w.z. er vindt een lexicografische en syntactische analyse plaats en het invullen van de macro's
- bepalen van instruktielengte en -formaat
- invullen van de constanten
- opbouwen van de naamlijst
- vastleggen van de fouten

en in pass 2:

- het invullen van adressen voor namen
- het produceren van de objectcode
- het verzorgen van de listing

Zo worden er twee files op het achtergrondgeheugen geproduceerd:
De objectcodefile, een niet leesbare file die door de loader gebruikt wordt om het programma in het werkgeheugen te laden
De listing, een tekstfile, die eventueel uitgeprint kan worden, en die hexadecimaal de objectcode weergeeft met daarnaast de programmatekst, waaruit de vertaling is ontstaan.

Als de assembler draait op een andere computer dan het type waarvoor hij objectcode produceert spreekt men van een cross-assembler. Een voorbeeld hiervan is een assembler die op een PC (80486 of 80586 computer) draait en code voor de 8051 aanmaakt.

9.2.2 Compiler

Assemblers hebben het nadeel dat ze processorafhankelijk zijn. De programmeur moet een redelijke kennis hebben van de computer waarvoor hij programma's schrijft. Moet het programma ook op een ander type processor kunnen werken, dan zal hij een heel nieuw programma moeten schrijven. D.w.z. hij zal eerst een assembler behorend bij die andere processor moeten leren gebruiken.
De assembler laat alleen instrukties toe die direct afgebeeld kunnen worden op de mogelijkheden die de hardware van de bijbehorende processor biedt. Om een probleemoplossing te vertalen naar deze vaak simpele hardwarebewerkingen is vrij lastig. Als bijvoorbeeld rekenkundige bewerkingen moeten worden uitgevoerd is het onplezierig dat de assembler slechts instrukties voor getallen met een klein aantal (8 of 16) bits toelaat of geen deling in zijn instruktieset beschikbaar heeft.
Ook conversie naar decimale floatingpoint getallen zullen zelf moeten worden opgelost.
Bij het gebruik van datastrukturen zal zelf een afbeelding op bitpatronen moeten worden bedacht.
Hoe de verschillende registers van de processor bij deze taken moeten worden ingeschakeld komt ook voor de verantwoordelijkheid van de programmeur.

Een hogere programmeertaal ondervangt een groot aantal van deze bezwaren. Een gegeven probleem kan op een hoger abstractie niveau worden opgelost. Doordat de instrukties veel krachtiger zijn kan het programma korter en overzichterlijker worden. De type instrukties zijn probleemafhankelijk, en niet processorafhankelijk.

Als voorbeeld het programma voor het berekenen van een term uit de reeks van Fibonacci. Op blz 124 is ditzelfde programma in assembler aan de orde geweest. Hier zal hetzelfde programma gegeven worden in de hogere programmeertaal SCHEME (een dialect van LISP) :

De reeks van Fibonacci is gedefinieerd als:

```
Fib(0)=0
Fib(1)=1
Fib(x)=Fib(x-1)+Fib(x-2)
```

Direct vanuit deze definitie kan het volgende programma geschreven worden:

```
(define (fibon x)
  (if (< x 2)
      x
      (+ (fibon (- x 1))
         (fibon (- x 2)))))
```

Dat de berekening van bijvoorbeeld `(fibon 7) ==> 13` mogelijk maakt.

Ook zonder kennis van de taal SCHEME is de werking te begrijpen. Het programma kan direct de definitie volgen. Als x<2 dan is het antwoord x en zoniet dan de som van 2 aanroepen van de eigen definitie. Het feit dat de taal recursie toelaat maakt het programma zo simpel. Deze hogere taal is dus probleemgeoriënteerd en niet processorafhankelijk.

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

- BASIC: (Beginners All-purpose Symbolic Instruction Code) een eenvoudige, makkelijk te leren taal met vrij veel rekenfaciliteiten. Oorspronkelijk op Dartmouth College (U.S.A.) ontwikkeld voor onderwijsdoeleinden. Veel toegepast bij microcomputersystemen. 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. Het is krachtiger dan BASIC.

- COBOL: (COmmon Business Oriented Language) een programmeertaal met (Engelse) spreektaal gerichte instrukties, vooral gericht op de administratieve toepassingen. Een nog steeds veel gebruikte taal.

- ALGOL 60: (ALGOrithmic Language) de eerste, wat syntax betreft, formeel gedefinieerde taal. De voorloper van de "moderne" imperatieve talen met een gestruktureerde controle, parameter transportmechanisme, blokstruktuur, enz.

- PASCAL: een afstammeling van ALGOL, met een door de definitie gedwongen, nette struktuur. Uitgebreide mogelijkheden voor type definitie. Populair ook als hogere taal voor microcomputers.

- ADA: een, boven PASCAL uitgaande, gedefinieerde taal met uitgebreide toepassingsmogelijkheden. Ontwikkeld als DE programmeertaal voor iedereen.

Oorspronkelijk opgezet door initiatief van het Amerikaanse Ministerie van Defensie.

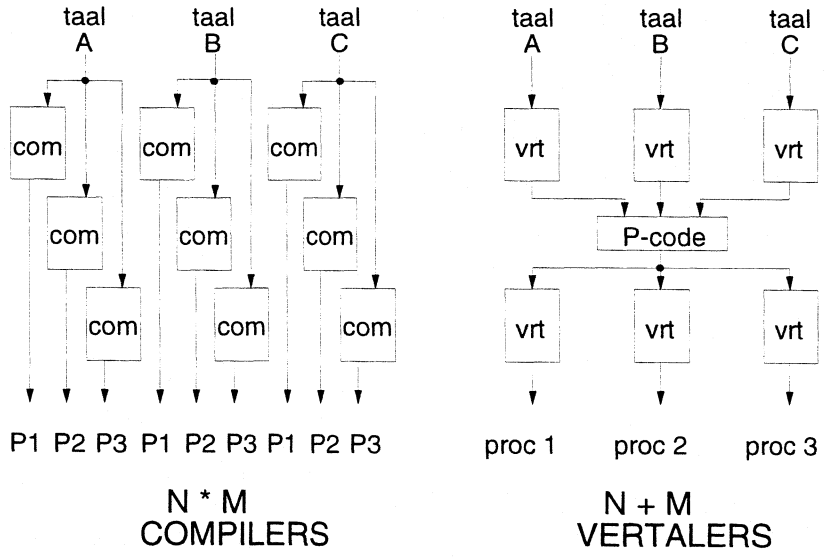
- C en C^{++} : Een taal die minder krachtig is dan PASCAL, maar een erg efficiënte code levert. Veel vertaalmechanismen evenals de C-compiler zelf, zijn geschreven in C. Omdat de taal dichter bij de machinecode staat, zijn specifieke programma's voor bijv. input/output-afhandeling, de handlers of drivers, redelijk eenvoudig te schrijven. C^{++} is C met daarbij de mogelijkheid om objectoriënted te programmeren.

- VISUAL BASIC en DELPHI: De talen BASIC en PASCAL uitgebreid met grafische objecten, leveren de talen VISUAL BASIC en DELPHI op. Zonder grote inspanning zijn hiermee programma's te maken met een grafische mens-machine interface.

- LISP: Één der eerste functionele talen veel gebruikt in het vakgebied der artificial intelligence. De bovengenoemde talen zijn alle van het imperatieve type, d.w.z. zij bestaan uit series opeenvolgende commando's. Een heel programma in LISP geschreven gedraagt zich als één functie, dat door herhaalde substitutie en evaluatie één antwoord levert. (Dit ene antwoord kan wel één lijstje getallen zijn). Het eerder gegeven voorbeeld is geschreven in SCHEME, een dialect van LISP.

- OCCAM, CONCURRENT C, enz: Talen met de mogelijkheid om systemen met meer dan één computer te programmeren. Vertaalprogramma's samen met het operating system zijn in staat om programmatuur voor een aantal (pseudo)parallel werkende processen te laten draaien. Wil de programmeur zelf echter kunnen bepalen hoe, binnen één programma, de verschillende handelingen parallel moeten worden afgehandeld, dan moet hij beschikken over een taal waarin parallellisme moet kunnen worden aangegeven.

Het vertaalmechanisme dat bij deze H(igh) L(evel) L(anguages) talen hoort is veel complexer dan een assembler. Deze vertalers noemt men ***compilers*** . Uiteindelijk zal ook een compiler weer machine-afhankelijke objectcode moeten genereren. Om te ontkomen aan het probleem dat toch weer voor elk type computer voor elke HLL een complexe compiler moet worden ontwikkeld, heeft men in het vertaalmechanisme een tussentaal gedefiniëerd.



Afbeelding 120

In afbeelding 120 is de vertaalstruktuur met en zonder p_code als tussencode aangegeven. Om drie compilers voor drie verschillende type computers te kunnen gebruiken zal men, zonder tussencode, over 3 * 3 = 9 compilers moeten beschikken. Is er wel een tussencode gedefinieerd, dan zijn er 3 vertalers nodig om de drie talen naar P-code te kunnen vertalen en verder nog 3 vertalers om van P-code de gewenste objectcode te genereren. De P-code is een soort machinecode voor een hypothetische machine. Deze hypothetische machine heeft een stack-architectuur, een registerloze architectuur die op blz. 66 genoemd is. Na de ontwikkeling van een nieuwe computer is alleen één P-code vertaler nodig om de beschikking te hebben over verschillende compilers.
De bovengenoemde hypothetische P-code machine is als "echte" machine, onder de naam "PASCAL Microengine" door Western Digital op de markt gebracht. Deze Microengine heeft als machinecode de P-code, en kan deze P-code dus direct executeren.

Het vertalen van een programma wordt door een compiler in een aantal stappen uitgevoerd:

- De lexicografische analyse. In deze fase worden de letters één voor één ingelezen, en gegroepeerd tot "tokens". Deze tokens kunnen de begrippen uit de taal, of namen die door de programmeur verzonnen zijn.

- De syntactische analyse. In deze fase wordt gekeken of de groepering van de tokens overeenkomt met de grammatica (syntax) van de taal.

- Generatie van tussencode. Dit kan bijvoorbeeld de P-code zijn. In zo'n code komen alleen nog elementaire rekenkundige bewerkingen, enkelvoudige tests en spronginstrukties voor.

- Code optimalisatie. Voorbeelden van optimalisatie in deze fase zijn:
  Het elimineren van spronginstrukties door testen om te draaien en daardoor de volgorde te veranderen.
  Rekenkundige bewerkingen, die in een veelvuldig te doorlopen loop staan, maar daarbinnen niet veranderen, buiten die loop te brengen.

- Code generatie. In deze fase wordt de tussencode omgezet in de machinecode van de targetcomputer. Ook hier kan een optimalisatie voorkomen door een efficiënt gebruik te maken van de registers en de instruktieset van de doelmachine.

In het bovenstaande wordt gesproken over optimalisatie van de code, eigenlijk zou dit verbetering moeten zijn. Ondanks het feit dat goede compilers, door "optimalisatie", een factor twee in tijd en geheugengebruik kunnen bereiken, zal het optimum voor willekeurige programma's niet gehaald worden. Dit kan theoretisch wel als in assembler geprogrammeerd wordt.

Als de compiler kan runnen op een ander type computer dan die waarvoor de objectcode wordt gegenereerd, spreekt men van een ***cross-compiler***.

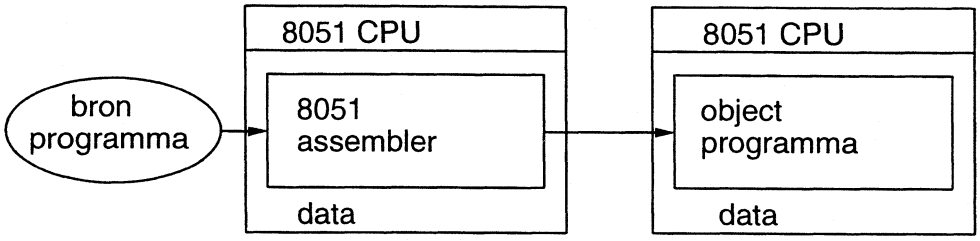
9.2.3 Interpretator

Naast compilers kennen we nog een andere vorm om programma's in een hogere taal geschreven, te kunnen uitvoeren. Dit is de interpretator. Een interpretator is een programma (of hardware) dat achtereenvolgens de statements van een programma uitvoert, door door een computer na te bootsen die deze statements meteen kan verwerken.
Computer + interpretatieprogramma vormen een "nieuwe computer" die de statemens van de hogere taal meteen kan verwerken. Als de interpretator niet alleen de verwerking van de instrukties nabootst, maar ook alle onderdelen van "nieuwe computer" dan spreekt men van simulatie.
Als voor het nabootsen van een "nieuwe computer" geen software, maar besturings-hardware wordt gebruikt, spreekt men van emulatie. Door de besturende hardware van een computer, het microprogramma, te veranderen, kan een machine worden nagebootst met een heel andere instruktieset. Dit nieuwe besturingssysteem is een hardware-interpretator.

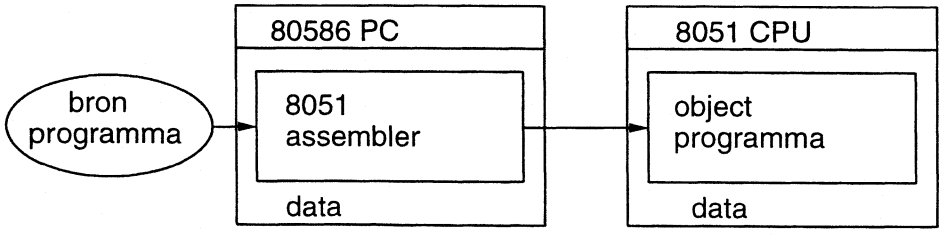
Een voordeel van een interpretator is dat na verandering van een programma direct opnieuw gerund kan worden. Dit bekort het foutzoeken aanmerkelijk. Een nadeel is dat naast het uit te voeren programma ook de interpretator, meestal een programma, aanwezig moet zijn. Interpretatie vergt ook meer handelingen, en dus meer tijd. Als een stukje programmatuur bijvoorbeeld 100 maal moet worden uitgevoerd, dan zal een compiler deze programmatuur éénmaal vertalen, de interpretator zal hetzelfde stuk programma 100 maal "vertalen".

9.3 Voorbeelden van executie

In de nu volgende tekeningen is aangegeven hoe de uitvoering van een programma kan plaatsvinden. Van elke fase van verwerking wordt aangegeven in welk type computer zij plaatsvindt, en wat er dan in het geheugen van de betreffende computer staat.
In afbeelding 121 is de rechtstreekse toepassing van een 8051 assembler gegeven. Verondersteld wordt dat de assembler op hetzelfde type computer draait als het type van de applicatie. De 8051 assembler is dus ook een 8051-objectcode programma.
In afbeelding 122 is de situatie geschetst dat de ontwikkelmachine een PC, een algemeen toepasbaar computersysteem, is. De benodigde cross-assembler is zelf een 80586-objectcode programma, maar levert 8051 objectcode af. Tussen de laatste fase van afbeelding 121 en afbeelding 122 is dus totaal geen verschil.



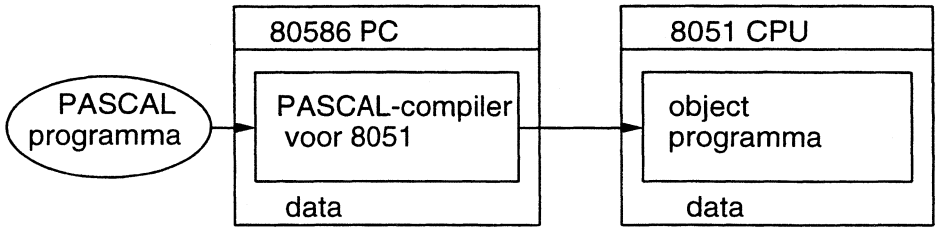
Afbeelding 121



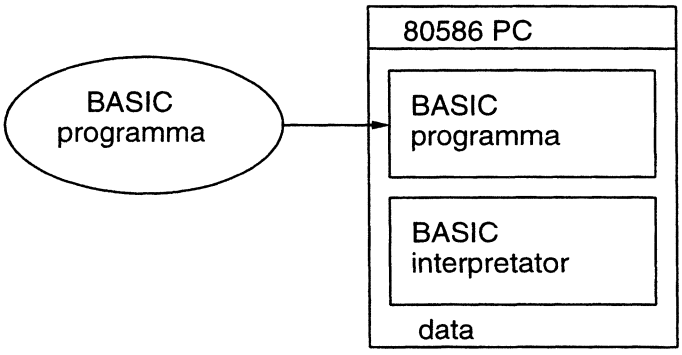
Afbeelding 122

Afbeelding 122 geeft de practicumsituatie weer. Voor programmaontwikkeling staat een PC met 8051-assembler ter beschikking, het 8051-systeem moet zelf in elkaar gesoldeerd worden.
De 8051-assembler zelf is ooit geschreven in PASCAL.

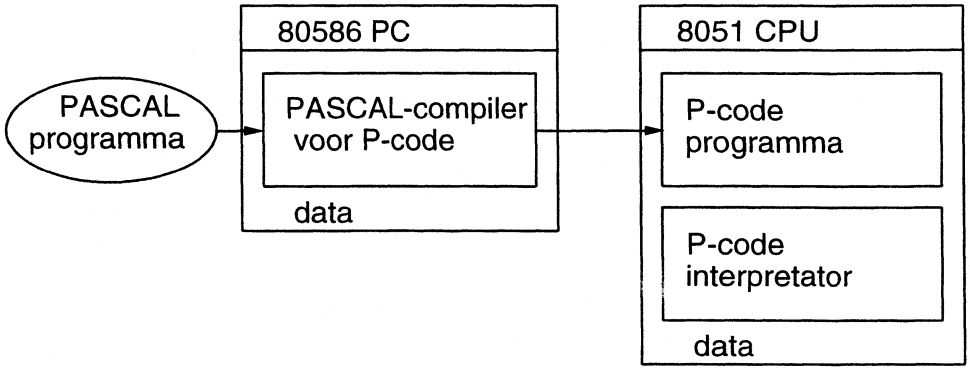
Afbeelding 123 geeft aan hoe met behulp van een cross-compiler, door een PC een PASCAL programma kan worden vertaald in een 8051-object programma, dat in de applicatie gerund kan worden.
In afbeelding 124 de meest voorkomende situatie bij het gebruik van de hogere taal BASIC weergegeven. Alhoewel BASIC ook door een compiler vertaald kan worden, wordt het meestal verwerkt door een interpretator. De machine waarop het programma gerund wordt (in dit geval een PC), moet naast het BASIC programma ook over de BASIC interpretator beschikken. Als de hogere taal interpretator veel groter is dan het applicatie programma, stelt dit onevenredig grote eisen aan grootte en snelheid van de applicatie.
Afbeelding 125 is een combinatie van 123 en 124. De PASCAL compiler levert nu P-code af. Deze machine onafhankelijke, maar redelijk eenvoudige code, wordt vervolgens in de applicatie door een P-code interpretator uitgevoerd. Stapt de gebruiker over op een ander type machine voor zijn applicatie, dan behoeft alleen een andere P-code interpretator te worden aangeschaft of gemaakt.



Afbeelding 123

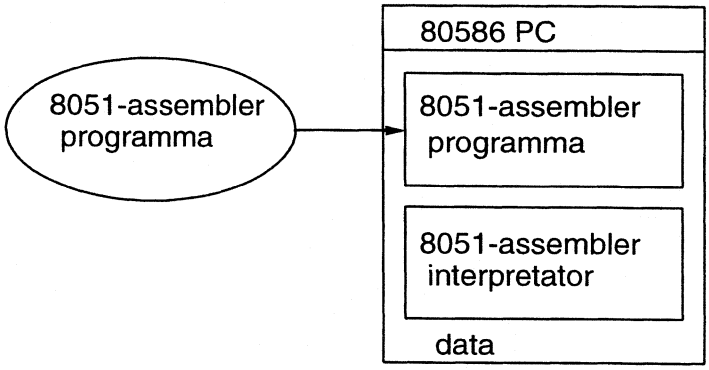


Afbeelding 124



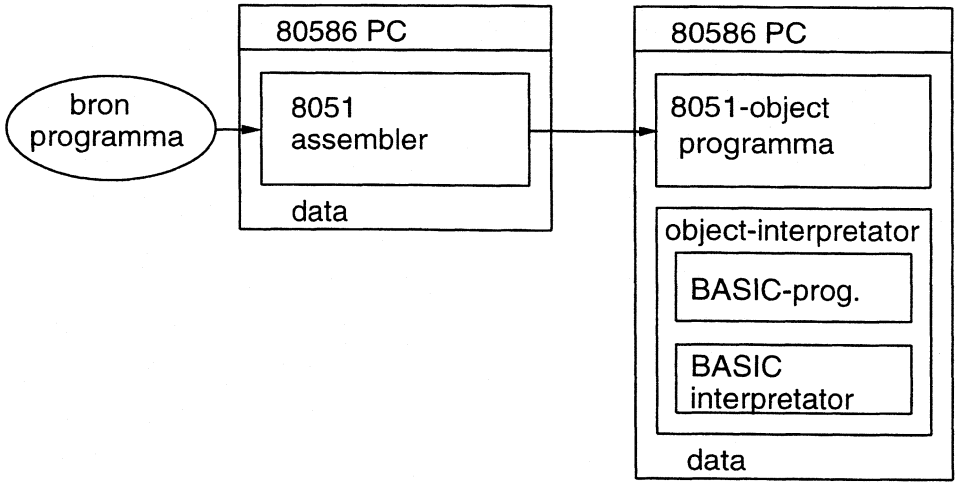
Afbeelding 125

Niet alleen hogere talen kunnen worden geïnterpreteerd. Ook programma's geschreven in assemblertaal, of zelfs programma's die in objectcode vertaald zijn kunnen door een interpretator worden verwerkt. In afbeelding 126 is aangegeven hoe een 8051 assemblerprogramma op een PC kan worden geïnterpreteerd. Door het interpretator programma uit te breiden met wat extra mogelijkheden om op bepaalde condities het programma te kunnen stoppen, om naar interne waarden van de geïnterpreteerde registers en adressen te kijken, heeft men de mogelijkheid om een assemblerprogramma uit te testen voor het in de applicatie gebracht wordt.In afbeelding 127 is een interpretator voor object- of machinecode geschetst. Hier is de mogelijkheid aanwezig om, met een uitbreiding van de object-interpretator, een reeds vertaald assembler programma, dus de programmatuur van de applicatie, te testen.



Afbeelding 126

In voorbeeld 127 is de object-interpretator zelf geschreven in BASIC, die op zijn beurt weer geïnterpreteerd wordt door een BASIC interpretator. De aanpassingen die voor testdoeleinden in de object-interpretator moeten worden aangebracht, kunnen plaats vinden door het BASIC programma te wijzigen. Omdat voor de uitvoering van 1 machine instruktie 2 interpretatie niveau's moeten worden uitgevoerd zal deze interpretatie relatief veel tijd kosten. Het voordeel zit in het makkelijk kunnen wijzigen van de objectcode-interpretator.



Afbeelding 127

9.4 Editor

De editor is een programma waarmee het mogelijk is om met teksten te manipuleren. Bestaande teksten kunnen worden gewijzigd en nieuwe teksten kunnen worden ingetypt. De bedoeling van een editor is om een tekstfile op het achtergrondgeheugen te produceren door of een bestaande tekstfile te veranderen of een nieuwe file aan te maken. De editor is veel eenvoudiger dan de gemiddelde tekstverwerker, er hoeft alleen "platte ASCII-tekst", die als invoer voor een vertaler kan dienen, te worden geleverd. De enige "lay-out" behoefte is een nette listing. Zo'n programma dient dus wel over printmogelijkheden te beschikken.

9.5 Foutzoeken

Door het gebruik van assemblertalen en hogere programmeertalen worden heel wat fouten al gedetecteerd in de vertaalfase door de assembler en de compiler. Dit soort fouten zijn syntaxfouten, de fouten die voortvloeien uit een verkeerd taalgebruik. Er is echter ook een aantal fouten dat niet in de vertaalfase wordt gevonden. Het toepassen van een verkeerd algoritme, of het onjuist coderen van een op zich goed algoritme door fouten in bijvoorbeeld de volgorde, verkeerde grenzen, foute constanten, verkeerde startvoorwaarden, dubbelgebruik van registers of variabelen enz. , kan een programma opleveren waarvan de syntax correct is, maar waarvan de werking anders is dan men verwacht had. Deze fouten worden dus pas ondekt bij het runnen van het programma. Om achter deze fouten te komen is het prettig te kunnen beschikken over debuggings (= ontluizings) faciliteiten. Deze testfaciliteiten kunnen bestaan uit programmatuur of apparatuur. De in programmatuur uitgevoerde testers worden kortweg ***debuggers*** genoemd. De met hardware uitgevoerde testers zijn de ***ontwikkelsystemen*** .

9.5.1 Debuggers

Een debugger is een programma dat de volgende faciliteiten biedt:

- het (stapsgewijs) volgen van het programmaverloop (tracing)
- het plaatsen van breekpunten (breakpoints), plaatsen waar het programma voor onderzoek tijdelijk gestopt wordt
- het uitlezen van registers, geheugenplaatsen enz.
- het (tijdelijk) veranderen van de inhouden van registers, geheugenplaatsen enz. (patching)

Bij het "tracen" van een programma zorgt de debugger ervoor dat het gebruikersprogramma stap voor stap wordt uitgevoerd. Na elke instruktie kan de waarde van de registers, de PC of de geheugenplaatsen worden opgevraagd en op een beeldscherm of printer worden zichtbaar gemaakt. Op die manier is te achterhalen op welk punt van het programma de afwijking van het gewenste resultaat optreedt. Heeft men een idee van wat er fout gaat, dan kan men op de "verdachte" plaatsen in het programma "breekpunten" aanbrengen, en alleen het programma op die plaatsen voor onderzoek te laten stoppen. Zo'n breekpunt kan gerelateerd zijn aan een adres (label) of aan een conditie die in het systeem optreedt. Voorbeelden van condities zijn bijvoorbeeld " als er op geheugenplaats 1305 geschreven wordt" of "als het link-bit geset wordt" enz. Om ook breekpunten binnen veelvuldig doorlopen programmalussen zinvol te kunnen gebruiken is vaak nog op te geven dat niet gestopt wordt bij het voor de eerste maal bereiken van het breekpunt, maar bij het voor de n^e maal bereiken van het breekpunt, waarbij n voor elk breekpunt te kiezen is. Als het programma bij het bereiken van een breekpunt gestopt wordt, kan de gebruiker alle inhouden van plaatsen binnen het systeem bekijken, of zelfs (tijdelijk) veranderen. Na het ondekken van de fout moet zo'n tijdelijke verandering als permanente verandering in de tekst van het programma worden aangebracht.

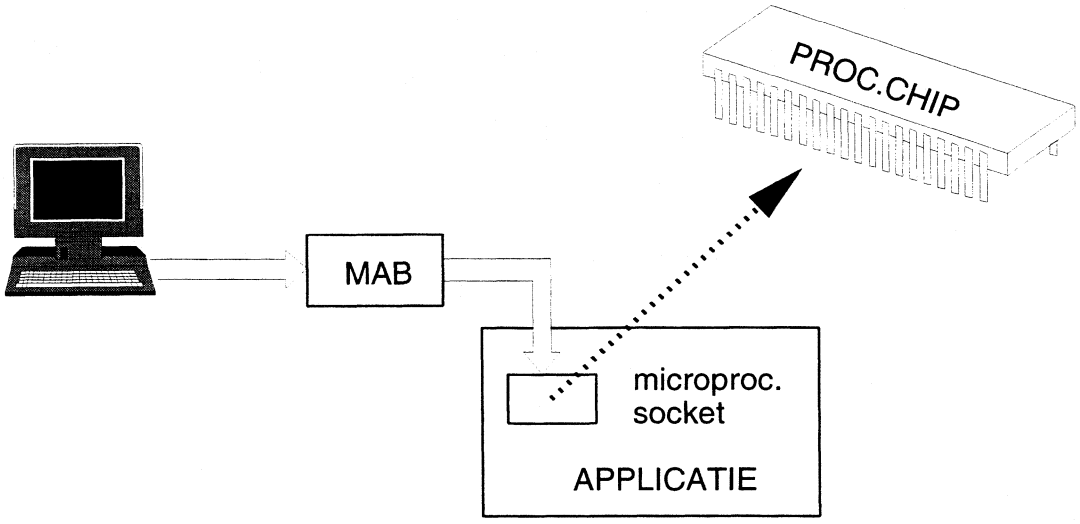
Het is duidelijk dat een interpretator op objectcode-niveau, die is uitgevoerd als simulator, (dus alle hardware zoals registers en geheugen worden exact nagebootst) en is uitgebreid met de mogelijkheid de simulatie te stoppen en het opvragen van de gesimuleerde registers en geheugenplaatsen enz., als debugger gebruikt kan worden.
Fouten die met de debugger moeilijk zijn op te sporen zijn timingsfouten. Als bijvoorbeeld een fout optreedt doordat een stuk programmatuur te traag is om het gewenste resultaat voor een bepaalde interrupt te leveren moet het programma worden getest terwijl dit op volle snelheid doorlopen wordt. In dat geval kan er van een ontwikkelsyteem gebruik gemaakt worden, of men moet van testroutines gebruik maken. Deze testroutines doorlopen de tijdkritische programma onderdelen, vaak repeterend, met aangepaste en herkenbare te verwerken data.

9.5.2 Ontwikkelsystemen

Alle ondersteuning die door software geleverd kan worden bij de ontwikkeling van programmatuur vor microprocessors is uitermate welkom. Een aantal mogelijkheden hiervan is genoemd in de vorige paragrafen. Ondanks dat moeten we ons realiseren dat de toepassingen van microprocessors een andere categorie vormen dan het ggebruiken van een computer of het toepassen van digitale electronica. Bij een computer mogen we ervan uitgaan dat de hardware werkt, en alleen de programmatuur kan problemen geven. Bij de digitale electronica wordt het schema ontworpen, en daarna moet het systeem gebouwd worden. 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 electronica nodig is. Als een éénmaal gerealiseerde microprocessortoepassing voor de eerste keer wordt ingeschakeld zal deze meestal niet werken, in ieder geval niet zoals de bedoeling was. Waar moet dan de fout gezocht worden, in de electronica (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 zgn. "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 de microprocessor eerder in zijn producten toepassen en dus meer hardware afnemen. Een MDS is een computersysteem, voorzien van programmatuur en van hardware aanpassingen, allen gericht op ondersteuning aan de gebruiker bij ontwikkeling van zijn applicatie. De programmatuur bestaat tenminste uit een editor en een assembler (evt. compiler). Het grote verschil met een gewoon computersysteem zit in de debufaciliteiten. 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 chip*** gaan optreden. Een dergelijke werkwijze wordt algemeen aangeduid met ***in-circuit-emulation*** of ICE.

In afbeelding 128 is de configuratie aangegeven.

De aansluiting van een MDS aan de applicatie gebeurt via een aanpassingscircuit, hier MAB (microprocessor adaption box) genoemd. Hierin zit o.a. een microprocessor van hetzelde type als in de applicatie was toegepast en een hoeveelheid snelle electronica.



Afbeelding 128

Omdat de MDS als CPU voor de applicatie dienst doet, hebben we, als gebruiker van de MDS:

- toegang tot alle interne registers in de CPU
- controle over alle signalen die de CPU in- 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 in een aantal stappen op te delen. De eerste stap is de programmaontwikkeling. Men schrijft een programma, m.b.v. de editor. Deze eerste versie kan summier worden gecontroleerd door, na de 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 instrukties uitvoert en in- en uitvoerinstrukties simuleert door akties 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. Parallele hieraan kan ook al een eerste test op het funktioneren 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 nagegaan of het systeem daar juist op reageert en aan de inputpoorten aangeboden waarden kunnen in de MDS worden ingelezen en gecontroleerd. Een volgende stap ken zijn dat het programma in de applicatie wordt ondergebracht, via een (E)PROM of direct vanuit de MDSvia de MAB in een RAM-geheugen (als de applicatie daarmee is uitgerust). Onder besturing van de MDS kan het programma worden uitgevoerd, waarbij de MDS de microprocessor en evt. 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 de 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 gedeelte in de MDS te situeren en een ander gedeelte 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 (vgl.breekpunten bij debuggen). In de MDS kan een geheugen aangebracht zijn, het "trace" geheugen, waarin de signaalwaarden van de diverse lijnen worden opgeslagen. Doet zich nu een conditie voor, als gespecificeerd, dan kan het lopend 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 mnemonische voorstelling van de instrukties, wat stukken simpeler voor de gebruiker is te interpreteren.

Als de gebruiker een MDS aanschaft van dezelfde firma als waar hij zijn circuits van betrekt, 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 te gaan toepassen, zal tegen deze beperking de nodige bezwaren kunnen opperen. Bepaalde fabrikanten onderkenden dit bezwaar en brengen universele MDSen uit, een ontwikkelsysteem dat ondersteuning levert voor een heel scala van microprocessors. 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 o.a. Philips, Hewlett-Pakard en Tektronix.

9.6 Assemblertaal of HLL

De keuze van de programmeertaal hangt van een aantal factoren af. Vooral de keuze tussen assemblertaal en H(igh) L(evel) L(anguage) is een belangrijke beslissing. In principe zal een hogere programmeertaal de voorkeur verdienen door de volgende voordelen:

- Het programma is processoronafhankelijk. Een éénmaal ontwikkeld programma kan na vertaling door de juiste compiler, op elk type computer draaien. Het programma hoeft niet geheel herschreven te worden.

- Door de overzichtelijke struktuur van een HLL, en de grote "bouwelementen" waaruit een HLL bestaat kan een programma in korte tijd geschreven worden.

- De programmeur behoeft geen grondige kennis van de te gebruiken computer te hebben

- Om de programmatuur te testen is een programma door de overzichtelijke struktuur eenvoudig in kleinere delen op te splitsen.

Toch zijn er ook voordelen te noemen van het gebruik van de assemblertaal:

- Een beter inzicht in de verschillende tijden die de computer, voor de uitvoering van bepaalde programmaonderdelen, nodig heeft.

- Een goed overzicht over de processing van I/O gebonden signalen. De I/O instukties samen met een combinatie van bit-test en -manipuleer instrukties zijn in assembler eenvoudig te programmeren.

- De mogelijkheid, om met veel gepuzzel, de hoeveelheid objectcode te verkleinen.
  Dit kan nodig zijn om een programma te laten passen in bepaalde afgemeten hoeveelheid geheugen.

Het laatste punt wordt steeds minder belangrijk als geheugens kleiner en goedkoper worden.
Het volgende is hier een voorbeeld van.
De creditcard-maatschappij Visa is bezig (1997) met een chip voor de betaalkaarten te onwikkelen waarop een "Java Virtual Machine" is aangebracht, samen met een besturingssyseem en een aantal bibliotheken met gebruikersroutines. Dat wil zeggen dat firma's die zo'n kaart voor bepaalde doeleinden willen gaan toepassen, hun eisen kunnen formuleren in de moderne HLL JAVA. Na toevoeging van dit JAVA programma aan de al op de chip aanwezige software is het systeem voor gebruik gereed. En dit alles voor een prijs die overeenkomt met de prijs van een chip-kaart.

10 Linking en Loading

10.1 Inleiding

Het draaien van een computerprogramma bestaat uit twee stadia. Eerst moet het bronprogramma, geschreven in assemblercode of een hogere taal, door een assembler of compiler vertaald worden in een objectprogramma, d.w.z. een programma in machinecode. Daarna moet het objectprogramma in het hoofdgeheugen geladen en vervolgens uitgevoerd worden.

Als het objectprogramma bij het assembleren of compileren op het achtergrondgeheugen, b.v. een schijf is gezet, is er een speciaal programma nodig om het objectprogramma in het hoofdgeheugen te laden. zo'n programma wordt een ***loader*** genoemd. De volgende types loaders worden behandeld:

- de absolute loader;
- de relocating loader;
- de linking loader;

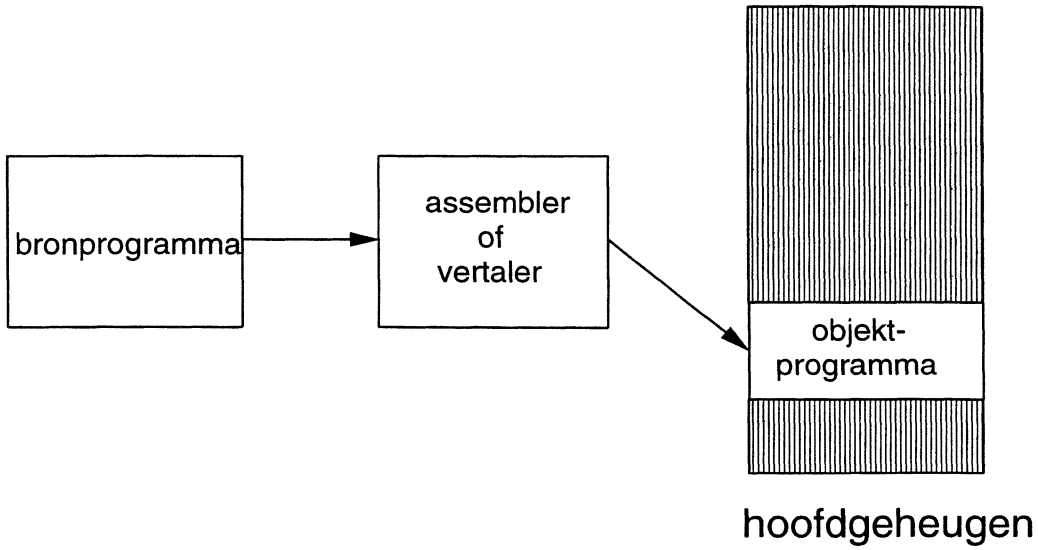
waarbij de geboden mogelijkheden, en daardoor de complexiteit van de loader, steeds verder toenemen.

Een bronprogramma bestaat vaak uit een aantal onafhankelijk van elkaar geschreven gedeelten. Zo'n gedeelte wordt een bronmodule genoemd en kan het hoofdprogramma, één of meer subroutines en/of declaraties van constantes en variabelen bevatten. verschillende bronmodulen kunnen in verschillende programmeertalen geschreven zijn. De bronmodulen worden onafhankelijk van elkaar geassembleerd of gecompileerd. Hierbij wordt een bronmodule omgezet in een objectmodule, die op het achtergrondgeheugen wordt bewaard. Als een bepaalde module gewijzigd moet worden, hoeft alleen de betreffende bronmodule opnieuw geassembleerd of gecompileerd te worden. Voordat een bronprogramma kan worden geladen en uitgevoerd, moeten de samenstellende objectmodulen worden samengevoegd tot een objectprogramma. Dit wordt gedaan door een programma dat een linker wordt genoemd. Naasr de linking loader, waarin de funkties van een linker en een loader zijn samengevoegd, worden nog behandeld:

- de linkage editor
- overlays.

10.2 Compile, load en go

De eenvoudigste manier om het objectprogramma in het hoofdgeheugen te laden is dat de assembler of de compiler de machinecode rechtstreeks in een door het operating system toegewezen gebied van dit geheugen zet. Zodra het hele bronprogramma naar machinecode is omgezet, wordt er naar de eerste instruktie van het objectprogramma gesprongen. Dit wordt aangeduid met assemble, load en go of compile, load en go.



Afbeelding 129

Deze vorm van laden van objectprogramma's werd in de begintijd van de computer erg veel gebruikt. tegenwoordig wordt het nog gebruikt voor kleine programma's die slechts éénmaal geassembleerd of gecompileerd en daarna gedraaid worden, b.v. programma's bij praktika voor studenten.

De belangrijkste nadelen van deze vorm van laden zijn:

- programma's die verschillende keren gebruikt worden moeten elke keer opnieuw geassembleerd of gecompileerd worden, hetgeen veel extra computertijd kost.

- het is erg moeilijk verschillende objectmodule samen te voegen, in het bijzonder wanneer de bronmodulen in verschillende talen zijn geschreven.

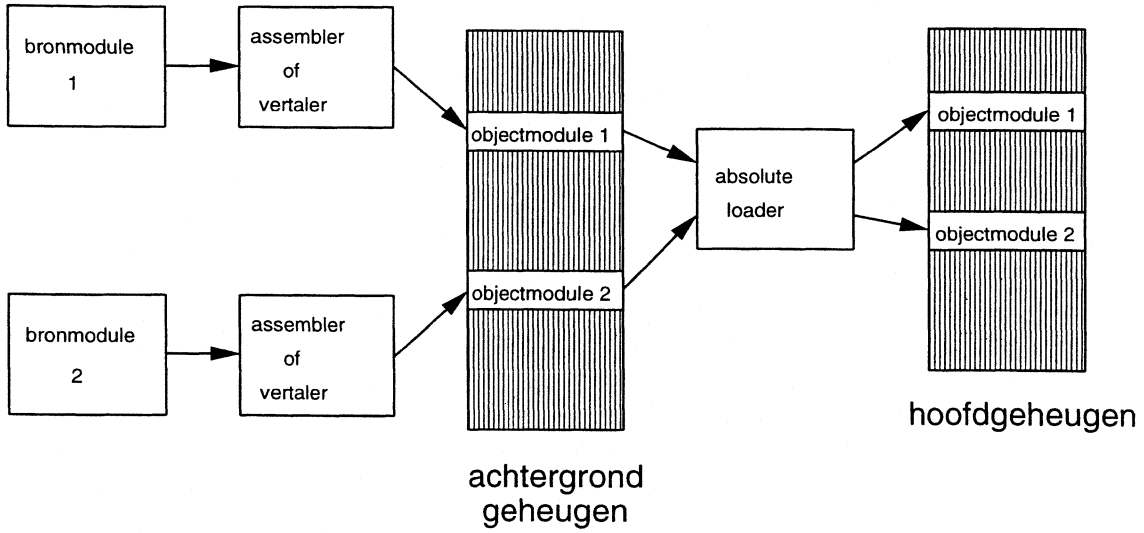
Het eerste nadeel kan worden ondervangen door het objectprogramma op een achtergrondgeheugen te zetten, waar het meer keren vanaf gehaald kan worden om te worden uitgevoerd. Hiervoor is een loader, die het objectprogramma in het hoofdgeheugen laadt, nodig.

10.3 De absolute loader

De absolute loader is het eenvoudigste type loader. De vorm van een objectprogramma dat op het achtergrondgeheugen staat is exact de vorm waarin het in het hoofdgeheugen wordt geladen. Het laden van een objectprogramma bestaat slechts uit het van het achtergrondgeheugen naar het hoofdgeheugen brengen, waarbij meestal een aantal controles worden uitgevoerd om de integriteit van het ingelezene te controleren.
De belangrijkste nadelen van deze vorm van laden zijn:

- alle in een objectprogramma voorkomende adressen van operanden en adressen in spronginstrukties moeten al bij het assembleren of compileren bekend zijn, en het objectprogramma moet altijd in het dan vastgelegde gebied van het hoofdgeheugen worden geladen, omdat de loader geen faciliteiten heeft om adressen aan te passen aan de plaats waar het objectprogramma in het hoofdgeheugen geladen wordt (de adressen in het objectprogramma zijn echte machine adressen of ***absolute adressen***); dit is een belangrijk nadeel in een computersysteem waarin meerdere objectprogramma's tegelijk in het hoofdgeheugen worden geladen, zoals in een computersysteem met meerdere gebruikers achter terminals, waarbij de plaats waar het objectprogramma geladen kan worden niet van te voren vastligt.

- als een programma uit meerdere modulen bestaat, moet bij het assembleren of compileren van een bepaalde module voor een adres uit een andere module eveneens het absolute adres worden genomen (als b.v. een bepaalde module een subroutine uit een andere module aanroept, moet bij het assembleren of compileren in de subroutine-aanroepinstruktie het absolute adres van deze subroutine worden gezet); er moet dan ook voor gezorgd worden dat elke objectmodule van het programma een afzonderlijk gebied in het hoofdgeheugen krijgt toegewezen.



Afbeelding 130

10.4 De relocating loader

Omdat het om de in 10.3 genoemde reden ongewenst is dat al bij het assembleren of compileren het adres wordt vastgelegd van de plaats in het hoofdgeheugen waar een objectprogramma geladen moet worden, is het beter een objectprogramma onafhankelijk van deze plaats te maken. Een relocating loader maakt dit mogelijk.

Een objectprogramma wordt reloceerbaar genoemd wanneer het op een willekeurige plaats in het hoofdgeheugen kan worden geladen om te worden uitgevoerd. Zo'n objecprogramma verschilt in twee opzichten van een objectprogramma in absolute vorm:

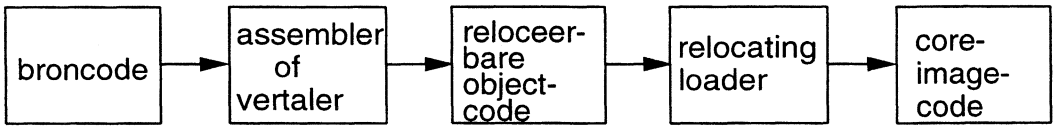
- adressen zijn ***relatief*** t.o.v het begin van het objectprogramma, i.p.v absoluut;
- aan het objectprogramma is ***relocatie informatie*** toegevoegd, wat aangeeft welke adresvelden in de instrukties veranderd moeten worden als het programma geladen wordt.

De relocatie informatie wordt gewoonlijk op één van de volgende manieren door de assembler of compiler aan het objectprogramma toegevoegd:

- aan elke machineinstruktie wordt een ***relocatie bit*** toegevoegd. Dit bit geeft aan of het adresveld van de instruktie wel of niet veranderd moet worden bij het laden.
- alle relocatie informatie wordt in een ***relocatie tabel*** gezet. Deze tabel bevat een verwijzing naar elke machineinstruktie waarvan het adresveld veranderd moet worden bij het laden.

Een relocating loader zet, m.b.v. de relocatie informatie, bij het laden van een reloceerbaar objectprogramma in het hoofdgeheugen alle relatieve adressen in het objectprogramma om in absolute adressen. Dit omzetten houdt in dat bij alle relatieve adressen wordt opgeteld het adres waar het begin van het objectprogramma wordt geladen. Het programma moet, nadat het geladen is, gedurende de hele uitvoering ervan wel op dezelfde plaats in het hoofdgeheugen blijven staan.

Nu er een onderscheid is gemaakt tussen de code die door de assembler of compiler wordt geproduceerd en de code die door de relocating loader in het hoofdgeheugen wordt geladen, gebruiken we hiervoor verschillende termen. De eerste blijven we objectcode noemen, de tweede noemen we ***core-image-code*** , hetgeen aangeeft dat het code is zoals die staat in het hoofdgeheugen, dat vroeger uit magnetische kernen (Engels: core) bestond.



Afbeelding 131

De assembler of compiler produceert een niet uitvoerbaar objectprogramma (een file), waarin relocatie informatie is opgenomen, en de relocating loader gebruikt de relocatie informatie om een uitvoerbaar ***core-image-programma*** in het hoofdgeheugen te laden.

De problemen bij het samenvoegen van verschillende objectmodulen tot één objectprogramma zijn nog niet opgelost bij een relocating loader. Dit leidde tot de ontwikkeling van linking loaders.

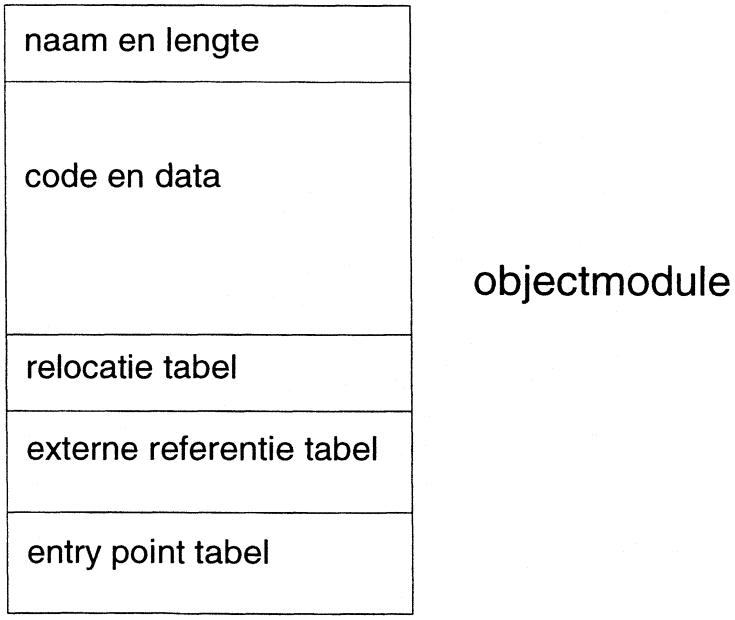
10.5 Linken, libraries en de linking loader

Het samenvoegen van een aantal objectmodulen tot één enkel objectprograma wordt linken genoemd. Een objectmodule is het resultaat van het assembleren of compileren van een bronmodule.

Een objectmodule bevat meestal de volgende elementen:

- vaste waarden die onafhankelijk zijn van de plaats in het hoofdgeheugen waar de objectmodule wordt geladen en daarom geen problemen veroorzaken.
- adressen waarvan de waarden relatief t.o.v. het begin van de objectmodule bekend zijn en die, voordat het programma uitgvoerd kan worden, absolute waarden gegeven moeten worden, hetgeen gedaan kan worden zodra het adres van het begin van de module bekend is.
- ***externe referenties*** , dat zijn verwijzingen naar objecten die in een andere module staan (b.v. de aanroep van een subroutine uit een andere module) en waarvan de waarde pas bepaald kan worden zodra alle objectmodulen bekend zijn.

Een objectmodule bestaat dan ook, zoals afbeelding 132 aangeeft, uit de volgende delen:



Afbeelding 132 objectmodule

- een aanduiding van de naam van de objectmodule en de lengte van code en data
- een blok code en data met relocatie informatie
- een tabel met voor alle externe referenties de symbolische naam van de referentie, b.v de naam van de aangeroepen subroutine, en de relatieve adressen van de plaatsen in het objectmodule waar de referentie voorkomt, b.v. de plaatsen waar een aanroep van de betreffende subroutine staat.

- een tabel met voor alle entry points, dat zijn objecten die in een module staan en in een andere module gebruikt mogen worden, de naam en het relatieve adres binnen de objectmodule. Als een module b.v. een subroutine bevat die in andere modulen gebruikt mag worden, bevat de tabel de naam van de subroutine en het relatieve adres binnen de objectmodule waar de subroutine begint.

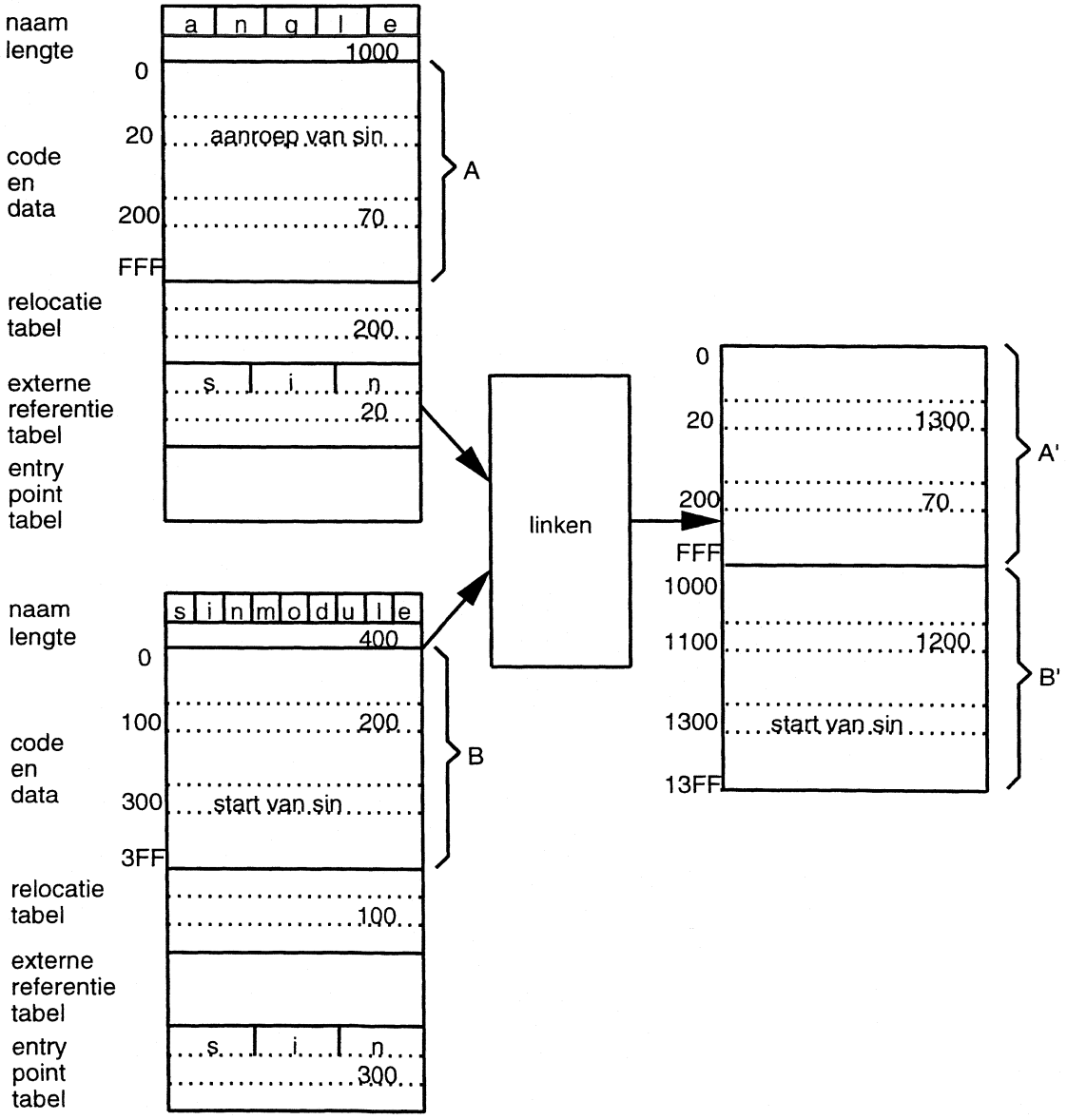
Het linken van een aantal objectmodulen wordt gewoonlijk in twee fasen gedaan, waarbij alle objectmodulen twee keer gelezen worden. In de eerste fase worden alle objectmodulen gelezen om een tabel met de namen en lengtes van de objectmodulen en een ***naamtabel*** met alle externe referenties en entry points te maken. Vervolgens wordt gecontroleerd of voor alle externe referenties het ermee overeenkomende entry point is gedeclareerd, en wordt aan elke objectmodule een adres relatief t.o.v. het eerste objectmodule toegekend en worden de adressen van de entry points hierbij aangepast. In de tweede fase worden alle objectmodulen opnieuw gelezen, en wordt bij alle adressen in een objectmodule het relatieve adres van de objectmodule t.o.v. de eerste objectmodulen opgeteld, en worden alle externe referenties ingevuld m.b.v. de informatie over de entry points in de naamtabel.
Een groot gedeelte van de te linken objectmodules komt meestal uit één of meer ***objectbibliotheken*** (Engels: object libraries). Een bibliotheek is een verzameling modulen die beschikbaar is gesteld om het programma te vereenvoudigen. Voorbeelden van bibliotheken zijn:

- invoer/uitvoerbibliotheken, met subroutines voor invoer en uitvoer van gegevens
- systeembibliotheken, met allerlei subroutines om het gebruik van het operating system te vereenvoudigen
- numerieke bibliotheken, met allerlei subroutines voor numerieke funkties, zoals sinus en logaritme.

Bibliotheken worden vaak verzorgd door de leverancier van de computer of de lokale systeemprogrammeurs, maar programmeurs kunnen ook hun eigen bibliotheken hebben. Bepaalde modulen uit bibliotheken worden vaak automatisch aan een programma gelinkt, zonder dat de programmeur dir expliciet hoeft te specificeren.

Een ***linking loader*** verzorgt zowel het linken van een aantal objectmodulen als het laden in het hoofdgeheugen van het daarbij geproduceerde objectprogramma. Bij het laden worden, m.b.v. de relocatie informatie, de adressen aangepast aan de plaats in het hoofdgeheugen.

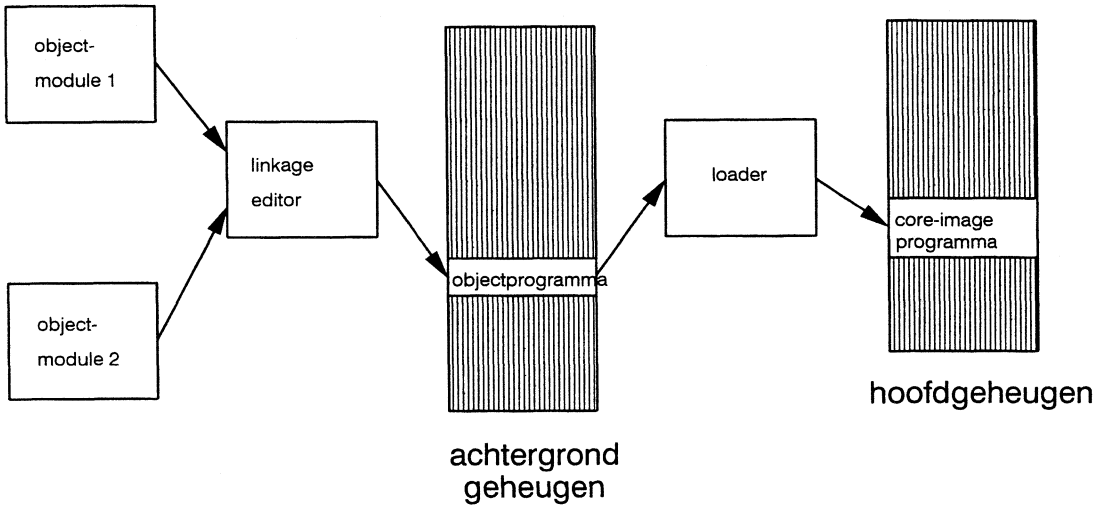
In afbeelding 133 is aangegeven wat er met de verschillende adreswaarden gebeurt als de module sinmodule gelinkt wordt aan het programma angle.



Afbeelding 133 linken van sinmodule aan angle

10.6 De linkage editor

De in 10.5 beschreven linking loader heeft als nadeel dat het linken van een aantal objectmodulen tot een objectprogramma elke keer herhaald moet worden wanneer het programma wordt uitgevoerd, hetgeen onnodig veel computertijd vergt. Dit kan voorkomen worden door de linkfase en de laadfase te scheiden.



Afbeelding 134

Een programma dat alleen het linken verzorgt wordt een linkage editor genoemd. Een linkage editor produceert een objectprogramma dat op het achtergrondgeheugen wordt gezet. Een loader haalt dit programma van het achtergrondgeheugen en laadt het in het hoofdgeheugen, waarbij met behulp van de relocatie informatie, de adressen worden aangepast aan de plaats in het hoofdgeheugen. In sommige systemen bevat het door linken verkregen objectprogramma nog relocatie informatie, zodat het nodig is naderhand een relocating loader te gebruiken.

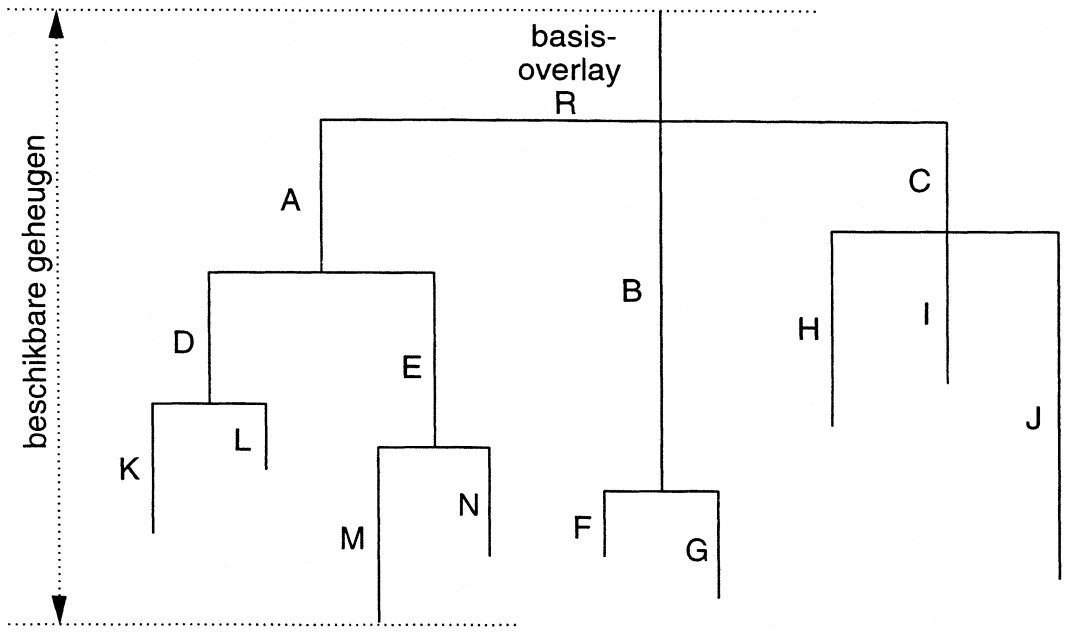
10.7 Overlays

Een objectprogramma is vaak groter dan de hoeveelheid hoofdgeheugen dat beschikbaar is om het te laden. Dit probleem kan worden opgelost door de programmeur zijn programma op te laten splitsen in een aantal delen die overlays worden genoemd. Elke overlay, die zowel data als code kan bevatten, neemt slechts een deel van de beschikbare hoeveelheid hoofdgeheugen in beslag. Alle overlays zijn op het achtergrondgeheugen opgeslagen. Om een objectprogramma te draaien, worden één of meer overlays ervan door de loader van het achtergrondgeheugen naar het hoofdgeheugen gebracht en wordt de uitvoering ervan gestart. Tijdens de uitvoering is het mogelijk dat in het hoofdgeheugen verblijvende overlays aan de loader doorgeven dat andere overlays van het achtergrondgeheugen naar het hoofdgeheugen moeten worden gebracht, waarbij deze nieuw te laden overlays op de plaats van één of meerdere aanwezige overlays komen te staan.

De programmeur is zelf verantwoordelijk voor het opsplitsen van een programma in overlays. Hij moet dit op een zodanige manier doen dat het transport van overlays tussen het achtergrondgeheugen en het hoofdgeheugen tot een minimum wordt beperkt. Elke overlay moet daarom een zoveel mogelijk op zichzelf staand en onafhankelijk van andere overlays draaiend deel

van het programma zijn. Zo moet er zeker voor gezorgd worden dat een groep instrukties die meerdere keren achter elkaar uitgevoerd moet worden zich niet over twee overlays uitstrekt, hetgeen er immers toe kan leiden dat de twee overlays bij elke doorgang van de insrukties beide een keer in het hoofdgeheugen moeten worden geladen.

Een erg eenvoudig scheme voor het gebruik van overlays wordt ***chaining*** genoemd. Een overlay wordt hierbij aangeduid met link. De eerste link wordt geladen en uitgevoerd. Aan het eind zorgt deze link ervoor dat ze wordt overschreven door de tweede link, die weer wordt overschreven door de derde, enz. Dit scheme vereist dat er een minimale hoeveelheid communicatie tussen overlays onderlingis, omdat er slechts één overlay tegelijkertijd in het hoofdgeheugen staat.



Afbeelding 135 overlayboom

In een meer geavanceerd schema voor het gebruik van overlays wordt gebruik gemaakt van een ***overlayboom***, waarvan hier een voorbeeld in afbeelding 135 wordt gegeven. De programmeur specificeert een ***basisoverlay*** R, die gedurende de gehele uitvoering van het programma in het hoofdgeheugen blijft staan. Het gebied in het hoofdgeheugen na de basisoverlay kan in beslag genomen door één van de eerste niveau overlays A, B of C. Als het in beslag wordt genomen door A, dan kan het gebied na A in beslag worden genomen door D of E. Als het D is, dan kan het volgende gebied door K of L in beslag worden genomen, als het E is, dan kan het of door M of door N in beslag worden genomen. K of L kunnen echter alleen dan in het hoofdgeheugen staan als ook D en A, en natuurlijk de basisoverlay, in het hoofdgeheugen staan. Als R, A, D en K in het hoofdgeheugen staan en in D op een bepaald moment L nodig is, moet deze overlay eerst worden geladen op de plaats waar K stond. Als in A op een bepaald moment E nodig is, moet deze overlay eerst worden geladen op de plaats waar D stond, en indien nodig, op de plaats waar K of L stond. Als in de basisoverlay op een bepaald moment F nodig is, worden zowel B als F geladen. Een bepaalde overlay kan alleen dan worden geladen als de overlay die zich in de overlayboom erboven bevinden al geladen zijn. De programmeur moet het begin en einde en de positie in de overlayboom van elke overlay specificeren. De assembler of compiler en de linker zorgen ervoor dat de benodigde aanroepen van de loader in het objectprogramma worden opgenomen.

De ontwikkeling van een geschikte en efficiënte overlaystruktuur voor een lang en gecompliceerd

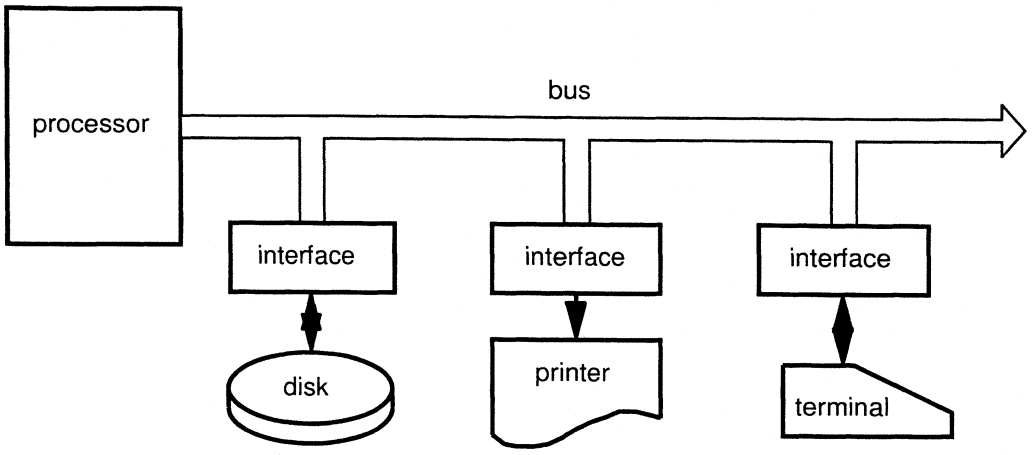
programma kan erg ingewikkeld zijn. Één van de doelstellingen bij het ontwikkelen van ***virtuele geheugen systemen***, was het volledig automatiseren van het gebruik van overlays, zodat het ongemak dat hieraan verbonden is, wordt overgeheveld van de gebruiker naar het operating system.

10.8 Virtuele geheugens

Bij de tot nu toe beschreven vormen van laden worden alle adresreferenties die eventueel gebruikt worden, aangepast door hetzij de linker, hetzij de loader, voordat de uitvoering van het programma begint. In het hoofdstuk van het diktaat waar het Operating System behandeld wordt, worden een aantal organisaties genoemd waarbij relocatie van een programma niet vooraf hoeft te gebeuren, maar door middel van hardware voorzieningen plaats vindt op het moment van laden van het programma door het vullen van één of meer registers die tijdens de executie voor de aanpassing van de, in de uitvoer van de vertaler of de linker opgegeven, logische of virtuele adressen naar fysieke adressen zorg dragen. Als men bovendien beschikt over segmenthardware kan tevens het linken tijdens executie geschieden.

11 Input en Output

Hoe mooi en indrukwekkend programma's in een computersysteem ook zijn, zonder input en output, kortweg I/O genoemd, zal men er niet veel van merken.
Daarom is elke computer verbonden met een aantal apparaten of systemen waarmee gegevens kunnen worden ingevoerd en resultaten naar buiten kunnen worden gebracht. Bij input kan men denken aan een simpele drukknop, maar ook aan een schijf met vele megabytes meetgegevens die verwerkt moeten worden. Evenzo kan een outputapparaat een schakelaar zijn, maar ook een apparaat dat ons bewerkte videobeelden laat zien. Al deze apparaten moeten op één of andere manier met de computer worden verbonden, en wel zodanig dat aan computerzijde de data slechts bestaat uit bits, gecodeerd in woorden van een vast aantal bytes met een voorgeschreven signaalniveau en een gelimiteerde verwerkingssnelheid.



Afbeelding 136 interfacing

In afbeelding 136 zijn de interfaces aan ene zijde verbonden met de bus van de processor en aan de andere zijde met de hardware van het randapparaat.
In feite is een interface een beschrijving van data en timing voor een bepaald verbindingssysteem, en de interface-adapter het kastje dat twee verbindingssystemen koppelt. Maar langzamerhand is men het woord interface gaan gebruiken voor het kastje met hardware (en vaak ook software) dat de signalen volgens de ene beschrijving omzet in signalen van de andere beschrijving.
Zo'n omzetting kan bijvoorbeeld bestaan uit signaalaanpassingen, codeomzetting en snelheidsaanpassing. In een randapparaat kunnen verschillende hoge (wissel)spanningen of grote stromen voorkomen die moeten worden vertaald naar 5 Volt en stromen van 1 milli Ampère. Een ander randapparaat, een meetapparaat, levert als meetwaarden getallen met 5 cijfers achter de komma, die netjes moeten worden gecodeerd in woorden van 1 of 2 bytes om als input van de computer te kunnen dienen. Ook snelheidsverschillen kunnen in een interface worden ondervangen. Als een randapparaat tijdelijk meer data levert dan direct door de computer kan worden verwerkt, dan kan de interface deze piekbelasting tijdelijk opslaan in een buffer (geheugen) en omzetten in een maximale computerbelasting gedurende een langere tijd.
Welk type interface moet worden gebruikt is dus sterk afhankelijk van het soort randapparaat dat moet worden aangesloten.

11.1 Randapparatuur

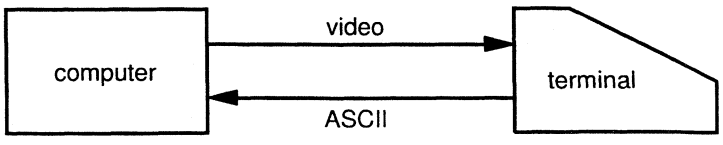
Randapparatuur is grofweg in de volgende groepen te verdelen:

- Randapparatuur voor communicatie met de menselijke gebruiker, waarbij het gaat om tekst, plaatjes of geluids- en lichtsignaleringen. Voorbeelden zijn terminal, muis, printer, controlepaneel, kassa ed.
- Achtergrondgeheugen, een vorm van geheugen waarin meestal grote hoeveelheden data gedurende korte of lange tijd kan worden opgeslagen. Voorbeelden zijn computertapes, harddisks, floppy disks, C(ompact) D(isks)'s waarvan in het bijzonder de CD-ROMS.
- Sensoren en aktuatoren, bijvoorbeeld gebruikt bij procesbesturing en onbemande controlestations. Zij vervangen en overtreffen hierbij de menselijke zintuigen en spierkracht. Ook niet door zintuigen waar te nemen signalen zoals radioactiviteit, radiogolven, infrarood en ultraviolet licht, ultrasoon geluid, aanwezigheid van giftige gassen vormen geen probleem.
- Andere computers. Computers staan zelden meer alleen, maar worden omringd door soortgenoten waarmee zij, via één of ander netwerk, één systeem vormen. Ook kleine lokale computers kunnen hun gegvens opsturen naar grote, centrale systemen waar de gegevens worden gecombineerd.

11.2 Mens-computer communicatie

11.2.1 De terminal

Het meest bekende randapparaat in deze categorie is de terminal. In feite bestaat het uit twee afzonderlijke randapparaten, de *monitor* of beeldbuis en het *keyboard* of toetsenbord. De monitor levert als output tekst en/of plaatjes en het keyboard verzorgd de tekstinput.
Een monitor bestaat uit een C(athode)R(ay)T(ube), een luchtledige glazen beeldbuis, waarvan het scherm bedekt is met 3 soorten fosforen, stoffen die in de kleuren rood, groen en blauw kunnen oplichten als zij getroffen worden door electronen. Deze electronen worden opgewekt in drie elektronenkanonnen, voor elke kleur één. Door slimme konstruktie en sturing kan elk kanon slecht fosforpunten van zijn "eigen kleur" treffen. De besturing voor de afbuiging van de electronenstralen zorgt ervoor dat het scherm geheel beschreven wordt met aaneensluitende horizontale lijnen van links naar rechts. Door de sterkte van de drie electronenstralen, gedurende dit schrijfproces, afzonderlijk in waarde te varieren, kunnen beelden worden opgebouwd die bestaan uit puntjes van elke gewenste kleur. (Door additieve kleurmenging is elke kleur samen te stellen door een gewogen gemiddelde van de kleuren rood, groen en blauw). Elk beeldpunt of *pixel* bestaat dus minstens uit het effect van 3 fosforpunten. De kleurinformatie alsmede de timingssignalen of *sync(hronisatie) signalen*, die aangeven op welk moment een nieuw beeld en wanneer een niewe beeldlijn begint, zijn gecodeerd in het *videosignaal* aanwezig. De kwaliteit van een monitor wordt bepaald door de grootte van de beeldpunten, de schermafmetingen, het aantal beelden dat per seconde gegenereerd kan worden, de te bereiken kleurverzadigingen (afhankelijk van de gebruikte fosforen) en of er interliniering wordt toegepast. Ter vergelijking: een Europees TV-toestel heeft een monitorgedeelte met verticaal 625 beeldlijnen van 800 pixels waarmee 25 beelden per seconde, geinterlinieerd, worden opgebouwd. Interlinering wil zeggen dat een beeld bestaat uit twee achter elkaar volgende halfbeelden, het eerste halfbeeld bestaande uit de beeldlijnen met oneven rangnummer en het tweede halfbeeld bestaande uit de beeldlijnen met even rangnummer.



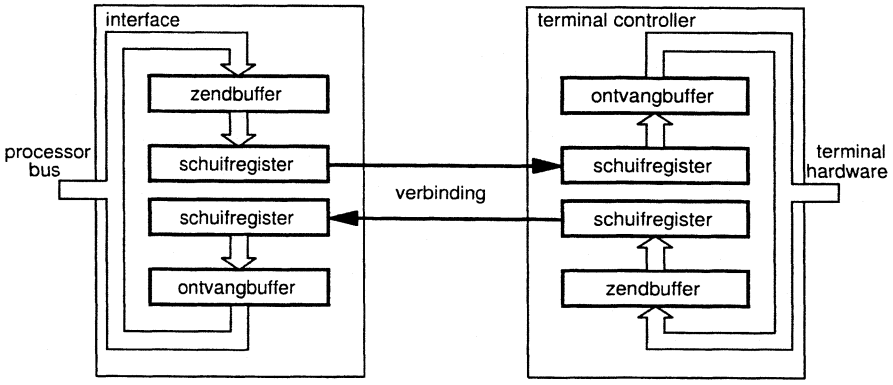
Afbeelding 137 de terminal

Bij een grafische terminal kan het beeld dus worden overgebracht met een standaard videosignaal. Het videosignaal moet dan aan computerzijde door een grafische kaart worden samengesteld. Bij zeer eenvoudige karakterterminals kunnen de uit 8 bits bestaande ASCII-patronen worden overgebracht en vindt de samenstelling van het videobeeld in de monitorhardware plaats.

Het andere gedeelte van de terminal, het toesenbord, is voor de invoer van karakters. Zoals bekend kunnen deze karakters volgens het ASCII recept vertaald worden naar 7-bit's woorden, maar meestal worden hiervoor 8-bit's woorden of bytes gebruikt. Bij het indrukken van een toets wordt het overeenkomstige bitpatroon door terminalhardware gevormd. Via een kabel met 8 parallelle signaaldraden kan dit eenvoudig naar de computer worden gebracht. In afbeelding 137 is grafische terminal aansluiting geschetst.

11.2.2 Serie - parallel omzetting

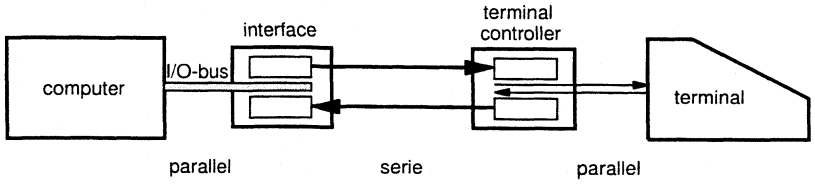
Als de kabels waarmee de karakters worden overgebracht wat langer worden is het makkelijker om de 8 bits niet parallel maar in serie te versturen. Door de bits in serie, dus na elkaar te versturen is per verbinding nog maar 1 signaaldraad nodig, maar de verzendtijd wordt 8 maal zo groot.



Afbeelding 138 serie-parallel conversie

Bovendien moet aan zendzijde een parallel -> serie en aan ontvangzijde een serie -> parallel conversie plaatsvinden. In afbeelding 138 is aangegeven hoe dit tot stand komt. De te verzenden bitcombinatie wordt in een schuifregister geladen, en vervolgens bit voor bit de verbindingslijn op geschoven. Aan ontvangzijde wordt het juist een schuifregister ingeschoven, om daarna parallel te worden uitgelezen. Het schuivenaan zend- en ontvangkant moet even snel gaan. Maar voor beide registers is geen gemeenschappelijke klokpuls beschikbaar. Daarom wordt de snelheid met elkaar afgesproken, en moeten beide zijden een klokpuls genereren die zo goed mogelijk aan die afspraak voldoet.

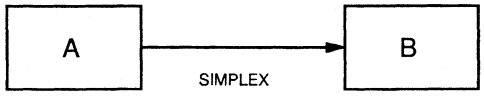
De zendbuffers zorgen ervoor dat gedurende de tijd dat een karakter wordt verzonden al een nieuw karakter kan worden klaargezet. De ontvangbuffers zorgen ervoor dat een ontvangen karakter nog gedurende de ontvangst van het volgende karakter kan worden uitgelezen.
Door deze buffering wordt de timing voor in- en uitlezen minder kritisch. In afbeelding 139 is de aansluiting van zo'n karakterterminal gegeven. Aan computerzijde zal de omzetting in de interface plaats vinden en aan terminalzijde door de terminalcontroller.
Er zijn in dit geval 2 signaaldraden nodig, voor elke richting één.



Afbeelding 139 serie aansluiting

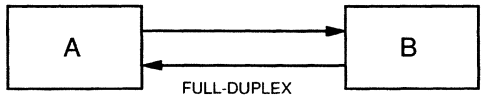
11.2.3 Transmissieconfiguraties

Voor verbindingen of transmissiewegen tussen twee plaatsen bestaan drie mogelijke configuraties: simplex, half-duplex en full-duplex.
Bij een simplex verbinding kunnen slechts gegevens over één enkele transmissieweg in één enkele richting worden verstuurd.
Dit is aangegeven in afbeelding 140.



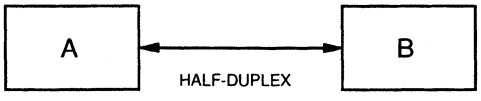
Afbeelding 140 simplex

Bij de full-duplex methode hebben we in feite een combinatie van twee simplex verbindingen in verschillende richtingen gecombineerd tot één transmissiekanaal dat het tegelijkertijd verzenden van data in beide richtingen mogelijk maakt.
Zie afbeelding 141.



Afbeelding 141 full-duplex

Bij de derde methode, de half-duplex, kan men langs één enkele verbinding, òf zenden, òf ontvangen, maar nooit beide tegelijk. Gedurende de data-overdracht over een half-duplex kanaal moet steeds één enkele kant zenden, terwijl tegelijk de andere kant luistert. Wanneer beide zenden, zullen beide verzonden boodschappen verloren gaan. Via een stelsel van afspraken wordt bepaald wie zendt en wie ontvangt. (deze afspraken vormen het zogeheten *protocol*). (Wanneer personen via een radioverbinding met elkaar spreken, schrijft het protocol bijvoorbeeld het gebruik van het woord *over* voor als men overgaat van zenden op ontvangen en de gesprekspartner moet overgaan van ontvangen naar zenden). Dit is aangegeven in afbeelding 142



Afbeelding 142 half-duplex

In afbeelding 139 was al te zien hoe data in serievorm van de ene plaats naar de andere plaats kan worden gezonden. Voor de koppeling van apparaten op seriebasis gebruikt men algemeen een standaard definitie, waarin is vastgelegd hoeveel lijnen de koppeling kan omvatten, wat de spanningsniveaus zijn, welke mogelijke transmissiesnelheden gekozen kunnen worden.

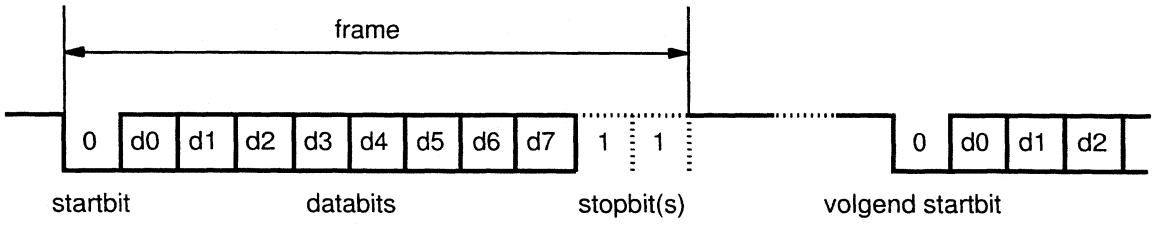
Voor transmissiesnelheden beneden 20K bits/sec is de algemeen geaccepteerde standaard de Electronic Industries Association norm RS-232-C, die nagenoeg gelijk is aan de CCITT norm V-24. Deze omvat 24 signaaldraden, waarvan men er in de praktijk vaak slechts 3 gebruikt. De spanningen die gebruikt worden om de 0 en de 1 te coderen moeten symmetrisch zijn t.o.v. de signaalaarde, en mogen gekozen worden van +3 Volt en -3Volt tot +15 Volt en -15 Volt. (Dit in tegenstelling met de "normale" digitale hardware, die 0 Volt en +5 Volt gebruikt).
Zelfs de te gebruiken kabelconnector (DB25) voor de verbindingskabels wordt voorgeschreven. Minimaal worden de signalen *Transmit Data , Receive Data en Signal Ground* gebruikt. (pennen 2, 3 en 7 op de DB25-connector) daarnaast zijn nog klaarmeldingssignalen en kloksignalen beschikbaar.

11.2.4 Synchroon en asynchroon

Voor de data overdracht in serie kan men gebruik maken van twee methoden: de asynchrone transmissie en de synchrone transmissie.

Asynchrone transmissie:

In afbeelding 143 is de positie van de bits voor asynchrone transmissie aangegeven. De aaneengesloten databits worden voorafgegaan door een zogeheten startbit. Dit bit, dat altijd de waarde 0 heeft, maakt de ontvanger erop opmerkzaam dat er een nieuw datawoord binnenkomt.

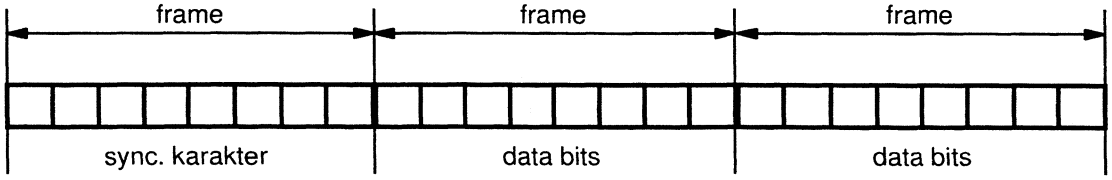


Afbeelding 143 asynchrone transmissie

Ter afsluiting worden nog één of twee stopbits met de waarde 1 toegevoegd. Dit geeft de verzekering, ook als de verzonden woorden elkaar direct opvolgen, de datalijn minstens één bit-tijd 1 is en dus tegengesteld aan het erop volgende startbit. De kloksnelheid van de ontvanger moet aangepast worden op de snelheid waarmee de bits ontvangen worden. Gangbare frequenties zijn 110, 300, 600, 1200, 2400, 4800, 9600 en 19200 bits/sec. Ook het aantal databits kan worden gekozen. Binnen de databits kan nog een zogenaamd pariteitsbit worden toegevoegd. Als de waarde van het pariteitsbit zodanig wordt gekozen dat het totale aantal enen in het datawoord+pariteit even is spreekt men van even pariteit en evenzo als het pariteitsbit ervoor zorgt dat het totale aantal enen oneven is heeft men met een oneven pariteit te maken. Als er 1 databit tijdens transmissie van waarde verandert, zal men dit op deze manier kunnen detecteren. Men weet echter alleen dat er een bit fout is, en niet welk bit dat is.
Het blok interface dat in afbeelding 138 staat aangegeven, voorzien van de instelmogelijkheden van bovengenoemde waarden, is in i.c.-vorm verkrijgbaar onder de naam UART, de afkorting van Universal Asynchronous Receiver/Transmitter. In de 80C51 is deze asynchrone transmissie uit te voeren met de in het SFR aanwezige registers SCON en SBUF op de plaatsen 98H en 99H. Via SCON is een en ander in te stellen, terwijl SBUS het bijbehorend dataregister is.
Om aan de RS-232 standaard te kunnen voldoen moet nog wel een i.c. worden toegevoegd om de 0 Volt en 5 Volt signaalwaarden om te zetten in -12 Volt en +12 Volt signaalwaarden.

Synchrone transmissie.

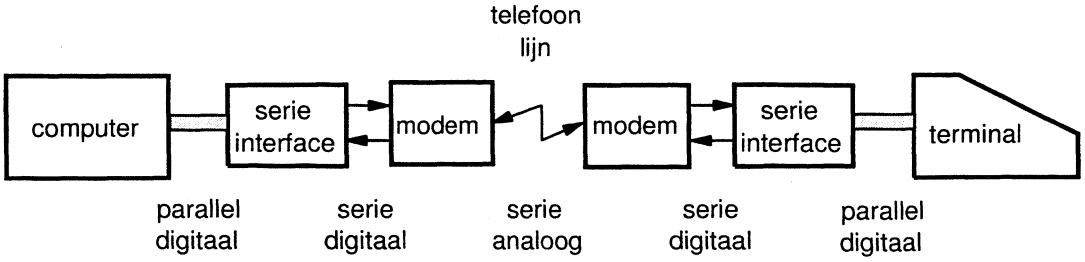
In afbeelding 144 is aangegeven hoe bij synchrone transmissie de tekens worden verzonden. Nu zijn er geen start- en stopbits toegevoegd, maar de tekens volgen elkaar naadloos op. Als aan ontvangzijde telkens netjes 8 bits worden afgeteld, weet men waar het volgende teken begint. Als er geen tekens beschikbaar zijn om verzonden te worden, dan worden speciale synchronisatietekens verzonden totdat er weer te verzenden data beschikbaar is. Aan ontvangzijde worden de synchronisatie tekens herkend en overgeslagen. Als om één of andere reden door een fout de synchronisatie verloren gaat, zal men moeten wachten op een synchronisatieteken om het aftellen van bits weer te kunnen starten. Doordat er nu geen start en stopbits in een teken aanwezig zijn kan men bij synchrone transmissie hogere snelheden bereiken dan bij asynchrone transmissie.



Afbeelding 144 synchrone transmissie

Voor synchrone transmissie is er een blokje verkrijgbaar onder de naam USRT, wat een afkorting is voor Universal Synchronous Receiver/Transmitter.

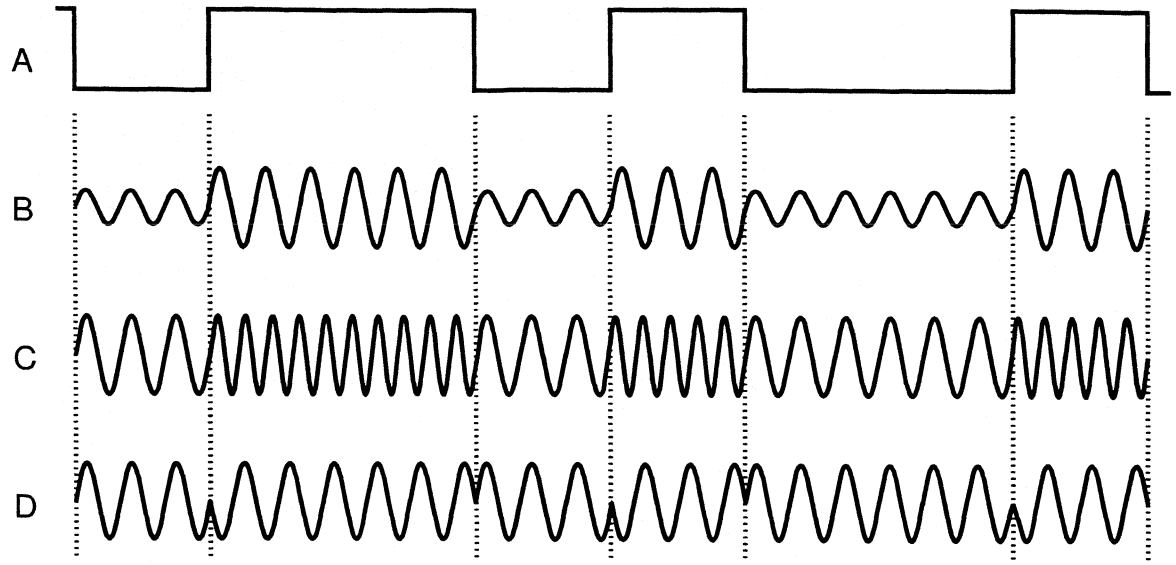
11.2.5 Modem



Afbeelding 145 communicatie met een modem

Bij wat grotere afstanden tussen zender en ontvanger is het prettig als gebruik gemaakt kan worden van bestaande verbindingslijnen. Zo'n bestaand systeem is het telefoonnetwerk. Maar dit netwerk is ooit ontworpen om spraaksignalen te over te brengen. Het gedeelte van de menselijke spraak dat voldoende is voor een goede verstaanbaarheid loopt van 400Hz tot 3000Hz. D.w.z dat het voor een goede telefoonlijn voldoende is om samenstellingen van sinusvormige signalen tussen 400 Hz en 3000 Hz te kunnen overdragen. Het netwerk is minder geschikt om direct digitale (blokvormige) signalen te transporteren. Om de informatie toch over te kunnen brengen worden de o en de 1 "vertaald" naar 2 sinusvormige signalen. In afbeelding 145 is aangegeven hoe de verbinding er dan uit komt te zien. De vertaling of modulatie wordt uitgevoerd in het modem. Aan ontvangzijde moet door terugvertaling het oorspronkelijke digitale signaal hersteld worden.

Ook deze omzetting, demodulatie genaamd, vindt in het modem plaats. Hier komt natuurlijk ook de naam mo(dulator-)dem(odulator) vandaan.
In afbeelding 146 zijn een aantal signalen getekend. Onder A is het digitale signaal weergegeven.



Afbeelding 146 modem signalen

Onder B is aangegeven hoe het signaal eruit kan zien als sinussen van verschillende sterkte of amplitude worden gebruikt. Dit staat bekend onder de naam *amplitudemodulatie* . Vaak wordt een modulatiediepte van 100% gebruikt, wat betekent dat de 1 door een sinussignaal en de 0 door geen signaal wordt aangegeven. Door de 0 en de 1 in sinussen met verschillende frequenties te vertalen wordt de signaalvorm C verkregen. Deze vorm van modulatie wordt *frequentiemodulatie* genoemd.
Een derde mogelijkheid is aangegeven met de signaalvorm onder D. Hier wordt de 0 en de 1 weergegeven met sinussen waarin een faseverschil aanwezig is. In het voorbeeld is voor 180° faseverschil gekozen. Deze vorm van modulatie wordt *fasemodulatie* genoemd.
De telefoonlijn laat het toe om meer dan twee frequenties tegelijkertijd te gebruiken. Als men de zender aan de ene kant bijv. 980 Hz en 1180 Hz toewijst en een zender aan de andere kant van diezelfde lijn bijv. 1650 Hz en 1850 Hz, is full-duplex verkeer mogelijk omdat beide zenders tegelijkertijd hun "nullen" en "enen" kunnen versturen zonder elkaar te storen.
Ook kan één zender meer frequentiecombinaties gebruiken. Als een zender bijvoorbeeld 3 frequenties en combinaties of "mengsels" daarvan mag gebruiken, dan zijn er voor elk seinteken 8 mogelijkheden (elke sinus kan in het seinteken wel of niet aanwezig zijn). Elk seinteken bevat dan 3 bits informatie. Om het aantal seintekens per seconde, de seinsnelheid, aan te geven wordt de maat *baudrate* gebruikt, terwijl het aantal bits per seconde door de *bitrate* wordt aangegeven. In het voorbeeld is de bitrate dus drie maal zo hoog als de baudrate. In het algemeen is dus bitrate = baudrate * ^{2}log (aantal mogelijke seintekens). Als er dus maar 2 mogelijke signaalwaarden voorkomen is de baudrate gelijk aan de bitrate.

11.2.6 De printer

De printer zorgt voor de hardcopy van ons beeldscherm. De meest gebruikte zwart/wit uitvoering is de laserprinter. Deze printer "bedrukt" telkens een heel vel papier. De oppervlakte van een trommel wordt eerst voorzien van een elektrische lading. Vervolgens wordt op de plaatsen waar het beeld wit moet worden de trommel "beschreven" met een laserstraal, waardoor het oppervlak plaatselijk ontladen wordt. Door nu de partieel geladen trommel door koolstofpoeder of toner te wentelen, ontstaat het beeld in koolpoeder. Dit beeld wordt afgerold op een vel papier. Door persen

en verhitten wordt de toner "vastgebakken", waardoor een houdbaar geschrift ontstaat. De kleinste weer te geven puntjes worden bepaald door de resolutie, die hier 600 tot 1200 D(ots) P(er) I(nch) kan zijn.
Omdat een bedrukt A4'tje uit meer dan $2 \cdot 10^7$ beeldpuntjes of pixels bestaat, wordt de informatie meestal niet in de vorm van beeldpuntjes of pixels overgebracht, maar in de vorm van een beschrijvingstaal. Elke printerfabrikant heeft hier een eigen idee over. Om toch een vorm van normalisatie te krijgen wordt nu vaak de taal PostScript gebruikt. Met dit hulpmiddel is het mogelijk een beeldopbouw te beschrijven in de PostScripttaal. Met een gewone karakterverbinding kan het PostScriptprogramma naar de printer worden verzonden, waarna het in de printer "gerund" wordt en een beeld genereert. In de printer moet dan dus een computertje aanwezig zijn die de PostScripttaal kan interpreteren. Het schrijven van "programma's" in PostScript wordt meestal gedaan door de programma's waarin de grafische informatie tot stand komt. Men kan ook zelf in PostScript programmeren, maar het is niet bepaald een gebruikersvriendelijke taal.
Een wat simpelere printer is de inktjetprinter. Deze printer heeft een printkop met een aantal inktspuitjes. Deze inkspuitjes, waarvan het aantal van 8 tot 24 kan varieren, zijn kolomsgewijs opgesteld. Door de printkop met "spuitkolom" in horizontale richting, heen en weer, regel voor regel over het papier te bewegen, en op het juiste moment de inktspuiten te bedienen, kan een tekst of plaatje worden samengesteld. Deze printers hebben meestal geen PostScript interpretator aan boord, maar kunnen slechts bediend worden met opdrachten die de printerfabrikant bedacht heeft. Is een type lettervorm of font, in de printer bekend, dan is karaktertransport voldoende, maar als dit niet zo is moet het als puntjesbeeld worden overgebracht, en kost nogal wat transporttijd.

11.2.7 De muis

Om de bediening van de computer verder te vereenvoudigen heeft men de zogenaamde muis geintroduceerd. Het is een apparaat waarmee een in het horizontale vlak uitgevoerde beweging naar de computer kan worden doorgegeven. Als deze bewegingsinformatie wordt overgebracht op een programmatisch daaraan gekoppeld pijltje of andere marker, kan de bediener een plaats op het beeldscherm aanwijzen. Door het apparaat bovendien met 2 of 3 drukknoppen uit te voeren, kan men bepalen wat op de aangewezen plaats moet gebeuren. Als op dat beeldscherm kant en klare commando's worden vertoond, kan de gebruiker dit commando doen uitvoeren door ernaar te wijzen en een drukknop te bedienen, en hoeft het commando dus niet in te typen. Het geven van fout ingetypte of niet bestaande commando's wordt hiermee uitgesloten.
De muis bestaat meestal uit een bolletje met enigszins ruw oppervlak die door wrijving de beweging over het tafelblad of muismatje overbrengt op 2 loodrecht op elkaar staande asjes. Aan de asjes zijn schijfjes met gleufjes bevestigd. Met behulp van een optocoupler, een combinatie van een ledlampje en bijhorende fototransistor, kan het aantal omwentelingen van een asje worden bepaald. Daarvoor wordt het aantal lichtflitsjes, dat het ledlampje via de gleufjes in de schijf opwekt en door de fototransistor wordt omgezet in spanningspulsjes, in een teller geteld. Door de telling voor beide assen gedurende een bepaalde tijd uit te voeren heeft men een goede indicatie van bewegingsrichting, snelheid en afgelegde weg. Meestal wordt de bewegingsinformatie, verpakt in 3 bytes, serieel, elke 100 msec naar de computer gestuurd. Het 1^e byte bevat de telling in x-richting gecodeerd als integer met teken, het 2^e byte de telling in y-richting eveneens als integer met teken en het 3^e byte de toestand van de drukknoppen. Voor de overdracht van deze 3 bytes kan het RS-232 protocol worden gebruikt.
Er zijn ook muizen die geen mechanische hulpmiddelen gebruiken, maar met lichtbron en fototransistor direct het passeren van een aantal lijntjes op een muizenmatje registreren, en op die manier de verplaatsing kunnen vaststellen.

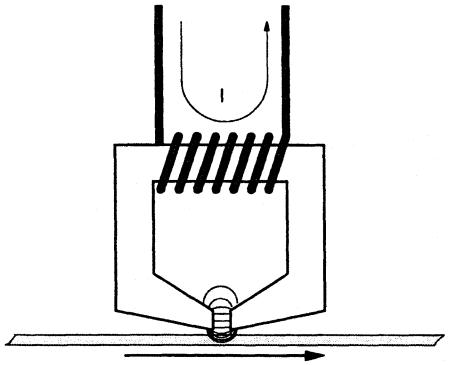
11.2.8 De scanner

Een nog niet genoemd apparaat is de scanner. Hiermee kan men de pixels van een plaatje op papier met een lichtbron beschijnen en de sterkte van het teruggekaatste licht opnemen. De opnemers, al of niet voor kleur, meten de pixel-eigenschappen, die vervolgens binair gecodeerd, naar de computer kunnen worden overgebracht. Per plaatje kunnen dit, afhankelijk van de resolutie waarmee dit gebeurt, vele megabytes zijn. Met behulp van bijbehorende software is het mogelijk om

uit deze pixelinformatie weer karakters te extraheren. Dit lukt goed bij gedrukte tekst, uitgevoerd in bekende fonts. Het is al moeilijker voor exotische fonts, en handgeschreven tekst.
Zo'n systeem, bestaande uit hardware en software, is in staat om O(ptical) C(haracter) R(ecognition) uit te voeren.
Een speciale uitvoeringsvorm is de scanner die de antwoordformulieren van het examenwerk verwerkt. Hier moet alleen op bepaalde posities gekeken worden of er een andere kleur dan rood voorkomt. Dit gaat bij normale invulling foutloos.
Een andere veel gebruikte scanner is de streepjescodelezer behorend bij een kassa. De "karakters" worden nu gevormd streepjes van verschillende mogelijke diktes. Om de streepjes van elkaar te onderscheiden zijn ze afwisselend wit en zwart, maar zowel de witte als de zwarte streepjes bevatten via hun dikte informatie. Er zijn verschillende coderingsrecepten in gebruik. Meestal worden alleen gecodeerde cijfers als karakters gebruikt.
Bij handscanners is de snelheid van bewegen en de variatie daarin een probleem. Gedeeltelijk is dit op te lossen door een vast synchronisatiepatroon aan het begin te zetten, en tijdens de scanning, softwarematig, de snelheid telkens opnieuw vast te stellen aan hand van de streepjes die gelezen zijn.
Als een laserscanner gebruikt wordt heeft men het voordeel van scanning met constante snelheid, en een klein en intens verlichtingsvlekje waardoor het streepjespatroon fijner kan worden.

11.3 Achtergrondgeheugen

Achtergrondgeheugen wordt gebruikt om gegevens te bewaren. Dit kan voor zeer korte tijd, maar ook voor zeer lange tijd zijn. Het liefst zou men alles gewoon in het RAM geheugen willen opslaan, maar dat is door de beperkte grootte daarvan niet mogelijk. Er moet dus gezocht worden naar methoden om grote hoeveelheden data tegen redelijke prijs te kunnen opbergen. Dan komt men terecht bij magnetische of optische registratie.



Afbeelding 147 lees/schrijfkop

Bij magnetische registratie wordt vrijwel altijd gebruik gemaakt van een dunne laag magnetiseerbaar materiaal, aangebracht op een magnetisch neutrale ondergrond, waarin via magnetische inductie een magneetveld kan worden aangebracht. Zowel lezen als schrijven kan met een lees of schrijfkop volgens afbeelding 147 worden uitgevoerd. Bij het schrijven zal ruwweg gelden :

$$\Phi = \alpha . I$$

Dus de magnetisatie is evenredig met de stuurstroom. De constante α wordt bepaald door het aantal windingen van de spoel, de magnetische eigenschappen van het materiaal van de kop, de eigenschappen van de magnetische laag, en de spleetbreedte en vorm van de kop.
Bij het lezen zal ongeveer gelden:

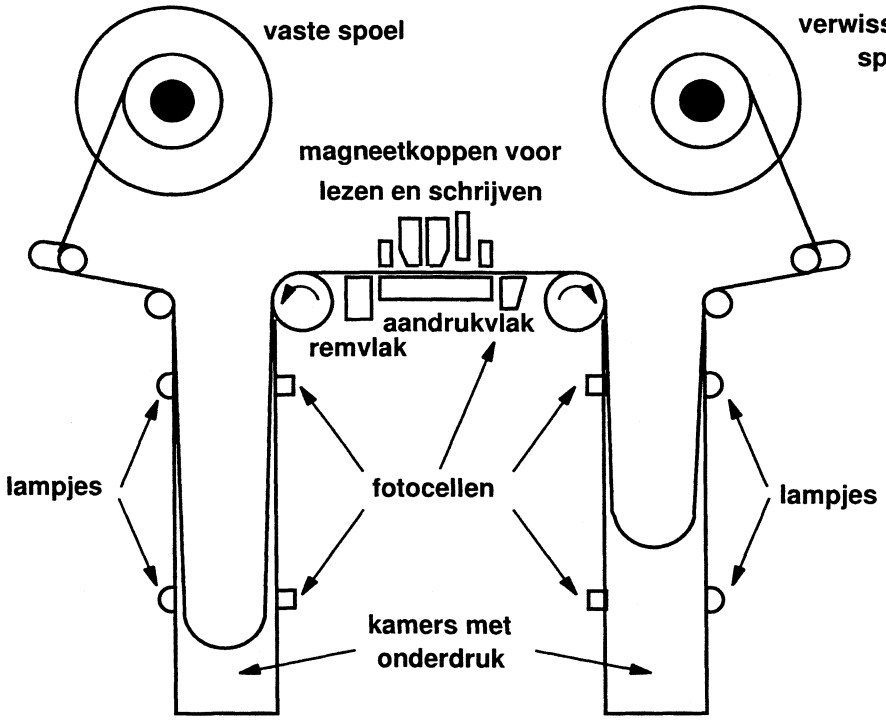
$$I = \beta . \frac{d\Phi}{dt}$$

Dus de opgewekte stroom is evenredig met de verandering van de magnetisatie. Hieruit zijn twee eigenschappen van magnetische registratie te zien nl het gelezen signaal is de afgeleide van het geschreven signaal en om iets te kunnen lezen moet de magnetische laag t.o.v. de kop met voldoende en konstante snelheid bewegen. Als een sinussignaal wordt geschreven zal er een sinussignaal met 90° fasedraaiing worden gelezen. Bij audioregistratie zal het menselijk oor weinig moeite hebben met deze fasedraaiing. Maar bij directe registratie van een blokvormig signaal zal het teruggelezen signaal, een positief pulsje gevolg door een negatief pulsje, niet tot het gewenste resultaat leiden. Voor magnetische registratie zijn dan ook verschillende coderingsrecepten bedacht om de bits te vertalen naar signaalvormen waarmee de schrijfkop gestuurd kan worden.

Bij optische registratie wordt gebruik gemaakt van de lichtreflecterende eigenschappen van een oppervlak. Hierbij wordt voor het "belichten" meestal een laser gebruikt, omdat hiermee een klein belichtingsoppervlak met hoge lichtintensiteit kan worden verkregen.

11.3.1 Magneetbanden

Bij magneetbanden is het te magnetiseren materiaal aangebracht op een dun polyester lint dat kan worden opgerold tot een spoel. Met één of meer lees/schrijfkoppen kunnen één of meer sporen, parallel aan de lengterichting van de band, van informatie worden voorzien en gelezen. In het voorgaande is al uiteengezet dat het lezen met konstante en hoge snelheid moet gebeuren. Als er niet gelezen of geschreven wordt staat de band stil. Om een bepaald item te kunnen lezen moet de band (of de kop) op snelheid worden gebracht om op de goede positie en met de goede snelheid dat item te lezen. Deze hele operatie is voor één byte niet efficient. Daarom wordt er niet met losse bytes, maar met blokken informatie gewerkt. Het lezen (en ook schrijven) van een heel blok bytes is de basisoperatie. Tussen de blokken bevindt zich een loze ruimte, de gap, om starten en stoppen tussen de blokken mogelijk te maken. Er zijn systemen waarin de gebruiker de fysieke blokgrootte kan kiezen, in andere systemen is de blokgrootte vast (hardwareafhankelijk). Met kleine blokken is er meer "gapruimte", bij grote blokken meer hardware en verwerkingstijd. De gebruiker kan wel meer logische records binnen een fysiek blok kiezen. Het aantal logische records opgenomen in een fysiek record noemt men de *blokkingsfactor* (Engels: blocking factor).



Afbeelding 148 magneetbandgeheugen

In afbeelding 148 is de schets van een magneetbandeenheid gegeven. Hierin is te zien op welke manier men de start- en stoptijden van de band heeft weten te bekorten. De te lezen of schrijven band wordt niet meteen van en naar de bandhaspels gebracht, maar uit een tweetal buffers. In de buffers worden de bandlussen met onderdruk gespannen en naar binnen getrokken. Per bandbuffer zijn twee detekties aangebracht die de minimale en de maximale vulling van de buffers kunnen detecteren. Deze optische detektie vormt geen mechanische belasting voor de band. Om de band te stopen of te starten heeft men schijnbaar slechts te maken met de massatraagheid van een klein stuk band. Met de detektiesignalen kunnen, afhankelijk van de bewegingsrichting van

de band, de opdrachten "meer band" of "minder band" omgezet worden in besturingscommando's voor de motoren die de haspels aandrijven.

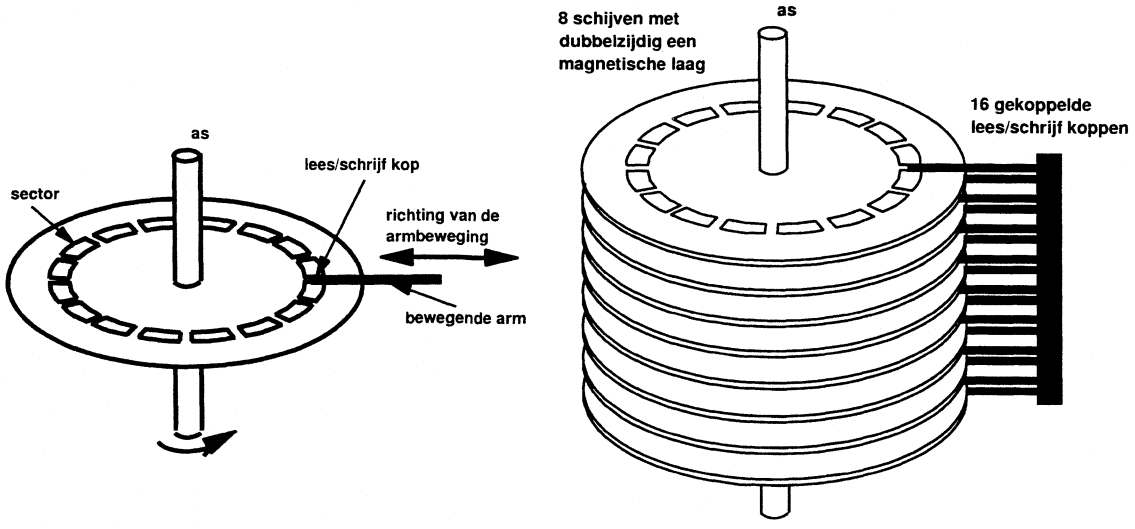
Een ander type bandeenheid is de *streamer*. Deze is speciaal voor back-up doeleinden geschikt. De start en stoptijden zijn minder belangrijk als men een grote hoeveelheid data als één groot blok moet wegschrijven, om later ook weer als één groot blok terug te lezen. De hoeveelheid data die op één band kan, en de snelheid waarmee die band geproduceerd wordt is van belang. Om grote hoeveelheden digitale data te dumpen zijn zelfs VHS-videotapes gebruikt. Een aangepaste videorecorder maakte elke nacht een totale dump van de schijven van een groot computersysteem. Voordeel was de goedkope tapes, maar een nadeel was dat de gegevens niet uitgewisseld konden worden met identieke gebruikers.

Bij magneetbandapparatuur komen zeer uiteenlopende coderingen van nullen en enen naar de stuursignalen van en naar de koppen toe. Ook de dichtheid waarmee de bits op de band gezet kunnen worden kan varieren van 800 tot 8000 bits per inch (bpi) of frames per inch als er meer sporen parallel geschreven worden. Telt men hier de varieteit aan voorkomende tapebreedtes bij dan is in te zien dat de uitwisselbaarheid niet erg groot is. Ook is het gevaar van veroudering van de informatie erg groot. Er worden in kasten nog steeds waardevolle informatie op tapes bewaard, die men nooit meer zal kunnen lezen omdat de bijbehorende tape unit al lang niet meer bestaat. (Vergelijk dit met het lezen van een floppy disk, een 5.25 inch floppy is nog goed leesbaar, maar een 8 inch floppy geeft al meer moeilijkheden.)

De datablokken op een magneetband staan in een lange rij achter elkaar. Om een bepaald blok terug te kunnen vinden kan de tape, voordat er data geschreven wordt, van een plaatsmarkering worden voorzien. Dit markeren levert een *geformatteerde* tape op. Dit markeren kan bijvoorbeeld gebeuren door een klokspoor aan te brengen. Het aftellen van de pulsen geeft dan de positie aan. Als er blokken van gelijke lengte worden gebruikt is het ook mogelijk vaste bloklabels (kleine blokjes met informatie over het volgend blok, gepositioneerd vlak voor de plaats waar later een blok moet worden geschreven) op de band te schrijven. Bij ongeformatteerde tapes zal men de tape moeten doorzoeken om het gewenste blok te vinden.

11.3.2 Magneetschijven

Zoals bij magneetbanden gegevens worden vastgelegd op een lange flexibele band, zo gebeurt dit bij magneetschijven op een ronde schijf, bestaande uit een magnetisch neutrale, van metaal of een legering vervaardigde, schijf waarop meestal aan beide zijden, een ferromagnetische laag is aangebracht.



Afbeelding 149 magneetschijven

Doordat de schijf snel ronddraait, is het circelvormig gebied waarboven de lees-anex-schrijfkop zich bevindt (om de construktie van de arm licht te houden hebben schijven in het algemeen gecombineerde lees/schrijfkoppen) serieel toegankelijk. (anders dan bij een grammofoonplaat, zijn de gegevens niet spiraalvormig opgeslagen, maar in concentrische ringen.) Net als bij een magneetband noemt men dit een spoor of track. Zo'n spoor is dan weer opgedeeld in blokken of sectoren die in hun geheel gelezen of geschreven worden. Meestal legt de fabrikant de blokgrootte vast, maar er zijn ook schijven waarvan de gebruiker de blokgrootte zelf kan specificeren.
In afbeelding 149 is de indeling van de magneetschijf in sporen, en de indeling van het spoor in sectoren aangegeven. In dezelfde figuur is de samenstelling van een schijvenpakket (Engels: disk pack) te zien. Het is een aantal boven elkaar gestapelde schijven, elk aan twee zijden voorzien van een magnetische laag, en door dezelfde as aangedreven. Voor elk van de magnetische oppervlakken is een lees/schrijfkop aanwezig. Ook de lees/schrijfkoppen zijn onderling verbonden, zodat alle koppen altijd boven (onder) overeenkomstige sporen op de oppervlakken gepositioneerd zijn. Het positioneren van de koppen moet nauwkeurig en snel gebeuren. De duur van deze mechanische handeling wordt seek-time genoemd, en ligt in de orde van 15 msec.
Het opzoeken van de juiste sector gebeurt door de schijf met een constante hoge snelheid rond te draaien, en te wachten tot de juiste sector onder de kop doorkomt. De tijd die hiermee gemoeid is wordt "rotational latency" genoemd en is ongeveer 5 msec.

Het kiezen van het juiste oppervlak kan worden bereikt door de gewenste kop in te schakelen. Dit omschakelen gebeurt electronisch. Hierbij moet men aan µsec denken. Het overschakelen naar een ander oppervlak is bijna te verwaarlozen t.o.v. de andere koppositioneringstijden. Daarom zal men bij elkaar horende gegevens het liefst opslaan op de verschillende sporen die horen bij eenzelfde koppositie. De verzameling sporen met hetzelfde spoornummer maar liggend op verschillende oppervlakken, noemt men een cylinder.
Er zijn twee methoden om de adressering van een sector binnen een spoor te verzorgen.
Bij hardsectoring voorziet de fabrikant de schijf van markeringen voor de plaats van de eerste sector en markeringen voor de plaats van de andere sectoren. Dit kan bijvoorbeeld een gaatje voor het aangeven van het begin van een omwenteling (index mark) en gaatjes voor elke sector (sector mark). De gaatjes worden met lichtdetectie uitgelezen. In plaats van sector marks kan ook een magnetisch klokspoor gebruikt worden. Nog een andere methode is om de sectornummers op een appart spoor magnetisch vast te leggen.
Bij softsectoring is er door de fabrikant niets voorbereid. De gebruiker zal dan, voor gebruik , de schijf kunnen voorzien van sectorlabels of headers voor de plaatsen waar de sectoren moeten komen. Dit voorbewerken is het formateren van de schijf.

drive nummer	cylindernummer	oppervlaktenummer	sectornummer

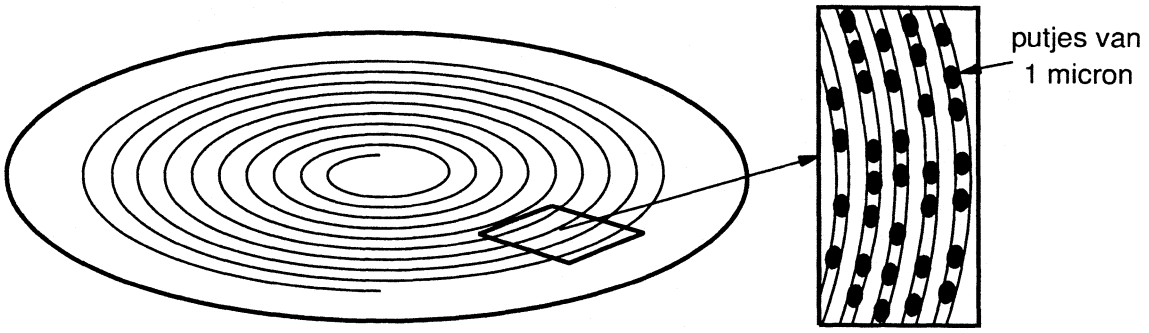
Afbeelding 150 diskadres

Als een disksysteem een aantal drives omvat, moet men deze diskunit nog kunnen aanwijzen. Een adres voor zo'n systeem heeft de opbouw volgens afbeelding 150.

Er zijn twee uitvoeringsvormen van schijven, nl de vaste schijf en de verwisselbare schijf.
De vaste schijf is gemonteerd in een hermetisch afgesloten doos, om stof en vochtdeeltjes te weren. Stofdeeltjes kunnen nl aanleiding geven tot een zogenaamde "head-crash" , waarbij magnetische laag en/of kop onherstelbaar beschadigd worden. Deze schijven hebben een opslagcapaciteit van 2 tot 10 Gigabyte ($10.2^{30} \approx 10^{10}$)
De meest bekende verwisselbare schijf is de floppy disk. De afmeting verminderde in de loop van de tijd van 8 inch via 5.25 inch naar 3.5 inch,terwijl de capaciteit juist vergrootte van 360Kbyte via 720Kbyte en 1.44Mbyte naar 120Mbyte.

11.3.3 C D ROM

Optische dataregistratie wordt gebruikt bij de C(ompact) D(isk) R(ead) O(nly) M(emory). Met behulp van een laser worden putjes van 1 micron (10^{-10} meter) in een masterschijf gebrand. Met deze masterschijf kan een matrijs gemaakt worden. Vervolgens kunnen met de matrijs een groot aantal plastic kopieën worden vervaardigd, die worden bedekt met een dun laagje aluminium, en worden afgewerkt met een doorzichtige afdeklaag ter bescherming. Langs deze weg is 650 Mbyte aan informatie op goedkope manier te vermenigvuldigen.



Afbeelding 151 CD Rom

In afbeelding 151 is te zien dat de putjes worden aangebracht in een spiraalvormige baan. In tegenstelling tot een grammofoonplaat wordt de spiraal van binnen naar buiten doorlopen. Met behulp van een elektronisch regelmechanisme wordt een laserstraal langs de spiraal geleidt, waarbij de snelheid langs deze spiraal konstant gehouden wordt. (dat betekent dat de rotatiesnelheid lager wordt naarmate er meer naar de buitenrand gelezen wordt). Bij het treffen van een putje zal minder licht worden teruggekaatst dan bij het treffen van het (spiegelend) oppervlak. De lichtsensor die het teruggekaatste licht opvangt, zal in het uitgangssignaal dus het putjes patroon hebben overgenomen. In het putjes patroon is de data gecodeerd. Hierbij is er rekening mee gehouden dat het kopieërproces foutgevoelig is. Door het gebruik van codes die een hoeveelheid controlebits toevoegen aan de databits is het mogelijk om fouten bij het lezen te herstellen.
De nullen en enen zijn niet afgebeeld op "geen putje" en "putje", maar worden gevorm door de overgangen van deze toestanden.

Naast bovengenoemd type is er nog de writable CD ROM's. Met een speciaal schrijfapparaat kan een gebruiker zelf zijn C D ROM's maken, door met een laser putjes in de schijf te branden. Dit is een éénmalig proces, ingebrande putjes kunnen niet meer verwijderd worden. (Intens en langdurige zonbeschijning blijkt er wel iets te kunnen bijbranden, wat de schijf dus vernietigt).

Dan is er nog de rewritable CD ROM. Op deze schijf is een materiaal aangebracht dat magnetiseerbaar is als het warmer dan de zogenaamde curie temperatuur is. Door de magnetisatie kan het kristalrooster worden veranderd, en daarmee ook de reflecterende eigenschappen voor laserlicht.
De schrijfkop is samengesteld uit een laser en een magneetspoel. Het kleine gebiedje dat door de laser snel wordt verhit kan door de spoel van de gewenste magnetisatie worden voorzien.
Het lezen van de informatie kan weer door middel van een gewone laser worden uitgevoerd.

11.4 Sensoren en Aktuatoren

11.4.1 Sensoren of opnemers

Veel voorkomende "randapparaten" zijn de opnemers of sensoren. Het zijn meestal kleine goedkope blokjes die een of ander fysisch verschijnsel waarnemen en/of zelfs meten qua sterkte en dit omzetten in een elektrische grootheid met overeenkomstige waarde. Enkele voorbeelden:

fysische grootheid	sensor	uitgang
temperatuur	NTC en PTC	weerstandswaarde
temperatuur	halfgeleiders	stroom
vervoming	rekstrookje	weerstandswaarde
licht	LDR	weerstandswaarde
licht	fotodiode, fototransistor	spanning
magnetisme	hallsensor	spanning
magnetisme	magnetische veld sensor	weerstandsbrug
druk	folie van geleidend rubber	weerstandswaarde
(ultra)geluid	piëzo sensor	spanning

Om een weerstandswaarde te kunnen bepalen wordt deze weerstand vaak opgenomen als frequentiebepalend element in een oscillatorschakeling. Het bepalen van de frequentie van een signaal komt neer op tijdmeting, en is door een microcontroller eenvoudig uit te voeren. (Vorm van A/D omzetting). Vaak heeft de fabrikant van de sensor de omzetting van weerstand naar frequentie al geïntegreerd in het sensorblokje.
Een andere vorm van integratie vindt men bij de infrarooddetector die gebruikt wordt bij afstandsbedieningen. Het gebruikte signaal bestaat uit pulsen van variabele lengte, waarbij de pulsen bestaan uit met 36KHz gemoduleerd infrarood licht. De gebruikte sensor heeft een infraroodfilter, een selectieve 36KHz versterker en een demodulator aan boord om meteen een pulssignaal op TTL niveau te kunnen afleveren en kan meteen aangesloten worden op een processor. De totale sensor heeft de grootte van een transistor en kost een paar gulden.

11.4.2 Aktuatoren

Als een outputdevice mechanische krachten moet leveren kan het niet meteen op de computer worden aangesloten, maar er is een vorm van versterking nodig. Vaak wordt voor het opwekken van een kracht of verplaatsing gebruik gemaakt van een elektromagneet om het gewenste doel te bereiken. De computer kan wel de grootte van de stroom of het moment van schakelen bepalen, maar de stroom zelf wordt bijvoorbeeld geleverd door een krachttransistor. Voorbeelden zijn motoren, stappenmotoren, relais (elektro-mechanische schakelaar), ventielen, luidsprekers enz.
Moet de computer apparatuur in en uitschakelen die met de netspanning van 220 Volt wisselspanning werkt, dan kan dit met relais, maar er zijn ook speciale thyristoren en triacs ontwikkeld die meteen aangesloten kunnen worden op de microcontroller. Aansturing, versterking en galvanische scheiding worden door één blokje verzorgd.
Op plaatsen waar flinke vermogens worden geschakeld of geregeld kunnen ook grote stoorsignalen optreden. Om zich hiertegen te wapenen moeten verbindingen naar de computer worden afgeschermd en moet de voeding goed worden ontkoppeld.

11.5 Computerverbindingen

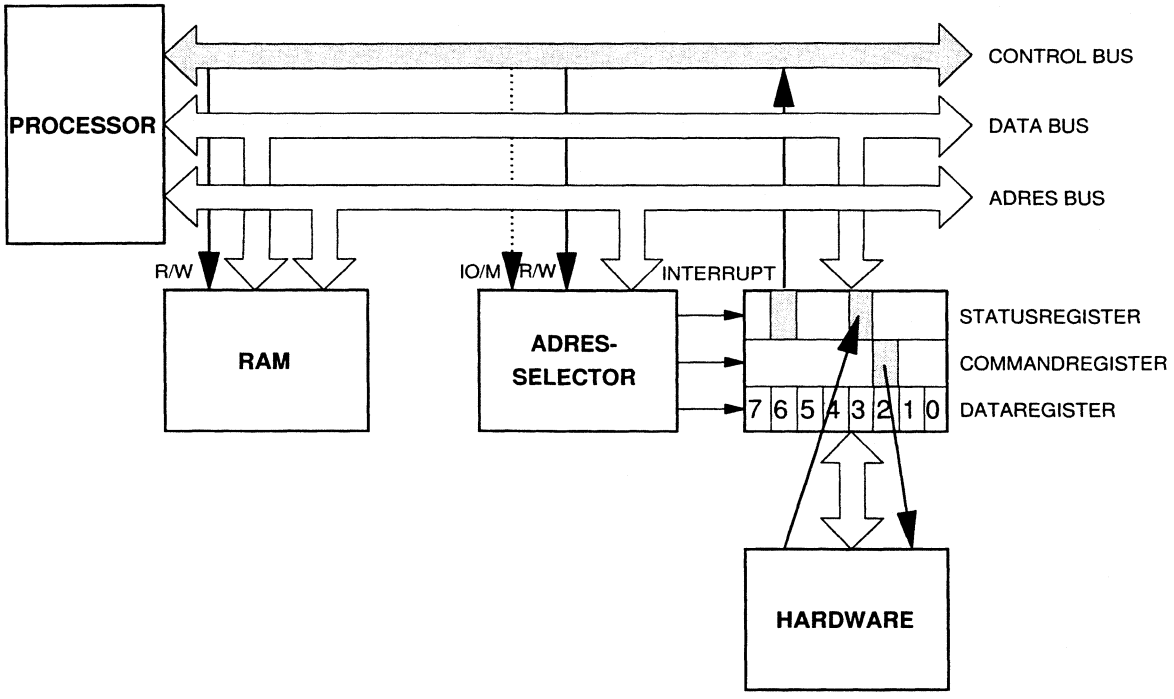
Voor computerverbindingen wordt gebruik gemaakt van allerlei netwerken. Zo zal een aantal PC's samen met een wat krachtigere PC, een server, onderling verbonden door een lokaal netwerk, de computerondersteuning van een afdeling kunnen vormen. Allerlei dure randapparatuur wordt niet lokaal op elke PC aangesloten, maar aan het netwerk gekoppeld zodat elke lokale gebruiker deze apparaten kan bereiken. Om gegevens met andere lokale systemen te kunnen uitwisselen worden deze lokale netwerken onderling weer verbonden tot een groter systeem. Wereldwijd gekoppelde systemen maken het internet populair. Om deze koppelingen te kunnen maken moeten de systemen op bepaalde punten met elkaar overeenstemmen.
Of men voor een internetaansluiting een kabelTV aansluiting, een telefoonlijn of een lokaal TU-netwerk gebruikt is voor de internetgebruiker onbelangrijk, maar op hardwareniveau zijn de kabels maar niet zo te verwisselen. Om systemen toch over systemen te kunnen praten en te kunnen vergelijken heeft "The International Standards Organization" (ISO) een model voor "Open Systems Interconnection" (OSI) gemaakt, waarin een netwerk op 7 niveaus kan worden beschouwd.
De lagen waarin men het systeem bekijkt zijn:

1. Physical layer
   Hier gaat het om het transport van het bit. Type kabel (UTP-kabels, glasvezelkabels, coaxkabels, radioverbinding enz.) met bijbehorende connectoren en de hardware voor de interfacekaart zijn hier belangrijk.
2. Data link layer
   Hierin komt aan bod hoe "pakketjes" bits met vaste lengte, foutvrij, van zender naar ontvanger worden verstuurd. Ook de foutdetectie is ingedeeld in deze laag.
3. Netwerk layer
   In deze laag wordt de adressering en decodering van het adres bekeken. Hoe kom ik op een netwerk, en hoe bereik ik de ontvanger zijn de vragen die hier gesteld worden. Met behulp van adresinformatie moet een bericht zijn weg kunnen vinden over verschillende type netwerken.
4. Transport layer
   Hier wordt de software van de zender en de ontvanger mede in beschouwing genomen. Dit is meestal ook de laatste laag die nog op operatingsystemniveau wordt afgehandeld.
5. Session layer
   Deze laag bemoeit zich met de protocols (afspraken) met hun methoden van afhandeling, die gebruikt kunnen worden om processen met elkaar te laten communiceren.
6. Presentation layer
   Verschillen in datarepresentatie spelen hier een belangrijke rol. Hoe karakters en andere data moet worden omgezet om toch met elkaar te kunnen communiceren is hier het onderwerp. De interface met de mens zou onafhankelijk van het type systeem en codering van de boodschap moeten worden gemaakt.
7. Application layer
   Hierin is een verzameling standaardprotocols ondergebracht, zoals file transferprotocols, electronic mail, remote login, virtuele terminals, enz.

Bij het ontwerpen van een nieuwe verbinding, bijv voor het overbrengen van beeldinformatie vanaf een ver verwijderde TV bewakingscamera, kan men de behoefte in elke laag analyseren en beslissen of men van bestaande dan wel nieuwe middelen gebruik maakt. Vragen als: zijn er al verbindingen, kan ik de bestaande protocollen gebruiken, en zijn er al geschikte coderingsrecepten en is er al toepasbare beeldbewerkingssoftware komen vanzelf naar voren als men de verschillende lagen van het OSI model de revue laat passeren.

11.6 Interfacing

Interfacen is het verbinden van een randapparaat met de computer op een zodanige manier dat de spanningsniveaus, de codering en de timing overeenkomen met het computersysteem. Hetzelfde probleem doet zich voor bij het aansluiten van geheugen, maar er zijn wat verschillen. Van een geheugen wordt een bepaalde snelheid geëist, maar het snelheidsgedrag van een randapparaat kan over een groot traject variëren. Bovendien moet er een dataverbinding met de buitenwereld zijn.



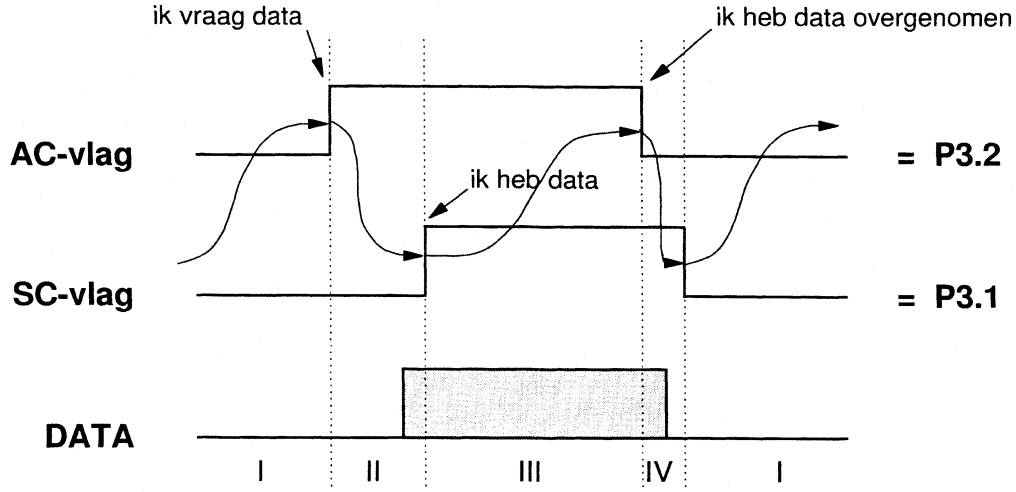
Afbeelding 152 Het aansluiten van een I/O apparaat

Zoals in afbeelding 152 is te zien dat de interface een dataregister bevat dat van buiten af bereikbaar is. Om de timing te kunnen regelen zijn nog twee besturingsregisters toegevoegd, nl het command register dat alleen beschreven kan worden, en het statusregister dat alleen gelezen kan worden. Deze registers zijn, vanuit programmastandpunt gezien, gewoon adresseerbare geheugenplaatsen, maar of er gelezen of geschreven mag worden hangt af van de boodschap van het randapparaat dat het er klaar voor is.
In het statusregister kan het randapparaat aangeven in welke toestand het zich bevindt. Dit kan een gereedmelding zijn, maar ook een melding van een opgetreden fout of andere belangrijke informatie. (bijv bij een printer: geen papier, zwart-wit mode of ontbrekend font). De computer kan dit statusregister lezen. Verder kan er nog een interruptbit aanwezig zijn. Zo'n bit wordt eigenlijk niet door de computer gelezen, maar vraagt zelfstandig via de hardware de aandacht van de computer. De werking van dit bit zal in het vervolg van dit hoofdstuk duidelijk worden.
Via het commando register, dat door de computer beschreven kan worden, kunnen de startmomenten van akties worden doorgegeven, maar ook aanvullende informatie hoe die akties moeten worden uitgevoerd. (Bij een RS-232 interface zal bijvoorbeeld de baudrate moeten kunnen worden ingesteld).
Als de registers van het randapparaat op dezelfde manier aangesloten zijn als geheugen, spreekt men van *memory mapped I/O* . Sommige microprocessoren hebben speciale instrukties voor I/O, samen met een adreslijn M/IO die het geheugenbereik verdeelt in een adresbereik voor echt geheugen en een, meestal kleiner, adresbereik voor randapparatuur. Het signaal M/IO selecteert dan het gewenste bereik. In dit geval spreekt men van *IO mapped IO* .

Bij de 80C51 zijn er 3 randapparaten en een aantal IO registers ingebouwd in de S(pecial) F(unction) R(egisters). Maar als er behoefte is aan een groter aantal I/O poorten, kan men deze gewoon memory-mapped aansluiten in het gebied voor het externe RAM-geheugen.
In het vervolg zal nu bekeken worden hoe met deze status- en commandsignalen computer en randapparaat met elkaar in de pas kunnen blijven.

11.6.1 I/O onder programmabesturing

Een veilige manier om een randapparaat qua timing te besturen, is gebruik te maken van het *volledige handshake* principe. Hierbij wordt met één input en één outputsignaal een vaste volgorde van besturingssignalen gemaakt, waarbinnen de dataoverdracht plaats vindt. In afbeelding 153 is dit aangegeven. De ontvanger of *acceptor* bestuurt de AC-vlag, en de zender of *source* bestuurt de SC-vlag. In het voorbeeld vervult de computer de rol van acceptor, en is de source een input-apparaat.



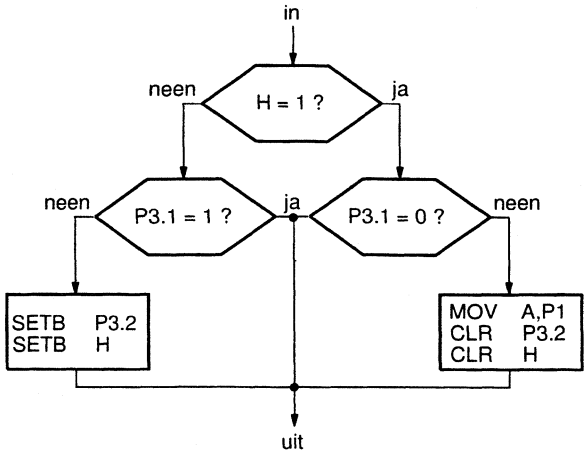
Afbeelding 153 handshake

Aan computerzijde wordt de AC-vlag door het outputsignaal P3.2, en de SC-vlag door het inputsignaal P3.1 gevormd. Nadat de computer gekeken heeft of het inputkanaal vrij is (P3.1 = 0) vraagt hij om data door de AC-vlag hoog te maken (P3.2 = 1). Dan wordt er gewacht op het antwoord dat er data beschikbaar is (P3.1 =1). Als de data is overgenomen geeft de computer dit aan door de AC-vlag weer laag te maken. (P3.2 = 0). Vervolgens wordt er weer gewacht op het vrijgeven van het inputkanaal via de SC-vlag. Op deze manier wordt de tijd verdeeld over 4 fasen I, II, III en IV. I vormt de fase dat het kanaal in rust is, voor II is de tijd die de source nodig heeft om data klaar te zetten, III is de tijd die de acceptor gebruikt om de data over te nemen en IV is de reactietijd van de source, waarna de rusttijd weer door de acceptor bepaald wordt. Om en om zijn acceptor en source verantwoordelijk voor een wachttijd. Elke signaalverandering van een vlag is het gevolg van de andere vlag. In de volgorde kan nooit een verandering optreden.
In principe kan de computer het afhandelen met:

```
P1   EQU    090H
P3   EQU    0B0H
   ......
T1   JB     P3.1,T1    ;is er rust?
     SETB   P3.2       ;vraag om data
T2   JNB    P3.1,T2    ;is er data?
     MOV    A,P1       ;haal data binnen
     CLR    P3.2       ;klaarmelding
     SJMP   T1         ;opnieuw
```

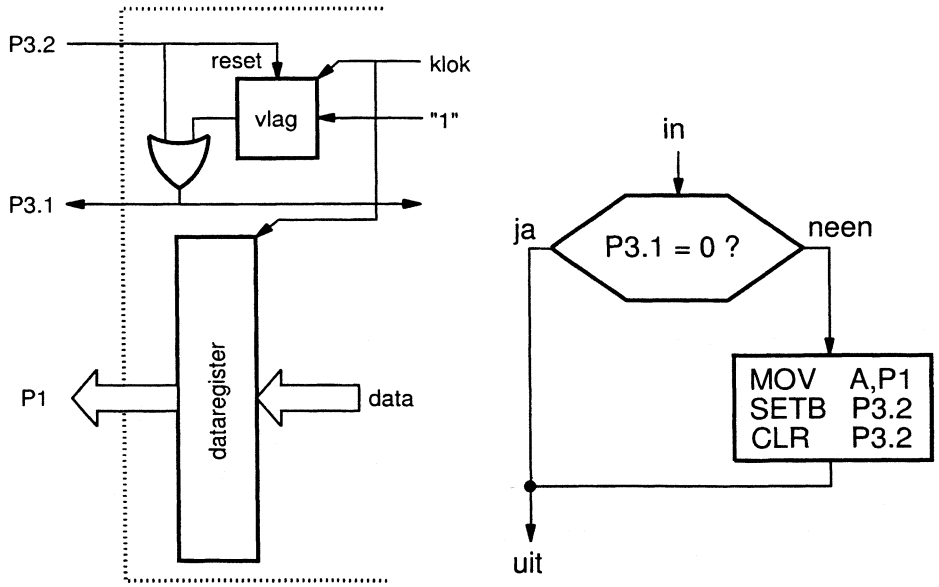
In het programma zijn bij de labels T1 en T2 twee wachtlussen die bij langzame randapparaten vaak doorlopen worden, terwijl de computer ook een andere taak had kunnen uitvoeren.
Met het invoeren van een hulpvariabele H, die bijhoudt welke flank aan de beurt is om getest te worden, kunnen de lussen worden samengevoegd. De routine ziet er dan uit als aangegeven in afbeelding 154.

```
IN   JB    H,T2
T1   JB    P3.1,UIT
     SETB  P3.2
     SETB  H
     SJMP  UIT
T2   JNB   P3.1,UIT
     MOV   A,P1
     CLR   P3.2
     CLR   H
UIT  ......
```



Afbeelding 154 I/O routine

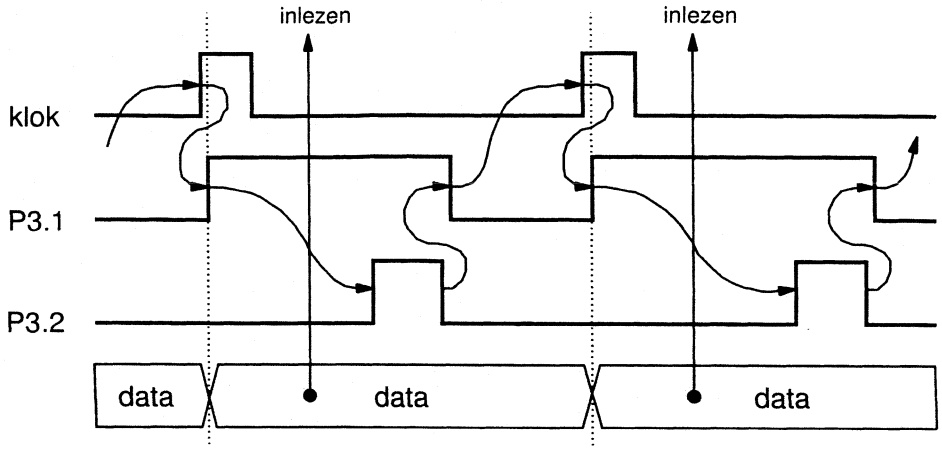
Een andere mogelijkheid is om niet een volledige handshake te gebruiken, en dus niet vier tijdsintervallen te onderscheiden, maar de tijd slechts in twee intervallen te verdelen.



Afbeelding 155 besturing met vlag

In afbeelding 155 is een hardware oplossing en de afhandelingsroutine gegeven. Beide deelnemers kijken naar dezelfde vlag, en kunnen die ook beide veranderen. De of-poort is toegevoegd om de vlag pas terug te zetten nadat de resetpuls is beëindigd. De vlag geeft aan wie van de twee deelnemers aktief mag zijn. Als er meer randapparaten zijn heeft elk apparaat zijn eigen vlag en data-register. De afhandelingsroutines kunnen achter elkaar gezet worden, zodat bij elke I/O service alle randapparaten bijgewerkt worden.

Als de dataregisters voorzien zijn van een three-state uitgang, kunnen zij een bussysteem vormen.De toegang tot dit bussysteem wordt bepaald door de stand van de vlag en een adresselectie van de processor.



Afbeelding 156 timing bij het vlagsysteem

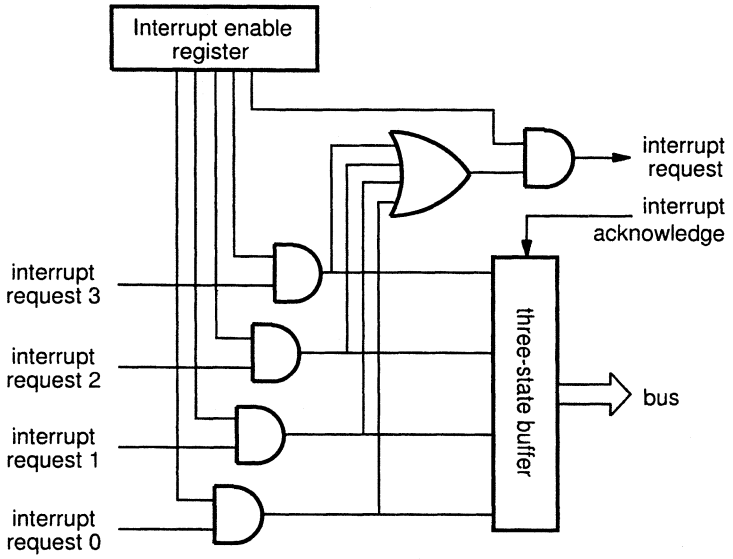
In afbeelding 156 is de timing met vlagbit getekend.

11.6.2 I/O met interrupt

Bij I/O onder programmabesturing moet het hoofdprogramma periodiek, na een bepaalde tijd, onderbroken worden om te testen of er input of output moet plaatsvinden. Wordt de bemonsteringstijd lang gekozen, dan geeft dit een slechte service, wordt de bemonsteringstijd kort gekozen dan kost dit veel testtijd die vaak nog voor niets zal zijn. Bovendien wordt de bemonsteringstijd gevormd door de verwerkingstijd van een stuk van het hoofdprogramma, wordt dit programma later veranderd, dan verandert men ongewild ook de servicetijd voor de randapparaten. Om dit probleem op te lossen is er aan de hardware een ingangssignaal met de naam *interrupt* toegevoegd. Dit interruptsignaal stelt de buitenwereld op elk moment in staat de aandacht van de computer op te eisen. Niet de computer maar de randapparaten kunnen het moment van I/O aktiviteiten aangeven. Bovendien is het na een interrupt zeker dat er minstens één apparaat afhandeling nodig heeft.
Om snel op een interrupt te kunnen reageren zorgt de hardware dat parallel aan de verwerking van de instruktie ook het interruptsignaal getest wordt. (dit kost dus geen extra tijd). Is het interruptsignaal aktief, dan zal na beëidiging van een instruktie niet de volgende instruktie, maar zal een soort subroutine sprong naar een vast interruptadres worden uitgevoerd. Om na afhandeling van de interrupt weer terug te kunnen keren naar het hoofdprogramma zal net als bij een normale subroutine het terugkeeradres op de stack worden opgeborgen. Op het vaste interruptadres zal dan de eerste instruktie van de interrupt-afhandelingsroutine te vinden zijn. De programmeur is vaak zelf verantwoordelijk voor het opbergen van registerinhouden uit het hoofdprogramma. Om tijdens deze akties niet meteen weer te worden overvallen door een nieuwe interrupt van een ander apparaat, wordt bij het uitvoeren van een "interruptsprong" de interruptmogelijkheid afgezet. Als na afhandeling van de interrupt, met behulp van een speciale terugkeerinstruktie, naar het hoofdprogramma wordt teruggekeerd, zal de interruptmogelijkheid weer worden aangezet. Bij de 80C51 is dat de RETI (RETurn from Interrupt) instruktie.
Overeenkomstig de mogelijkheid van nesting bij subroutines is ook hier interrupt op interrupt mogelijk, maar niet altijd wenselijk. Niet elk randapparaat is qua timing even kritisch. Het is prettig om een prioriteitsregeling in te kunnen aanbrengen. Dit kan softwarematig door na het vaststellen van de interruptbron deze voorlopig te negeren als de prioriteit niet hoog genoeg is. Maar dit komt overeen met de situatie dat iemand u de vraag stelt "mag ik u even storen", en dit door het stellen

van de vraag maar meteen te doen zonder het antwoord af te wachten. Bij een goede prioriteitsregeling zal hardware ervoor zorgen dat een aanvraag van te lage prioriteit de interruptingang niet kan aktiveren, en de computer dus niet kan storen.
Er zullen hier drie methoden worden beschouwd:

- Het maskeren van de interrupt(s)
- Gebruik van een daisy chain
- Gebruik van een priority encoder



Afbeelding 157 Interrupt maskering

In afbeelding 157 is aangegeven hoe hardwarematig de interruptrequest kan worden geblokkeerd. Van 4 randapparaten kan de interrupt selectief worden afgezet, en ook de totale interrupt kan worden uitgeschakeld. Als de interrupt wel wordt toegelaten wordt met de interruptacknowledge de afzonderlijke lijnen op de bus gezet om de computer de gelegenheid te geven softwarematig te bepalen welk apparaat de interrupt heeft veroorzaakt. Dit achtereenvolgens programmatisch testen van de interruptlijnen kan het eerste stuk van interruptserviceroutine zijn.
Bij de 80C51 zijn voor de interruptafhandeling twee registers aanwezig zoals aangegeven in afbeelding 158

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	x	x	ES	ET1	EX1	ET0	EX0

Interrupt Enable Register (IE) = A8

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
x	x	x	PS	PT1	PX1	PT0	PX0

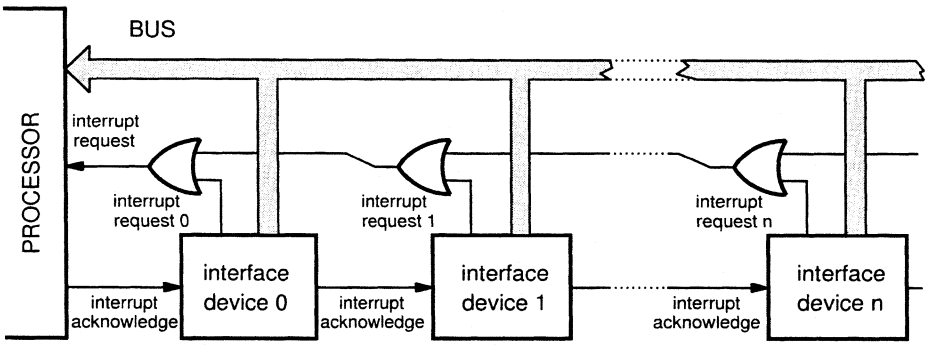
Interrupt Priority Register (IP) = B8

Afbeelding 158 Prioriteit bij de 80C51

De 80C51 heeft 3 ingebouwde randapparaten nl 2 counters en een UART. Samen met de 2 externe interrupts geeft dit 5 mogelijke interruptbronnen. Er zijn 2 prioriteitslevels ingebouwd. Door een 1 in het prioriteitsregister te zetten krijgt het overeenkomstige apparaat hoge prioriteit.
Uitgesplitst naar bits is de betekenis dan:

```
A8.0  Enable externe int.0        B8.0  Hoge prioriteit externe int.0
A8.1  Enable int. timer0          B8.1  Hoge prioriteit timer0
A8.2  Enable externe int.1        B8.2  Hoge prioriteit externe int.1
A8.3  Enable int. timer1          B8.3  Hoge prioriteit timer1
A8.4  Enable int. UART            B8.4  Hoge prioriteit UART
A8.5        -                     B8.5        -
A8.6        -                     B8.6        -
A8.7  Enable de interrupt         B8.7        -
```

Een andere vorm van prioriteit wordt o.a gebruikt door de firma ZILOG bij de Z80 microprocessor. De prioriteit wordt bepaald door de plaats in een keten van in serie geschakelde interfaces. In afbeelding 159 is deze zogenaamde *daisy chain* (ketting van madeliefjes) getekend.
Als een interface een interrupt heeft gegeven via interruptrequestlijn, zal de computer dit beantwoorden met een interruptacknowledge signaal. Een interface dat geen interrupt heeft gegeven zal dit signaal naar zijn buurman doorgeven, maar een interface dat wel een interrupt heeft gegeven zal dit signaal als bevestiging van zijn eigen interrupt zien.

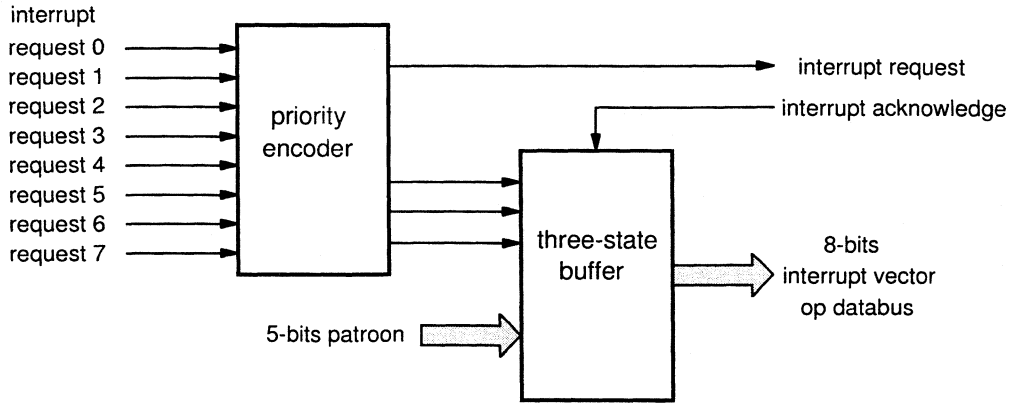


Afbeelding 159 Daisy chain

Het device dat in de keten het dichtst bij de computer staat zal zo de hoogste prioriteit hebben. Via de bus kan zo'n interface zien dat een interrupt afgehandeld is. (De uitvoering van de RETI instruktie). Daarna wordt de interruptrequest weer doorgeschakeld.

Een ander voorbeeld is gegeven in afbeelding 160. Hier is gebruik gemaakt van een zogenaamde *priority encoder.*

Dit i.c. met de Texas Instruments code **sn74148** geeft op één uitgang aan of één der (genummerde) ingangen is geaktiveerd. Aktief betekent bij dit i.c. "0" maken. Op drie andere uitgangen komt het hoogste nummer van de geaktiveerde ingangen te staan.



Afbeelding 160 Priority encoder

Dit uit 3 bits bestaande getal vormt samen met een vast 5-bits patroon een nieuw 8-bits getal dat doorgegeven wordt naar de computer. Dit getal wordt direct als sprongadresadres gebruikt bij het optreden van een interrupt. In de interruptafhandelings routine wordt nu niet meer softwarematig getest welk randapparaat de interrupt veroorzaakte, maar er wordt meteen gesprongen naar het adres waar de afhandelingsroutine van het betreffende randapparaat staat. Deze methode om samen met de interrupt ook meteen een adres te leveren dat bij het interrumperende device hoort, wordt *vectored interrupt* genoemd.
Deze methode wordt door de 80C51 gebruikt. Bij elk der 5 mogelijke interruptbronnen hoort een vast startadres voor afhandeling nl:

externe interrupt0	0003H	00	000	011
timer0	000BH	00	001	011
externe interrupt1	0013H	00	010	011
timer1	001BH	00	011	011
UART	0023H	00	100	011

In het lijstje met de laatste 8 adresbits is te zien dat voor elke interrupt 8 bytes in het geheugen gereserveerd zijn. Meestal zal hier dus ook een spronginstruktie naar de rest van de interruptroutine staan. Wordt er in het systeem geen interrupt toegepast, dan kan het geheugengebied gewoon voor het hoofdprogramma gebruikt worden.

11.6.3 DMA

In het voorgaande is, door toevoeging van hardware, de software die nodig is voor de I/O van een datawoord steeds minder geworden. Maar nog steeds is voor elk datawoord dat wordt in- of uitgevoerd een klein stukje programma nodig. Om de computer nog minder te belasten is de volgende stap om niet per datawoord maar met een heel blok datawoorden te werken. Met één aktie van de computer wordt een heel blok data naar het geheugen geschreven of uit het geheugen gelezen.
Omdat dit zoveel mogelijk zonder hulp van de computer gebeurt heet deze vorm van I/O *Direct Memory Access* of kortweg DMA. Om DMA te kunnen toepassen is extra hardware nodig die zelfstandig het geheugen kan lezen en beschrijven. Deze hardware, *DMA controller* genaamd,

bestaat o.a. uit een adresregister, een wordcountregister en de hardware die nodig is voor de afhandeling van de timingssignalen.
Er kunnen 3 soorten van DMA worden onderscheiden nl:

- Burst DMA
- Cycle-stealing DMA
- Transparent DMA

Burst DMA

Bij burst DMA wordt een blok woorden met maximale snelheid van of naar het geheugen gebracht, terwijl de computer in rust verkeert. Deze maximale transportsnelheid wordt bepaald door de maximale snelheid van het geheugen, de snelheid van de DMA controller en de snelheid van het randapparaat.
De uitvoering gebeurt in een aantal stappen:

1. De computer bereidt het DMA tansport voor door het beginadres en de wordcount naar de DMA controller te schrijven.

2. Het randapparaat start de I/O aktie door via de controller een *DMArequest* naar de computer te zenden

3. De computer maakt zijn instruktie af, maakt de bussen vrij en zendt als laatste aktie een *DMAacknowledge* naar de controller.

4. De controller neemt de besturing over en verzorgt de signalen voor het lezen of schrijven naar het geheugen. Het adresregister wordt hierbij telkens verhoogd en het wordcountregister verlaagd, totdat dit op nul staat, en het opgegeven aantal woorden is getransporteerd.

5. De controller stuurt een interrupt naar de computer als sein voor de overdracht van de besturing, en de computer gaat weer verder met het onderbroken hoofdprogramma.

Burst DMA wordt vaak toegepast bij het transport van blokken data van band en schijfgeheugens, en voor het verversen van beeldschermgeheugens.

Bij processoren die het refreshen van het dynamische RAM geheugens verzorgen, zijn er wat beperkingen. Immers als de processor zich tijdens het DMA transport niet meer met het geheugen bemoeit wordt ook het geheugen niet meer gerefreshed. Of de controller zal ook het refreshen voor zijn rekening moeten nemen, of de burst-DMA mag maximaal de tijd tussen twee refreshcycli duren.

Cycle stealing DMA

Bij cycle-stealing DMA wordt de processor niet stilgelegd, maar vindt het transport plaats tussen de uitvoering van de instrukties door. Er worden voor de DMA cycli ingelast, of men kan ook zeggen er worden cycli van de computer gestolen. De cycle-stealing DMA duurt dus langer dan een burst DMA.

1. De computer schrijft waarden naar het adresregister en het worddcountregister.

2. Het randapparaat zendt via de controller een *DMArequest* naar de computer.

3. De computer maakt zijn instruktie af, maakt de bussen vrij en zendt een *DMAacknowledge* naar de controller.

4. De controller leest of schrijft één woord naar het geheugen, verhoogt het adresregister en verlaagt het wordcountregister en test op 0 daarvan. Als het wordcountregister niet

0 is ga terug naar stap 2, anders ga door met 5.

5. De controller meldt zich via een interrupt af bij de computer.

Transparent DMA

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

Omdat een DMA controller complex en relatief duur is, is het vaak beter om i.p.v een DMA controller een extra processor als I/O processor aan een computersysteem toe te voegen.

11.7 Interface voorbeelden 80C51

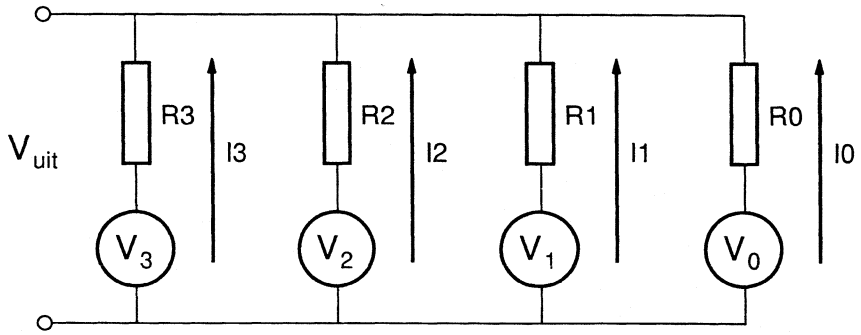
Een 3-tal I/O devices, 2 timers en een UART, zijn al in de 80C51 ingebouwd. De status-, command- en dataregisters zijn te bereiken als plaatsen in het interne RAMgeheugen. Zij liggen in het gebied van de Special Function Registers in het adresbereik van 0080H t/m 00FFH.
Voor de zelf aan te sluiten hardware zijn de poorten P0, P1, P2 en P3 te gebruiken voor zover zij nog niet in gebruik zijn voor andere doeleinden. (adresbus, databus, interrupt enz).
Komt men nog aansluitmogelijkheden te kort, dan is het mogelijk om de registers van de te interfacen devices aan te sluiten als extern RAM geheugen (memory mapped I/O). Als het mogelijke externe RAMbereik van 64K, over extern RAM en externe I/O verdeeld wordt, dan zal de gebruiker moeten zorgen voor de noodzakelijke adresuitcoderingen. Verder moet de timing van de randapparatuur voldoen aan de timing voor data, adres, read en write die de 80C51 voor extern RAM gebruikt. Alleen de instrukties MOVX A,@DPTR en MOVX @DPTR,A voor een groot RAM bereik (> 256), of MOVX A,@Ri en MOVX @Ri,A met i=0,1 met een extern bereik tot 256, zijn dan voor de command- , status- en dataoverdracht te gebruiken.

11.7.1 D/A en A/D omzetting

Als een binaire output moet worden omgezet naar een analoge waarde, dan kan dat met een weerstandsnetwerk. In afbeelding 161 is zo'n netwerk voor 4 bits getekend. Er moet gelden:

$$I0 + I1 + I2 + I3 = 0 \qquad \text{of}$$

$$\frac{V_0 - V_{uit}}{R0} + \frac{V_1 - V_{uit}}{R1} + \frac{V_2 - V_{uit}}{R2} + \frac{V_3 - V_{uit}}{R3} = 0$$



Afbeelding 161 gewogen weerstanden

Hieruit volgt de relatie:

$$V_{uit} = \frac{\frac{V_0}{R0} + \frac{V_1}{R1} + \frac{V_2}{R2} + \frac{V_3}{R3}}{\frac{1}{R0} + \frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}}$$

Wordt voor de weerstandswaarden gekozen: R0 = 8R, R1 = 4R, R2 = 2R en R3 = R dan geldt:

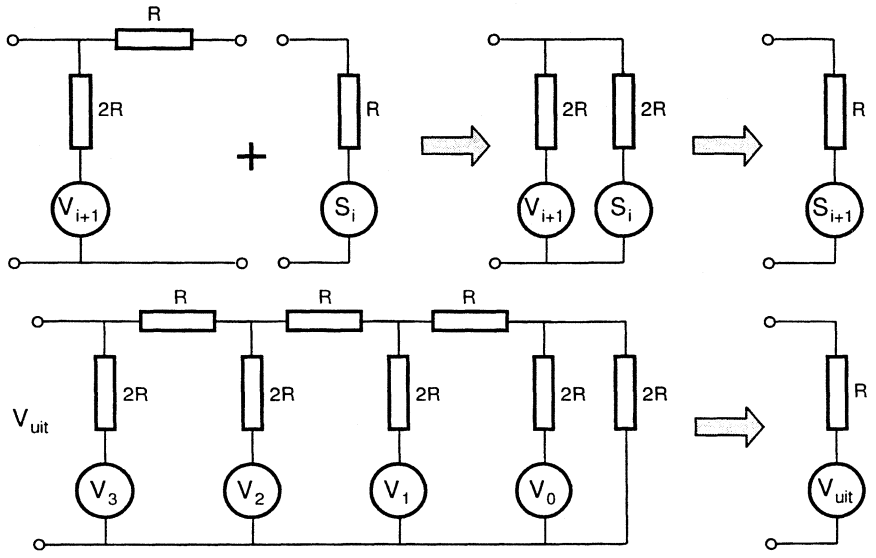
$$V_{uit} = \frac{1}{15} \cdot (8.V_3 + 4.V_2 + 2.V_1 + V_0)$$

De spanningen V_0, V_1, V_2 en V_3 kunnen worden gevormd door de bits b_0 , b_1 , b_2 en b_3 van het binaire getal een spanning tussen 0 en $V_{digitale1}$ te laten schakelen. (dus gewoon de uitgangen van een digitaal blokje). Dus in formule: $V_0 = b_0.V_{digitale1}$, $V_1 = b_1.V_{digitale1}$, $V_2 = b_2.V_{digitale1}$, $V_3 = b_3.V_{digitale1}$

Dan levert dit op:

$$V_{uit} = \frac{b_3 b_2 b_1 b_0}{15} \cdot V_{digitale1}$$

V_{uit} is dus evenredig met de waarde van het getal dat gevormd wordt door de bits b_i .
Een nadeel van het weerstandsnetwerk is dat alle benodigde weerstanden een andere waarde hebben en qua grootte ver uit elkaar liggen. Wil men het weerstandsnetwerk uitbreiden tot een omzetter van getallen van 12 bits, dan zal de grootste weerstand 2048 maal zo groot moeten zijn als de kleinste weerstand. Bij een maximale fout van 1 eenheid mag deze grootste weerstand minder dan 0.05% afwijken.
Meestal wordt daarom een ander netwerk toegepast dat is getekend in afbeelding 162.



Afbeelding 162 laddernetwerk

Als een spanningsbron S_i wordt uitgebreid met 2 weerstanden en een spanningsbron V_{i+1} dan wordt een nieuwe spanningsbron gevormd met een spanning S_{i+1} waarvoor geldt $S_{i+1} = 1/2.(S_i + V_{i+1})$.
Als gestart wordt met $S_0 = 1/2.V_0$ dan levert het toevoegen van 3 spanningsbronnen met bijhorende weerstanden: $V_{uit} = S_3$ of uitgeschreven:

$$V_{uit} = \frac{1}{2} \cdot \left(\frac{1}{2} \cdot \left(\frac{1}{2} \cdot \left(\frac{1}{2} \cdot V_0 + V_1 \right) + V_2 \right) + V_3 \right) = \frac{1}{16} \cdot (8 \cdot V_3 + 4 \cdot V_2 + 2 \cdot V_1 + V_0)$$

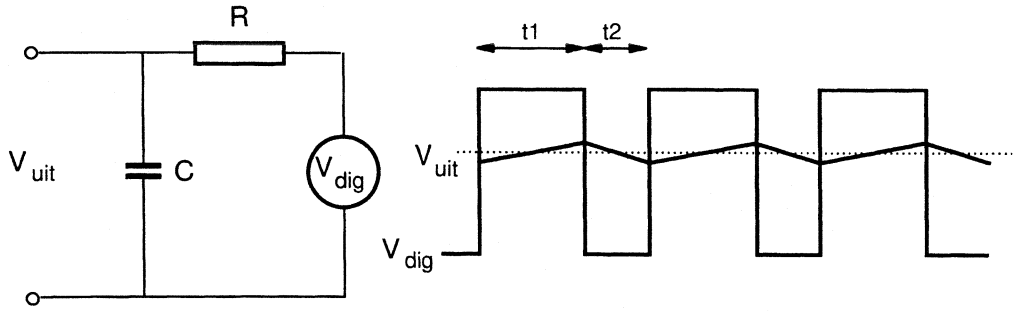
Als ook hier de spanningen gevormd worden door de bits een spanningsbron tussen 0 en $V_{digitale1}$ te laten schakelen, dan geldt:

$$V_{uit} = \frac{b_3 b_2 b_1 b_0}{16} \cdot V_{digitale1}$$

Het voordeel van deze methode is dat slechts weerstanden R en 2R gebruikt worden. Gelijke weerstanden, die in dezelfde produktieslag gemaakt zijn, vertonen minder spreiding dan diegene die op verschillende tijdstippen gemaakt zijn. Deze D/A omzetters zijn in IC vorm verkrijgbaar.

Een methode van D/A omzetting die niet van een weerstandsnetwerk uitgaat is geschetst in afbeelding 163. De digitale bron V_{dig} wordt aan en uitgeschakeld met de tijden t1 en t2. De aldus verkregen pulsen met een geregelde breedte worden door een RC combinatie "geintegreerd". Zo wordt de gemiddelde spanning

$$V_{uit} = \frac{t1}{t1+t2} \cdot V_{dig}$$



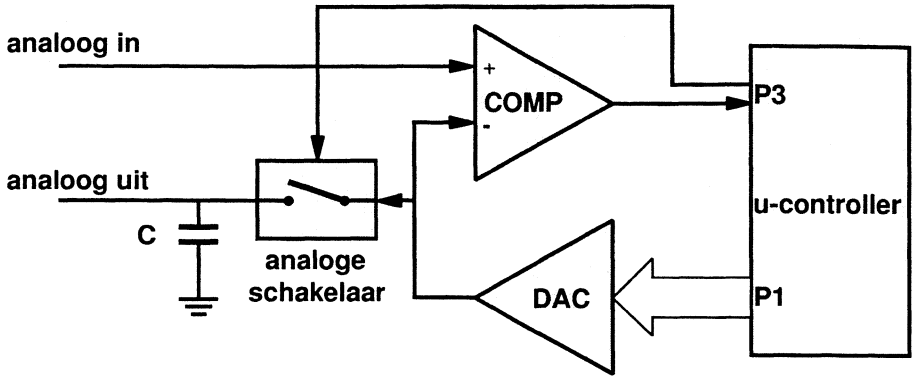
Afbeelding 163 pulsbreedte sturing

De tijdconstante van de RC combinatie moet groot zijn t.o.v t1 en t2, om geen last te hebben van het schakelen, maar moet klein zijn t.o.v de veranderingen in het signaal, om de veranderingen in het om te zetten signaal wel snel genoeg te kunnen volgen.
De uitgang van de schakeling moet voor de meeste toepassingen nog gebufferd worden. Als de uitgangsspanning bijvoorbeeld gebruikt wordt om het toerental van een motor te regelen, moet een flinke stroomversterking plaats vinden.

Een veel gebruikte methode om A/D omzetting uit te voeren is de methode van de *successive approximation*. Hierbij wordt de uitgangswaarde van een D/A omzetter vergeleken met de te converteren spanning. Afhankelijk van het resultaat van de vergelijking wordt de binaire waarde aangepast. A/D omzetters die op dit principe berusten zijn ook in chipvorm te verkrijgen.
Stel het bereik van de te converteren spanning ligt in het interval [0 ,255]. De eerste vergelijking vindt dan plaats met 128 om te kunnen bepalen of de analoge waarde in het interval [0, 127] of [128, 255] ligt. Afhankelijk van de uitslag van de test zal de volgende waarde waarmee vergeleken wordt 128+64=192 of 128-64=64 zijn. Na de tweede vergelijking is dus vastgesteld of de waarde in [0, 63] of [64, 127] of [128, 191] of [192, 255] ligt. De volgende testwaarde wordt telkens in het midden van het vastgestelde interval gelegd. In de laatste bewerkingsslag wordt de waarde bepaald door afhankelijk van de laatste test, de correctie 0 0f -1 aan te brengen.
Voor de eindwaarde geldt dus:
$V = 128 \pm 64 \pm 32 \pm 16 \pm 8 \pm 4 \pm 2 \pm 1 + \begin{matrix} 0 \\ -1 \end{matrix}$ waarbij + of - gekozen wordt op grond van de uitslag van de test.
De benaming successive approximation geeft dus aan dat elke test het mogelijke interval waarin de te converteren waarde ligt met de helft reduceert. De binaire testpatronen zien er als volgt uit:

			00010000	
			00110000	
		00100000	01010000	
10000000	01000000	01100000	01110000	enz.
	11000000	10100000	10010000	
		11100000	10110000	
			11010000	
			11110000	

Hier is als voorbeeld de 80C51 gebruikt om A/D omzetting volgens dit principe softwarematig uit te voeren.
In afbeelding 164 is een analoge ingang en een analoge uitgang op de 80C51 gemaakt met de hardwaretoevoeging van een D/A converter, een analoge schakelaar en een comparator (vergelijker).



Afbeelding 164 combinatie van A/D en D/A

Voor de uitvoer van een digitale waarde wordt poort P1 gebruikt. De uitgang van de comparator wordt ingelezen op P3.1 en de schakelaar wordt bediend door P3.0
De analoge waarde kan lopen van 0 tot 255. De eerste waarde waarmee vergeleken wordt ligt in het midden van het interval, en is dus 128. Het programma kan erdan als volgt uitzien:

```
P1      EQU     90H       ; data uit
P3      EQU    0B0H       ; besturing
DATA1   EQU     40H       ; analoog in
DATA2   EQU     41H       ; analoog uit
        ......
        ......
ANAIN   MOV    A,#80H     ; eerste test is 128
        MOV    DATA1,A    ; macht van 2
        CLR    C          ; carry 0 maken
LUS     JC     KLAAR      ; is carry 1 ?
        MOV    P1,A       ; test starten
        XCH    A,DATA1    ; 2-macht naar A
        RRC    A          ; door 2 delen
        XCH    A,DATA1    ; terugzetten
        JNB    P3.1,MIN   ; analoog < test ?
        ADD    A,DATA1    ; next test groter
        SJMP   LUS        ; volgende bit
MIN     SUBB   A,DATA1    ; next test kleiner
        SJMP   LUS        ; volgende bit
KLAAR   MOV    DATA1,A    ; waarde opbergen
```

De macht van 2 wordt elke rondgang van de lus opgeteld of afgetrokken van het testpatroon.
De laatste keer moet het echter +0 of -1 zijn. Dit gaat hier automatisch, doordat in de laatste slag de macht van 2, 0 is met een 1 in de carry. De ADD instruktie zal dus +0 uitvoeren, maar de SUBB, zal met SUBtract with **Borrow** als bewerking -1 geven.
De uitvoer van een analoge waarde vind via de DAC plaats. Wel moet de schakelaar even gesloten worden. Als 1 instruktietijd voor de schakelaar te kort is kunnen extra instrukties tussen de instruktie voor het sluiten en de instruktie voor het openen van de schakelaar worden ingevoegd. De condensator doet dienst als analoog geheugen. Dit gaat alleen goed als de uitgang niet belast wordt, en dus bijvoorbeeld met een operationele versterker (OP AMP) wordt gebufferd. Het programmastuk voor analoge output:

```
          ......
          ......
ANAUIT    MOV   P1,DATA2  ; analoge spanning maken
          SETB  P3.0      ; in condensator zetten
          CLR   P3.0      ; verbinding verbreken
```

12 Operating System

12.1 Inleiding

Een Operating System is een **programma** dat op bijna elke computer te vinden is, en twee bedoelingen heeft:

- Het verhogen van het gebruiksgemak van de gebruiker.
- Het optimaliseren van de efficiëntie van de hardware van de computer(s).

Als voorbeeld van het eerste punt is de grafische interface van een Windowssysteem. Geen moeilijk te onthouden commando's, maar het aanklikken van een symbooltje is voldoende om het gewenste effect te krijgen. Maar ook bij het aanzetten van de computer heeft het OS er al voor gezorgd dat er met het systeem te werken valt. Hier is overigens al meteen het eerste kip en ei probleem. Het OS is een programma dat te groot is om in vast geheugen op te slaan, en moet daarom ingelezen worden vanaf een achtergrondgeheugen. Maar het lezen van een achtergrondgeheugen is een taak van het OS dat nog gelezen moet worden. De oplossing is een klein *bootstrap* programma in (E(P(ROM))) geheugen, dat in staat is een vast aantal bytes, van een vaste plaats op achtergrondgeheugen, in het werkgeheugen te laden en te starten. Dit bootstrapprogramma bevindt zich meestal op adres 0 in het geheugen. Als de computer aangezet wordt, of er op **reset** gedrukt wordt, zal het bootstrapprogramma gestart worden, een klein stuk van het OS ophalen, dat op zijn beurt het hele OS kan lezen en starten.
Een voorbeeld van efficiëntieverhoging is het doorwerken van de programma's tijdens de verschillende wachttijden die optreden bij de besturing van randapparaten. Het OS moet ervoor zorgen dat de CPU voor een zo hoog mogelijk percentage bezig is met "nuttige" programma's.

Niet alle toepassingen zullen hetzelfde type OS gebruiken. Een grove indeling in typen OS systemen geeft:

- Job-shop systems
  - Batch systems of single user systems
  - Multiprogrammed batch systems
  - Time-Sharing of multitasking systems (multi user systems)

- Real-time systems
  - Hard real-time systems
  - Soft real-time systems

Bij *Batchsystemen* worden de opdrachten netjes in volorde van binnenkomst afgewerkt. De computer is, uitgezonderd de IO, slechts met de uitvoering van één job (die nog uit verschillende stappen kan bestaan) bezig. Dit is een nabootsing van de oude situatie met een operateur. De gebruiker moest bij zijn programma's een aantal commando's, geschreven in JOB-controle language, meeleveren. Bij uitvoering van de job kreeg de operateur, via een printer of beeldscherm, opdrachten als bijvoorbeeld het inlezen van de Fortrancompiler of het opspannen van een bepaalde tape met data, waarna de job weer verder kon worden verwerkt. De jobcontrole is van ponskaarten naar disk verhuisd, en de disk, dus het achtergrondgeheugen is zonder hulp van de operateur te bereiken. De afhandeling is daarom sneller en de operateur is er voor het (re)configureren van systemen en voor noodgevallen (storingen) zoals het vervangen van de tonercassette van een laserprinter.
In de geautomatiseerde versie kan het voorkomen dat bij het wachten op IO de in behandeling zijnde job ook niet verder kan, en dus de CPU werkeloos is. Dan krijgt men te maken met systemen die men *multiprogrammed batch systems* noemt. Een aantal jobs moet nu in het geheugen klaar staan om verwerkt te kunnen worden. Kan een bepaalde, in uitvoering zijnde, job niet verder, dan

wordt een andere job vast in behandeling genomen.
Toen het terminalgebruik zijn intrede deed, en gebruikers binnen redelijke tijd een reaktie op een bijvoorbeeld grafische input verlangden moest de tijd die de computer aan een proces besteedde worden teruggebracht. Jobs worden opgedeeld in kleinere eenheden, de processen, terwijl deze processen niet alleen bij blokkering maar ook na het verstrijken van een bepaalde tijd door het OS worden afgetikt om plaats te maken voor een ander proces. Hier krijgen we te maken met *Time-sharing of Multi-tasking systems.* Iedere gebruiker krijgt de indruk dat hij de hele computer voor zichzelf heeft en niet deelt met bijvoorbeeld 299 andere terminalgebruikers. Nadat de processor een bepaalde tijdsduur of *timeslice* aan het proces gewerkt heeft geeft een timer een interrupt, en wordt er weer van proces gewisseld.
Het is duidelijk dat dit soort besturingen niet door de gebruikers, maar door een centraal besturingsprogramma, het Operating System, moet worden uitgevoerd.
Een ander type OS is het *real-time OS.* Hierbij wil de gebruiker dat Input en Output op vaste, van te voren bepaalde momenten, aktief worden. Hierbij is het dus nodig dat van elke aktie bekend is wat de maximale verwerkingstijd kan zijn. Men kan nog onderscheid maken tussen *hard real-time systems* en *soft real-time systems.*
Bij *hard real-time systems* zijn de tijdstippen van akties bindend. Meestal treedt dit op in procescontrol. Als de computer de terugkoppeling van een aantal regelprocessen op zich neemt, dan kan dit alleen als de garantie bestaat dat de door de regelsystemen gedefiniëerde tijdkonstanten door de computer niet overschreden worden, ook niet bij de meest ongelukkige samenloop van omstandigheden. Immers de stabiliteit van de regelprocessen kan hiervan afhangen.
Bij *soft realtime systems* zijn de timings eisen minder rigoureus, maar nog steeds van groot belang. Een voorbeeld is het gebruik van de computer bij Multi media toepassingen. Als een plaatje door de computer van een bijbehorend geluid wordt voorzien, zal het startmoment van het geluid niet zo tijdkritisch zijn, maar de frequenties waaruit het geluidssignaal is opgebouwd mogen niet te veel afwijken. Ook bij het genereren van bewegende beelden is een zekere constantheid in snelheid een noodzaak.

Operating Systems voor real-time toepassingen zijn sterk afhankelijk van de soort toepassing en lenen zich minder voor een algemene beschouwing. Daarom wordt voor het vervolg een simpele versie van een multitasking systeem bekeken.

Functies die in een OS terug te vinden zijn:

- **Gebruikersinterface:** Het uitvoeren van de voor het OS zelf bestemde opdrachten. Dit kan d.m.v ingetypte opdrachten, maar ook d.m.v met een muis "aangeklikte" commando's.
  Het geven van informatie over het stadium van afhandeling van een opdracht. Ook dit kan tekstueel of grafisch plaatsvinden.
  Onder gebruiker moet men hier zowel de eindgebruiker als de systeembeheerder (operator) verstaan.

- **Foutafhandeling:** Het afwikkelen en melden van opgetreden foutcondities.(Bijvoorbeeld het niet executeren van een programma waarvan de compilatie niet foutloos verliep, of het afbreken van een programma dat door 0 probeert te delen.). Ook detekteerbare fouten van de hardware, mits niet te ingrijpend, kunnen door het OS gemeld, en soms verholpen worden.

- **Protectie:** Het beschermen van programma's tegen elkaar en tegen zichzelf, en met name het beschermen van het OS zelf.

- **Boekhouding:** Het registreren van de gebruikte tijd per proces, en de gebruikte middelen. Dit kan gebruikt worden voor het in rekening brengen van de kosten voor de klant, maar ook voor het opsporen van systeemfouten, knelpunten en misbruik.

- **Resource control:** Het voeren van het beheer over de middelen: randapparatuur, hoofdgeheugen-ruimte, en processortijd.

- **Scheduling:** Het bepalen van de volgorde waarin de aangeboden programma's verwerkt zullen worden.
- **Input/Output:** Het verzorgen van de communicatie met de randapparatuur.
- **Netwerkbeheer:** Het bieden van netwerkfaciliteiten naar andere computersystemen.
- **Interrupt:** Het opvangen en doorgeven van de interruptmeldingen.
- **Time sharing:** Het verdelen van de beschikbare processortijd over de in behandeling zijnde processen.

Bij een timesharing proces verdeelt de computer de beschikbare processortijd over de af te handelen processen, waarbij een proces de aktie is die ontstaat door een programma te starten. Een programma is een lijstje opdrachten en een proces de uitvoering van deze opdrachten. Processen worden schijnbaar tegelijkertijd uitgevoerd, maar in werkelijkheid wordt er, repeterend, een kort moment aan elk proces gewerkt. Om deze taak voor het OS mogelijk te maken zijn er nog een tweetal hardware faciliteiten nodig die tot nog toe onbesproken zijn.

Allereerst de *real-time clock*, een teller waarmee een bepaalde tijd, *timeslice*, kan worden afgemeten, en na het verstrijken van een vaste tijd een volgend proces aan de beurt komt voor verwerking. Deze real-time clock is niets anders dan het randapparaat timer, maar nu specifiek gebruikt door het OS.

Ten tweede hebben processoren (ook de grote µ-processoren) twee toestanden. Zij kunnen in *supervisor of systemmode* of in *usermode* werken. In welke mode de processor werkt wordt bepaald door een enkel bit in één der hardwareregisters. In systemmode kan de gehele instruktieset van de computer gebruikt worden, maar in usermode zal een beperkt deel der instrukties niet worden uitgevoerd, maar resulteren in een *trap*, een soort interrupt maar nu niet van buiten maar direct vanuit de hardware van de computer. (Zo'n trap komt bijvoorbeeld ook voor als een hardwaredeling door 0 wordt uitgevoerd). Hiermee kan het OS processen tegen elkaar en zichzelf beschermen.
De instrukties die voor de gebruiker kunnen worden afgeschermd zijn de instrukties voor:

- het aan- of uitzetten van interrupts
- het veranderen van proces
- het beschrijven van registers voor het memory management
- de input of output
- het stoppen van de processor

Het is duidelijk waarom gebruikers deze instrukties niet op eigen houtje mogen laten uitvoeren (vooral de laatste niet). Als (het proces van) een gebruiker toch een aktie wenst die door deze afgeschermde instrukties moet worden uitgevoerd, dan kan hij zich met een *supervisor- of systemcall* tot het OS wenden, en laten uitvoeren. Een systemcall is een soort subroutineaanroep naar een vast adres. Als ervoor gezorgd wordt dat op de plaatsen waar de systemcalls naar verwijzen slechts "veilige" OS-software kan staan, is het systeem beschermd. Het betekent wel dat de gebruikersprocessen slechts met systemcalls met het OS kunnen communiceren.
Om van systemmode naar usermode over te gaan kan het OS d.m.v een afgeschermde instruktie het bit in een hardwareregister omzetten. De omzetting van usermode naar systemmode gebeurt automatisch bij het optreden van een interrupt of van een trap (interrupt van de hardware) of de uitvoering van een systemcall. Al deze akties verwijzen naar vaste adressen waar OS-software staat.

Een middel om processen te kunnen synchroniseren is het gebruik van semaforen (seinpalen). Deze semaforen zijn voor het eerst in 1965 geïntroduceerd door de Nederlander Dijkstra. Zij worden

toegepast om de toegang te regelen tot een stuk software dat slechts één gebruiker tegelijkertijd kan toelaten. Het is te vergelijken met de situatie op een treinemplacement. De toegang van een aantal sporen tot een stuk enkelspoor wordt bewaakt met een seinpaal. De seinpaal geeft met een getal S de situatie aan. Bij S=0 staat het sein op onveilig en moet er gestopt worden. Bij S>0 kan er doorgereden worden maar moet S:=S-1 worden uitgevoerd. Als de trein het enkelspoortraject verlaat en dus zijn plaats weer vrijgeeft wordt op de seinpaal S:=S+1 uitgevoerd. Als er maar één trein tegelijk op het enkelspoor mag komen zal voor de beginwaarde van de semafoor dus 1 moeten worden gekozen.
Er zit een gevaar aan deze regeling, nl als twee treinen tegelijk komen aanrijden zullen zij beide het sein op veilig zien staan en niet gaan remmen. Dit kan worden opgelost door het testen en het onveilig zetten van het sein tot één handeling te maken, d.w.z. tussen het testen en op onveilig zetten een ander niet de mogelijkheid tot testen geven.

Er zijn nu 2 calls WAIT(S) en SIGNAL(S) ingevoerd, met:

WAIT(S): als S=0 wacht
als S>0 ga door en S:=S-1

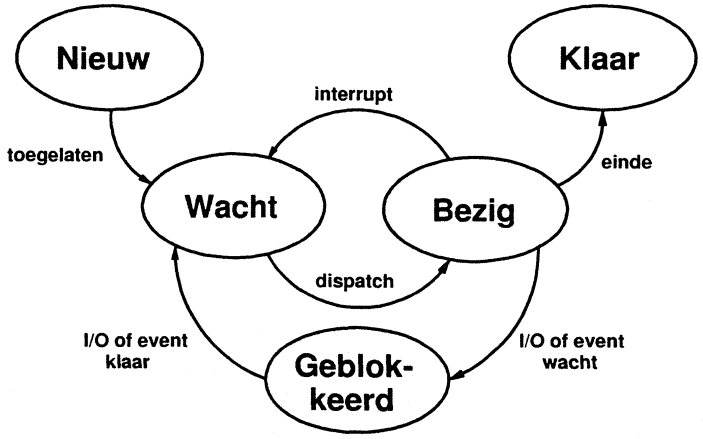
SIGNAL(S): S:=S+1

12.2 Kernal

De kernal of kern van een OS bestaat uit 3 delen nl:

- De first level interrupt handler
- De afhandeling van WAIT en SIGNAL
- De dispatcher

In de kernal wordt het wisselen van proces uitgevoerd. In afbeelding 165 is aangegeven in welke situatie of toestand een proces zich kan bevinden.

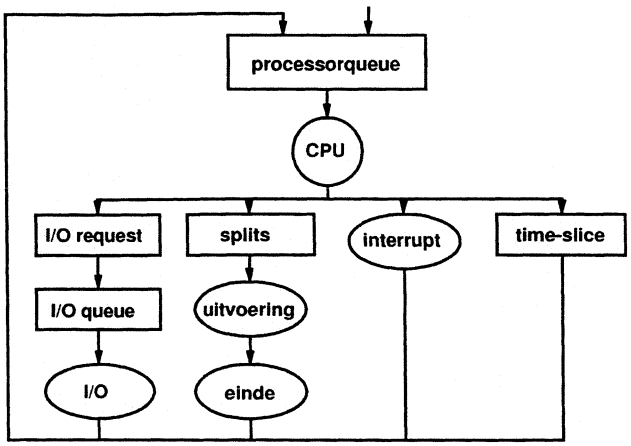


Afbeelding 165 proces toestanden

Slechts één proces kan bezig zijn. Als er een bepaalde tijd verstreken is (timer geeft interrupt) of er een andere interrupt optreedt, wordt het proces "bevroren" en in een wachtrij gezet. Deze wachtrij wordt de *processorqueue* genoemd. Bij de volgende beurt van het proces zal de *dispatcher* het proces weer "ontdooien" en verder laten uitvoeren. Als het proces bezig is en er moet Input of Output plaatsvinden, of er moet van een andere faciliteit gebruik gemaakt worden, dan zal het proces worden gestopt en worden afgevoerd naar één of meer wachtrijen aangegeven door de toestand geblokkeerd. Processen die zich in deze toestand bevinden kunnen naar de

processorqueue worden overgebracht als de seinpaal waarop zij wachten op veilig wordt gezet. Een derde mogelijkheid is dat het proces klaar is. In dat geval wordt het proces gewoon verwijderd uit de processorqueue. Een nieuw proces kan gewoon tot de processorqueue worden toegelaten.

In afbeelding 166 is te zien hoe een proces van de toestand "bezig" naar de toestand "wacht" kan komen, en zich dan dus in de processorqueue bevindt.



Afbeelding 166 procesafhandeling

Bij het optreden van een interrupt of een timerinterrupt wordt het proces direct naar de processorqueue teruggezet. Is de reden voor het verlaten van de toestand "bezig" een Input/Output verzoek geweest, dan zal gewacht moeten worden op de voltooiing van de I/O voordat er in de processorqueue mag worden plaatsgenomen. Een andere nog niet genoemde reden om een proces tijdelijk te beëidigen is het het afsplitsen van een nieuw proces. Een proces, dan *parentproces* genoemd, kan een nieuw proces, het *childproces* genereren. In het simpele geval zal het nieuwe childproces dan in behandeling genomen worden, terwijl het parentproces in een wachtrij wordt geplaatst, en moet wachten op het klaar zijn van zijn childproces.

12.2.1 First level interrupthandler

De first level interrupthandler is de programmatuur die de interrupts opvangt en verwerkt. De interrupt kan zowel van een randapparaat buiten de processor als van de computerhardware zelf (trap) komen of van een speciale instruktie die een softwareinterrupt opwekt.

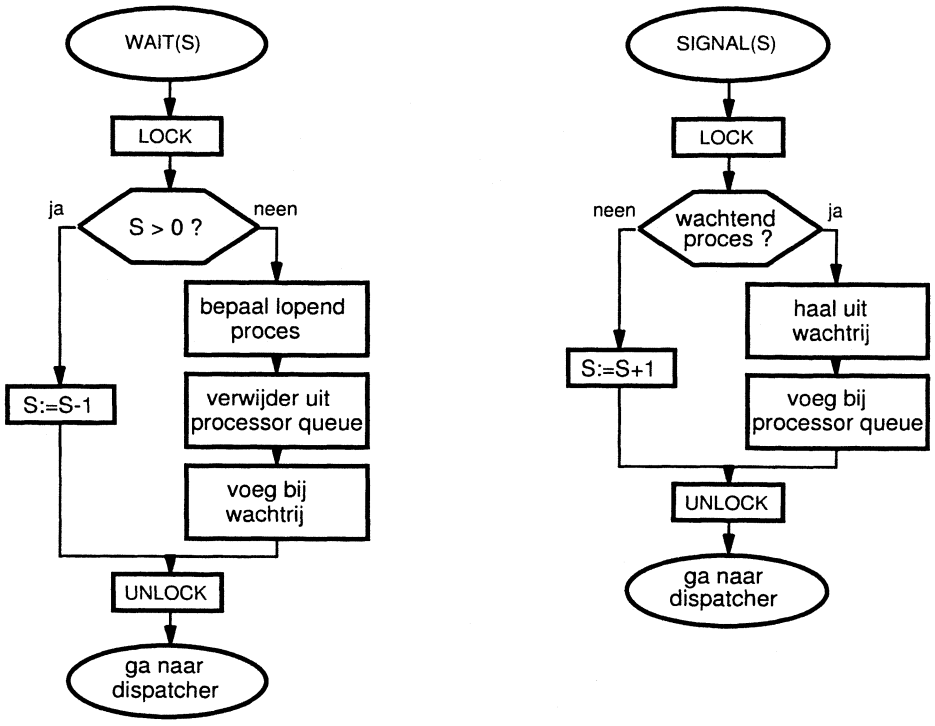
Na het optreden van een interrupt worden een aantal handelingen verricht.

- De interruptmogelijkheid van een volgende interrupt is hardwarematig afgezet.
- De Program Counter wordt opgeborgen.
- De andere noodzakelijke processoregisters worden opgeborgen.
- De bron van de interrupt wordt vastgesteld. Dit kan programmatisch door de lijnen te testen, maar ook hardwarematig als het een "vectored interrupt" betreft.
- Springen naar een bij de interruptbron horende interruptroutine.
- De toestand van een proces wordt aangepast. Dit kan een proces van "bezig" naar "wacht" brengen, maar ook een SIGNAL voor een "geblokkeerd" proces opleveren.
- De interrupt aanzetten.
- Naar de dispatcher springen.

De duur van de interruptafhandeling wordt kritisch als het aantal interrupts per tijdseenheid gaat toenemen. In het ergste geval zullen er interrupts verloren gaan.

12.2.2 WAIT en SIGNAL

WAIT en SIGNAL zijn de boodschappen waarmee de software kan testen of de noodzakelijke middelen voor een proces al aanwezig zijn, of melden dat gebruikte middelen weer worden vrijgegeven. In afbeelding 167 is aangegeven hoe door deze signalen de toestand van een proces kan veranderen.



Afbeelding 167 Afhandeling van WAIT en SIGNAL

De LOCK en UNLOCK akties geven aan dat slechts één proces tegelijk van de routines gebruik mag maken. Als er maar één processor is kan dit worden uitgevoerd door het uitzetten van de interrupt, dit is immers de enige mogelijkheid om in het lopende programma "in te breken".
Als bij een WAIT(S) het gevraagde middel beschikbaar is (dus S>0) dan wordt het bezet verklaard, en toegewezen aan het vragende proces, en het lopende proces blijft gewoon in de "bezig" toestand. Als het gevraagde middel echter niet voor handen is (dus S=0) dan wordt het lopend proces uit de processorqueue gehaald en naar de wachtrij gebracht. Het proces komt in de toestand "geblokkeerd".
Bij een SIGNAL(S) wordt een middel vrijgegeven. Eerst wordt gekeken of een ander proces hierop staat te wachten. Is dit niet het geval dan wordt het middel als beschikbaar bestempeld (dus S:=1), is dit wel het geval dan wordt dit middel aan het andere proces toegekend waarna dit andere proces in de processorqueue wordt geplaatst. Het vrijgegeven middel wordt dus meteen weer bezet. Het lopende proces gaat als "bezig" proces naar de dispatcher.

12.2.3 Dispatcher

De dispatcher is het programmaonderdeel van de kernel dat het lopend proces kan vervangen door een wachtend proces.

Een proces wordt bepaald door de "process descriptor" die uit 3 onderdelen bestaat nl:

Een naam (pointer)
De toestand waarin het proces zich bevindt
De inhouden van de registers van de CPU op het moment van afbreken van het proces

De achtereenvolgende stappen waarin de dispatcher zijn taak uitvoert zijn:

- Testen of het lopend proces het meest geschikt is om verder te gaan. Is dit het geval dan wordt de Program Counter en de usermode hersteld en teruggesprongen naar het lopende programma.
  Bij een "onbelangrijk" interrupt zal het terugkeeradres van de stack gehaald worden om het programma te kunnen voortzetten. Ook na een geslaagde WAIT call en het vrijgeven van een middel via een SIGNAL call kan het lopend programma de beste voortzetting blijken te zijn.Ook dan is het terugkeeradres beschikbaar.
  Als er wel van proces gewisseld moet worden:
- Bevriezen van het lopende proces door de registers van de CPU in de process descriptor op te bergen.
- Het volgende proces uit de processorqueue halen. Met de informatie uit de processdescriptor de registers herstellen. De processor in usermode zetten.
- Met de oude PC inhoud weer naar het "ontdooide" proces springen.

Men kan ervan uitgaan dat van alle processen die in de processorqueue staan het programma gewoon in het RAM geheugen aanwezig is. Als dit niet het geval is, is het proces in de toestand geblokkeerd.

In het voorgaande wordt gesproken over het in- en uit een wachtrij halen maar niet hoe de wachtrij georganiseerd is. Er worden allerlei algoritmes gebruikt om de volgorde in de wachtrij te laten afhangen van de eigenschappen van het in te sorteren proces. Hier wordt gewoon verondersteld dat de eerste in de rij eruit gehaald wordt en bij het in de rij zetten het principe van achteraan aansluiten geldt.

12.3 Geheugensystemen

12.3.1 Computergeheugen als hierarchie

Tot dusverre hebben we een computersysteem beschouwd als een stelsel van afzonderlijke technishe componenten, zoals CPU, main memory, i/o processoren, disks, tapes, andere randapparatuur, enz. Een wat meer algemene beschouwingswijze kan zijn: een **proces** , zijn aktiviteiten verrichtend met behulp van een **processor**, daarbij instrukties en data verwerkend die verblijven in een algemeen **storage**-module, waarin alle vormen van gegevensopslag zijn verenigd.

We kunnen de verschillende vormen van "storage" als volgt karakteriseren:

SOORT	GEBRUIK	KARAKTERISERING		
CPU registers	voor veelvuldig gebruikte data	snel	weinig	duur
cachegeheugen	copie van het laatst gebruikte gegevens uit het hoofdgeheugen	"	"	"
hoofdgeheugen	voor instrukties en data	"	"	"
backing-geheugen	secundair (traag) hoofdgeheugen	"	"	"
magnetische disks	direct toegankelijk achtergrondgeheugen	"	"	"
optische disks	direct toegankelijk achtergrondgeheugen	"	"	"
magneetbanden	serieel toegankelijk achtergrondgeheugen	"	"	"
massageheugen	automatisch magazijn met vele (kleine) magneetbanden	"	"	"
tape vault	brandvrije kluis waarin de in gebruik zijnde magneetbanden worden bewaard.	"	"	goed
		traag	veel	koop

Niet in alle computersystemen komen deze varianten allemaal voor, toch bestaat steeds het probleem hoe we de verschillende vormen van "storage" zo goed mogelijk benutten.
In de oudere computersystemen was dit een probleem dat door de programmeur c.q. gebruiker zel moest worden opgelost, bijvoorbeeld:

- bij het programmeren in assemblertaal bepaalt de programmeur zelf hoe hij de CPU-register gebruikt (hij tracht de eigenschap van snelle toegang zo goed mogelijk te benutten);
- bij een hoofdgeheugen dat te klein is om zijn programma plus gegevens te bevatten dient hij zelf zorg te dragen voor het achtereenvolgens inlezen van de benodigde gedeelten vanaf bijv. disk ("overlay"-technieken);
- voor het verwerken van bestanden op disk of tape bedient hij zich van allerlei in/uitvoer-hulproutines;
- voor het "on-line" brengen van een tape uit de kluis richt hij zich met een verzoek tot de operator van het systeem.

Het is echter duidelijk, dat een aanzienlijk deel van deze taken verricht kan worden door het computersysteem zelf: het geautomatiseerde **memory management**. De programmeur behoeft zicl dan niet meer bezig te houden met de "systeem-organisatorische" aspekten van zijn programma, omdat in hardware en systeemsoftware diverse hulpmiddelen zijn ingebouwd, bijvoorbeeld:

- de compiler, die de programma's in b.v. Pascal vertaalt, regelt ook het gebruik van de CPU-registers;
- het cache geheugen versnelt de toegang tot het hoofdgeheugen door het "in voorraad houden" van veelgebruikte geheugenwoorden;
- het hoofdgeheugen wordt geadresseerd via diverse hulpmethoden, en bevat slechts de direkt benodigde programma- en data-gedeelten (de rest bevindt zich op disk en wordt automatisch in het hoofdgeheugen gelezen wanneer dat nodig is);

- het "backing" geheugen fungeert als automatische buffer tussen het hoofdgeheugen en het achtergrondgeheugen (b.v. disk);
- disks bevatten de gebruikte bestanden (programma's, data);
- magneetbanden bevatten bestanden, die wanneer dat nodig is automatisch naar disk worden overgebracht, waar zij sneller toegankelijk zijn ("staging");
- het "mass storage system" kiest automatisch de juiste magneetband uit zijn magazijn;
- een magneetband die zich in de kluis bevindt wordt door de operateur opgespannen na een verzoek dat automatisch door het OS is gegenereerd.

Andere aspekten die eveneens een rol spelen bij memory management zijn:

- het verbeteren van de mogelijkheden voor multi-user gebruik van het systeem: *relocatie* (het "verschuiven" van delen programma en/of data binnen het hoofdgeheugen) en *protectie* (contrôle op ongewenste onderlinge beïnvloeding door programma's);
- het verschaffen van hulpmiddelen aan de programmeur om zijn programma's en gegevens zodanig te structureren dat deze beter aansluiten bij de logische organisatie van zijn probleem-oplossing.

In vrijwel alle computersystemen komen wel enkele van de genoemde mogelijkheden van memory management naar voren.

12.3.2 Virtueel geheugen

Virtueel geheugen is het geheugen zoals de programmeur dat ziet. Zijn programma zal met allerlei adresseringsmethoden een bepaald stuk van het virtuele geheugen gebruiken. Dit hoeft niet gelijk te zijn aan de adressen die gebruikt worden om het fysische of reële geheugen te adresseren. Het reële hoofdgeheugen kan groter zijn dan de maximale virtuele adresruimte. Om het virtuele adresbereik af te beelden in de adresruimte die het reële hoofdgeheugen biedt, moet een adresvertaling plaats vinden.
Er is dus sprake van een omzettingsfunctie f, die het recept levert voor de vertaling van adres a naar a'

```
f: N --> M
a' = f(a)        waarin

N: de reeks adressen die door een programma
   gegenereerd kan worden (het voor de programmeur
   "zichtbare" adresbereik);
M: de reeks adressen van het hoofdgeheugen;

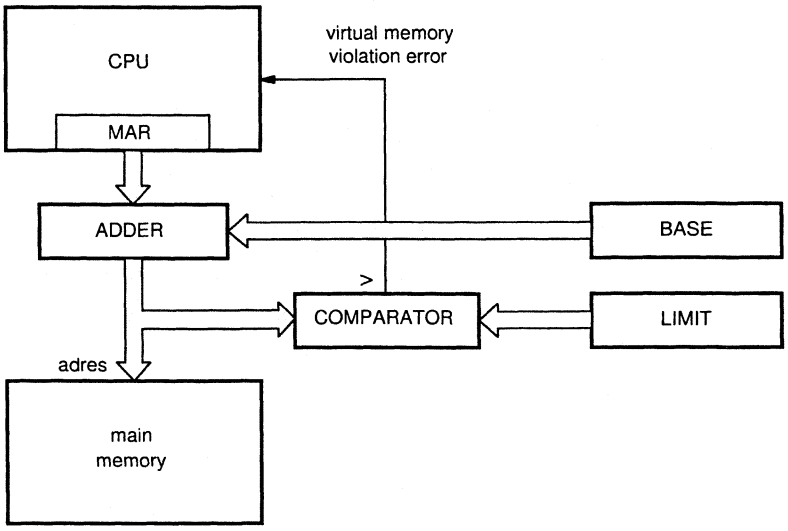
a:  een adres in N

a': een adres in M
```

Het virtuele geheugen met adresbereik N behoeft qua grootte en organisatie niet overeen te stemmen met het werkelijke geheugen met adresbereik M. Het is de taak van de hardware en de systeemsoftware om aan ieder virtueel adres a een reëel adres a' te koppelen. Deze koppeling behoeft echter geenszins dezelfde te zijn voor alle a, noch behoeft zij in de tijd onveranderlijk te zijn. Enige implementaties van zo'n adresvertaling zullen hierna behandeld worden.

12.3.3 Base en limit registers

Een eenvoudige mogelijkheid om een virtueel geheugen te implementeren werkt met een baseregister en een limitregister. Dit is aangegeven in afbeelding 168



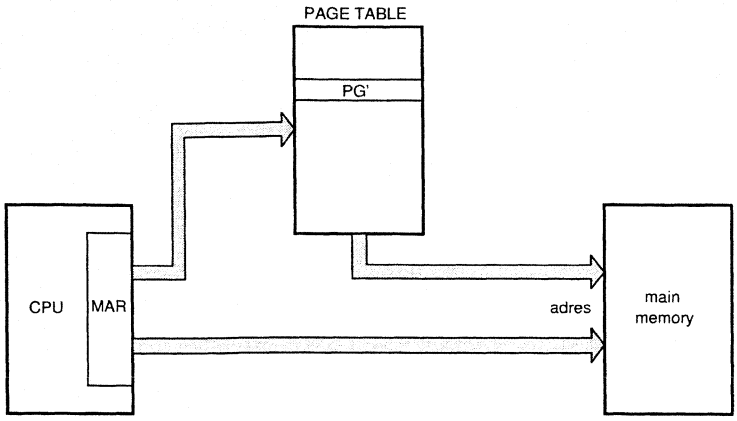
Afbeelding 168 base- en limitregisters

De virtuele adresruimte N bestaat uit de adressen 0....p terwijl de hieraan gekoppelde ruimte in M bestaat uit de adressen b....b+p. De adrestransformatiefunctie f is simpelweg de optelling van het virtuele adres a en de inhoud van het base register. Indien de resulterende waarde a' groter is dan de waarde in het limitregister overschrijdt het proces de toegestane virtuele adresruimte 0....p. Deze methode maakt relokatie zeer eenvoudig want na een verschuiving is alleen aanpassing van de base- en limit registers nodig. Ook protektie is door het limit register verzekerd. Bij overschrijding volgt een interrupt door "virtual memory violation", waarop het OS het veroorzakende proces verder afhandelt. De base- en limit registers mogen natuurlijk alleen door het OS worden ingevuld.

Merk op dat voor elk proces moet gelden N <= M, of in woorden : de werkelijke ruimte is minstens zo groot als de virtuele ruimte.

12.3.4 Paginering

Bij paginering wordt de adresruimte N van elk proces opgedeeld in gelijke stukken, de *pages* , en M opgedeeld in stukken even groot als de pages, de *page-frames*. De grootte van de pages en dus ook pagaframes is altijd een macht van 2. De virtuele adresruimte N wordt afgebeeld in M met behulp van een PAGE TABLE. Elk woord uit de PAGE TABLE levert een soort BASE adres op dat altijd wijst naar een begin van een page-frame. Omdat de grootte van de pagina's vast is zijn "LIMIT registers" overbodig.



Afbeelding 169 Paginering met N<=M

In afbeelding 169 is aangegeven hoe dit er in hardware kan uitzien.
Met het hoogstwaardige deel van het virtuele adres wordt een woord in de PAGE TABEL aangewezen. De inhoud van dat woord levert het hoogstwaardige deel van het "echte" adres op.
De offset in de page is dezelfde als de offset in het pageframe.
Als voorbeeld nu een virtuele adresruimte N van 64K(ilo) woorden, die loopt van 0000 t/m FFFF, en een reële adresruimte M van 1 M(ega) woorden, die loopt van 00000 t/m FFFFF. De paginagrootte is 8K(ilo) woorden. De PAGE TABLE ziet er bijvoorbeeld als volgt uit:

NUMMER	INHOUD
000	0101111
001	0110011
010	0100111
011	1111111
100	1100000
101	0001100
110	1010001
111	1001011

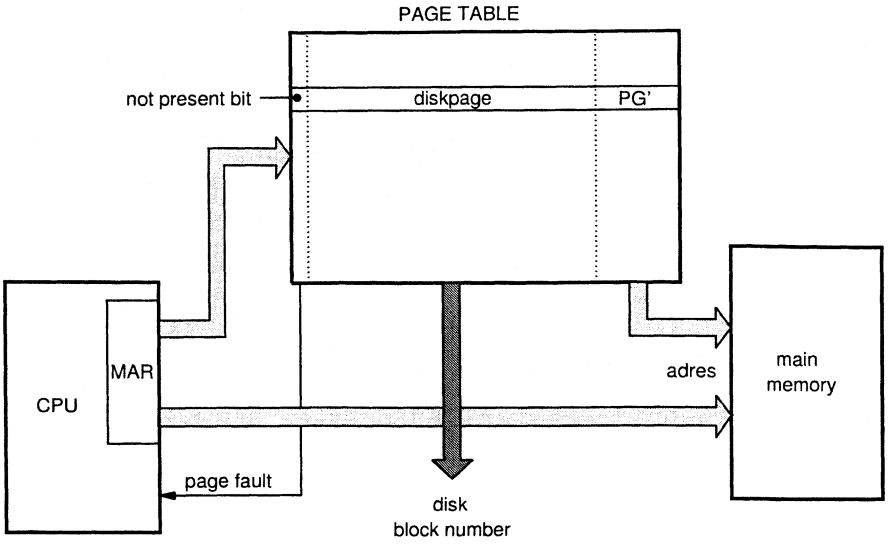
PAGE TABLE

Nu een tweetal adresvertalingen van virtueel naar reëel:

$73C2_H$ = 0111001111000010 (16 bits)
011 naar tabel en 1001111000010 offset
uit tabel 1111111,1001111000010 = $FF3C2_H$

$D234_H$ = 1101001000110100 (16 bits)
110 naar tabel en 1001000110100 offset
uit tabel 1010001,1001000110100 = $A3234_H$

Een probleem doet zich voor als de gezamenlijke virtuele adresruimten N van alle aktieve processen groter is dan de adresruimte M. Het gebrek aan RAM geheugen kan In dat geval worden aangevuld met achtergrondgeheugen, meestal de disk. Alle benodigde pageframes zijn op achtergrond aanwezig, maar slechts een beperkt deel is in het hoofdgeheugen beschikbaar.
In afbeelding 170 is te zien hoe met een kleine uitbreiding van de PAGE TABLE de zaak kan worden opgelost.



Afbeelding 170 paginering met disk

Voor elk aan te wijzen pageframe is naast een adres voor het hoofdgeheugen ook een **not present bit**, en een **diskpage** aangegeven. Bij een geheugenreferentie wordt naar het not present bit gekeken. Is dit bit 0 dan wordt gewoon het hoofdgeheugen uitgelezen. Is het not presentbit echter dan betekent dat dat de gevraagde page niet in het hoofdgeheugen aanwezig is, maar van disk moet worden gehaald. Via een interrupt **page fault** wordt de geheugenreferentie dan door een leesopdracht, dus een I/O opdracht voor de disk. Het lopende proces komt in de toestand "geblokkeerd" tot het moment dat de gevraagde pagina wel in het hoofdgeheugen aanwezig is. Als de missende pagina van disk is gelezen, moet een pageframe in het hoofdgeheugen vrij gemaakt worden om de nieuwe pagina in te kunnen zetten. Voor de bepaling welk pageframe verwijderd (overschreven) moet worden door de van disk gehaalde pagina, wordt vaak het *least recently used* principe gebruikt. Hievoor moet bij elk pageframe een teller worden gedacht die wor opgehoogd als er een geheugenreferentie plaats vindt, behalve het frame dat zelf gerefereerd word waarvan de teller op 0 wordt gezet. Moet er nu een pagina worden vervangen dan wordt dat gedaan in het frame met de maximum tellerstand. Als de gemiste pagina in het vrijgemaakte pageframe is gezet wordt in de PAGE TABLE van het proces het adres en het not present bit nog ingevuld, en het proces kan weer worden gedeblokkeerd

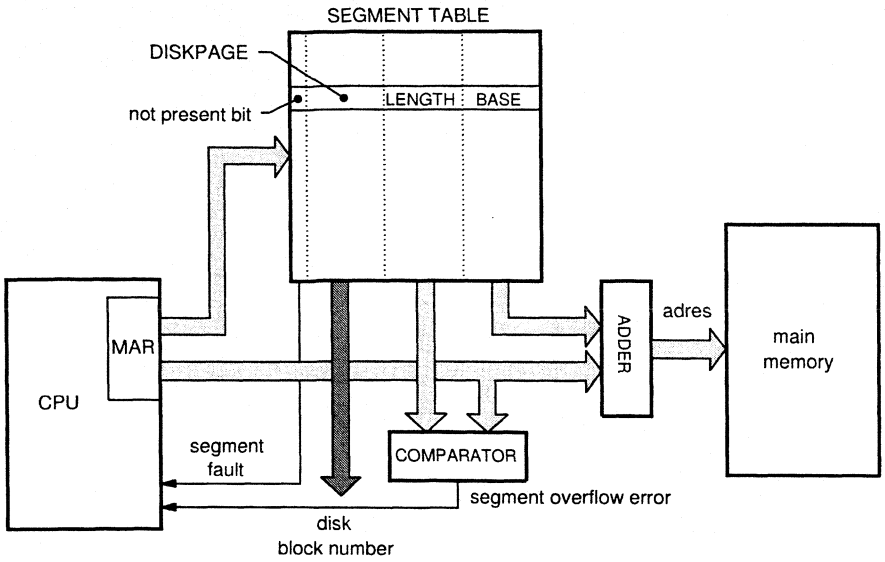
12.3.5 Segmentering

Eén van de mogelijkheden van memory management, namelijk die waarbij het geheugen wordt ingedeeld in overeenstemming met de logische organisatie van het programma is nog onbesproken gebleven. Deze logische organisatie uit zich in een modulaire opbouw, bijvoorbeeld:

- het hoofdprogramma met de grote lijnen
- diverse groepen logisch bij elkaar horende procedures en functies
- de gegevens, eventueel opgesplitst naar aard en doel
- hulpvariabelen, stacks, enz.

Het memory management kan hieraan tegemoetkomen door niet één enkele, maar een te kiezen aantal virtuele geheugenruimten ter beschikking te stellen. Iedere afzonderlijke virtule adresruimte kan dan bestemd worden voor een bepaalde kategorie programmacode of gegvens. Om spraakverwarring te voorkomen noemt men de afzonderlijke virtuele ruimten *segmenten*. Een adre

in N wordt nu aangeduid met (s,a) waarin s het segmentnummer en a het adres binnen dit segment. In afbeelding 171 is een eenvoudige hardware realisatie gegeven.



Afbeelding 171 segmentering met disk

Er is nu een SEGMENT TABLE toegevoegd, die eigenlijk bestaat uit een aantal Base en Length registers. Met behulp van het segmentnummer wordt in de SEGMENT TABLE een Base en Length kombinatie gekozen. Als het not present bit aangeeft dat dit in het geheugen aanwezig is, wordt het adres binnen het segment opgeteld bij de Base uit de tabel om het memoryadres te krijgen. Bovendien wordt dit adres vergeleken met de LENGTH uit de tabel. Is het adres groter dan LENGTH dan ontstaat een segment overflow error die via een interrupt aan de CPU gemeld wordt, en verder afgehandeld als zodanig. Als de aanvraag valt binnen een segment dat niet in het geheugen aanwezig is, wordt dit via de interrupt als een segment fault aan de CPU gemeld, die het proces blokkeert en het benodigde segment, m.b.v. DISKPAGE, naar het geheugen brengt en het Base adres in de bij het proces behorende SEGMENT TABLE invult. Vervolgens kan het proces worden gedeblokkeerd. Als het proces weer wordt geaktiveerd, wordt de SEGMENT TABLE gevuld met de segmentgegevens van het proces en kan het programma vervolgd worden.

Hoewel de overeenkomsten tussen paginering en segmentering groot lijken, is er toch een aantal belangrijke verschillen, zoals in de bijgaande tabel is te zien.

Overwegingen:	Paginering	Segmentering
Is de programmeur zich bewust van deze management techniek ?	Neen	Ja
Hoeveel lineaire adresruimten zijn er ?	1	Veel
Zijn de delen van een onveranderlijke grootte ?	Ja	Neen
Kan de totale adresruimte groter zijn dan die van het fysieke geheugen ?	Ja	Ja
Kunnen bijvoorbeeld programma en data eenvoudig onderscheiden en beschermd worden ?	Neen	Ja
Kunnen bijvoorbeeld arrays van variabele grootte eenvoudig worden toegepast ?	Neen	Ja

Het doel van paginering is het gebruik van een grote adresruimte zonder dat deze werkelijk aanwezig hoeft te zijn. Een resultaat ervan is dat geen relocatie van programma's nodig is voorafgaande aan de programmaexecutie: de programmeur, of eigenlijk de vertaler, kan code afleveren voor een denkbeeldige geheugenruimte waarover we vanaf plaats 0 de beschikking hebben.
Het doel van segmentering is het verschaffen van een aantal afzonderlijke segmenten, waarmee programma en gegevens opgedeeld kunnen worden in qua organisatie onafhankelijke adresruimten.

Segmentering met paginering

Veelal wordt segmentering gebruikt waarbij de segmenten zelf weer gepagineerd worden uitgevoerd. In dat geval behoeven grote segmenten niet als één aaneensluitend gebied in het geheugen te staan, maar kunnen worden verdeeld over willekeurige plaatsen in het geheugen. Bovendien hoeven niet alle pagina's van een segment tegelijkertijd aanwezig te zijn. Hierdoor kan een betere vulling van het werkgeheugen worden verkregen.
Men moet zich echter wel bewust zijn dat ieder proces zijn eigen segment table heeft en dat bij elk segment weer een page table hoort. Bij elke wisseling van proces zal de hardware voor PAGE TABLE en SEGMENT TABLE door het OS moeten worden ingevuld met de bij het proces en segment horende waarden.

12.4 Input en Output

Het probleem van input en output, voor een OS, is de grote verscheidenheid aan randapparatuur waarmee men te maken kan krijgen. Eigenschappen waarin randapparatuur van elkaar kunnen verschillen zijn:

- Snelheid

Disks en netwerkaansluitingen kunnen snelheden van 10^7 bytes/sec hebben, terwijl er keyboards met 10 karakters/sec in het systeem meedraaien.

- Transporteenheden

De eenheden waarmee randapparaten met de buitenwereld communiceren kunnen sterk uiteenlopen. Zo komen eenheden voor als karakters, bytes, woorden, blokken bytes (aantal bytes constant) en files (aantal bytes variabel).

- Data representatie

De codering van de over te brengen data kan verschillen. Zelfs als het standaards betreft zijn er nog verschillen. Zo bestaat voor de codering van karakters naast ASCII ook de EBCDIC code. Daarnaast zijn er talloze input/outputapparaten waarvan de code door de fabrikant bedacht is en compatibel met geen enkel ander apparaat.

- Mogelijke operaties

Sommige commando's zijn sterk afhankelijk van het type randapparaat. Er zijn apparaten die alleen geschikt zijn voor input, er zijn er die alleen output toelaten en apparaten die beide modes van operatie kennen. Maar er kunnen ook meer specifieke operaties voorkomen. Zo zal het commando "rewind" dat bij tape units voorkomt geen equivalent bij andere apparaten hebben

- Mogelijke fouten

Fouten die kunnen optreden zijn vaak afhankelijk van het type randapparaat. Enkele foutmeldingen:

"parity error", "checksum error", "disk/tape full", "out of paper", "out of range" enz.

De taak van het OS is nu om alle randapparatuur toch op een uniforme manier te kunnen bedienen. Er moet zoveel als mogelijk gezorgd worden voor:

- Onafhankelijkheid van de gebruikte codes

Bij het uitwisselen van teksten tussen computers is de gebruiker niet geïnteresseerd in de wetenschap dat de van een IBM-computer komende karakters in EBCDIC zijn gecodeerd. Het OS kan dit zelfstandig omzetten in de code die de ontvangende computer hanteert, en als karakters naar de gebruiker doorgeven.

- Onafhankelijkheid van het randapparaat

Hier zijn 2 typen onafhankelijkheid te onderscheiden. Een voorbeeld van het eerste type vanonafhankelijkheid treedt op bij **identieke** apparaten zoals disks. De gebruiker hoeft bijvoorbeeld niet te weten op welke diskunit zijn "home page" zich bevindt. Hij zal het verhuizen van de ene unit naar een andere unit niet mogen merken.
Een voorbeeld van het tweede type onafhankelijkheid treedt op bij **verschillende** randapparaten zoals bijvoorbeeld printers. Bij het printen van een stuk tekst stelt de gebruiker slechts belang in de plaats van de printer en niet in merk, gebruikte code of aansturingswijze.

- Efficiëntie

Het OS is de enige plaats waar een zo goed mogelijk gebruik van randapparaten kan worden geregeld. Een gebruiker is door gebrek aan kennis niet in staat om de armbewegingen van een disk te optimaliseren of de positie van veelgebruikte blokken op een tape unit te bepalen. De gebruiker is ook niet de meest aangewezen persoon om de prioriteit van het gebruik van een printer zodanig te regelen dat een zo groot mogelijke output van het totale systeem wordt bereikt.

- Eenheid van bediening

Om de gebruiker niet te vermoeien met het leren van apparaatafhankelijke gebruikscommando's moet het OS ervoor zorgen dat liefst met één van parameters te voorzien commando alle beschikbare randapparaten kunnen worden gebruikt.

12.4.1 I/O procedures and handlers

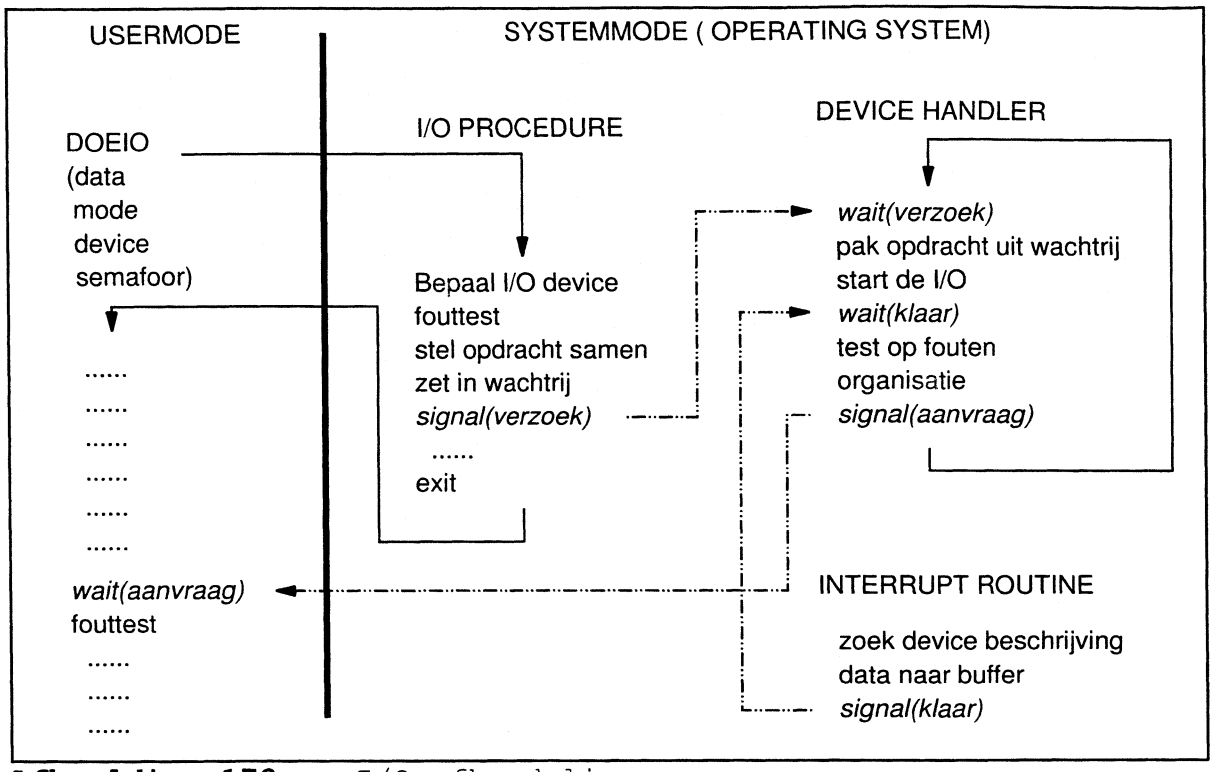
Om de eerder genoemde doelstellingen te bereiken zijn in het OS een drietal programma's werkzaam nl:

- De I/O procedure
- De device handler
- De interrupt routine

De I/O procedure regelt de uniforme afhandeling naar de gebruiker, en de communicatie met het OS. De I/O procedure wordt aan het gebruikersprogramma gelinked, en behoort daarmee tot hetzelfde proces als het gebruikersproces.
De device handler is een programma dat de besturing van het randapparaat verzorgt. Elk randapparaat heeft zijn eigen devicehandler, die op de hoogte is van de gebruikte codes, grootte van de gebruikte data eenheden, timing, mogelijke fouten, enz. De device handler genereert een apart proces.
De interruptroutine is een randapparaat afhankelijk stuk van de interruptafhandeling in de kernal.

Hiermee wordt de primaire timing en prioriteit geregeld.

In afbeelding 172 is aangegeven hoe deze processen elkaar aansturen.



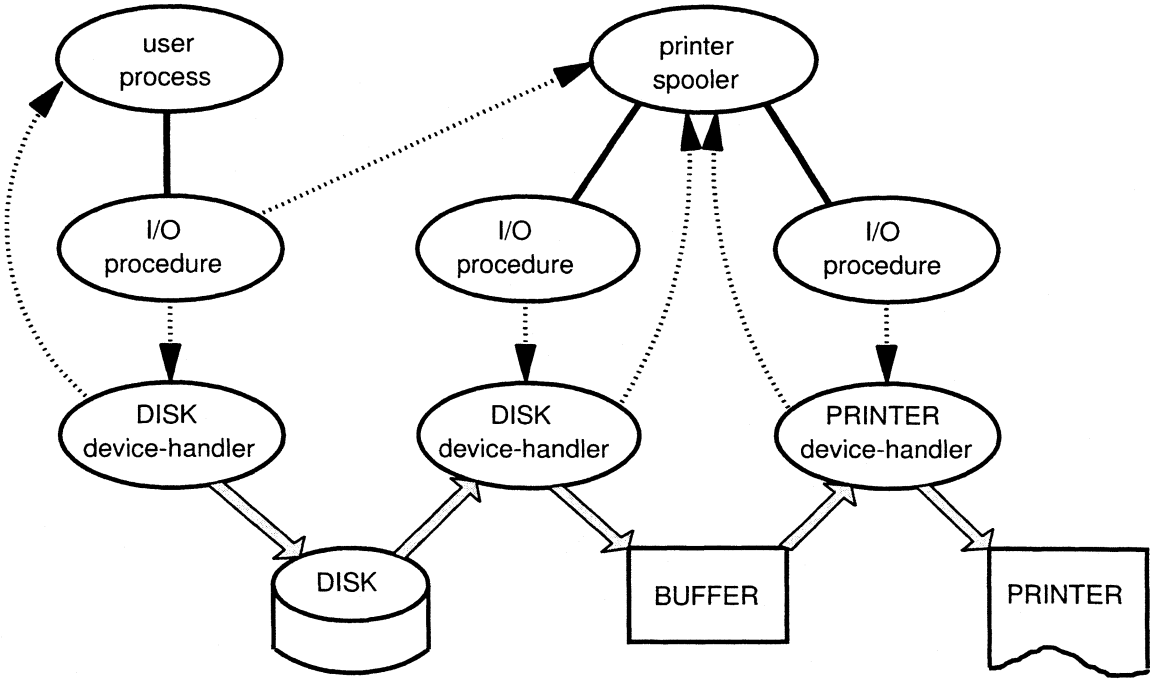
Afbeelding 172 I/O afhandeling

De gebruiker geeft in zijn I/O opdracht **DOEIO** aan met welke data, in welke mode en voor welk randapparaat de data bedoeld is. Ook geeft hij de naam van de semafoor die voor synchronisatie gebruikt zal worden. De I/O procedure kan hieruit het device bepalen, zal vervolgens testen of de parameters van de opdracht wel zinvol zijn, en stelt een "opdrachtpakketje" samen voor een specifiek randapparaat en deponeert dit "pakketje" in een wachtrij. Vervolgens wordt met een SIGNAL het sein voor de gekozen device handler op "veilig" gezet waarna weer naar het userdeel van het proces wordt teruggesprongen. In het userdeel van het proces komt een WAIT op de opgegeven semafoor voor, die pas op veilig gaat als de aangevraagde I/O voltooid is. In afbeelding 172 staat de WAIT in het userdeel van het proces, dat betekent dat de gebruiker kan bepalen tot hoever het programma mag doorwerken voordat er gestopt wordt om de voltooiïng van de I/O af te wachten. Men spreekt dan van *asynchrone I/O.* Er zijn ook systemen waarbij de WAIT deel uitmaakt van de I/O procedure. Dan wordt van *synchrone I/O* gesproken. Bij synchrone I/O is het voordeel dat de gebruiker zich niet hoeft te bemoeien met de synchronisatie, maar het nadeel is dat een proces al bij een I/O verzoek wordt "geblokkeerd", en niet door kan werken tot het punt waar de I/O echt nodig is.
De device handler wacht op het sein "veilig", wat betekent dat er minstens één opdracht in de wachtrij aanwezig is. Met behulp van zo'n pakket wordt de hardware van het randapparaat aan het werk gezet, en vervolgens wordt er gewacht op voltooiing wat blijkt uit een SIGNAL vanuit de interruptroutine. Er wordt op fouten getest, er vindt eventueel een codeomzetting plaats en de data wordt naar een buffer in het geheugen gebracht. Daarna kan de semafoor van het userproces weer op "veilig" gezet worden. In de getekende situatie wordt ervan uitgegaan dat er één datatransport plaatsvindt. Als met de I/O opdracht een aantal bytes gemoeid zijn, zal in de device handler een programma**lus** voorkomen waarin de WAIT(klaar) voorkomt en waarin de organisatie voor het "aftellen" en het wegschrijven naar een buffer zorgt. De interruptroutine zal in dat geval evenzoveel

SIGNAL's (klaar) genereren voordat de devicehandler het SIGNAL(aanvraag) kan geven.

12.4.2 Spooling

Bij een I/O verzoek wordt op een bepaald moment het I/O vragend proces "geblokkeerd" tot het moment dat de I/O voltooid is. Stel nu dat de I/O een opdracht naar een printer is, die met andere gebruikers moet worden gedeeld. Volgens de vooromschreven besturing zou het gebruikersproces geblokkeerd blijven tot het moment de printer voor dat userproces vrij is en het printen klaar is. Om dit te voorkomen wordt de data vaak niet meteen naar de printer gestuurd, maar naar een buffer, meestal de disk. Dit afhandelen van de I/O via een buffer wordt *spooling* (Simultaneous Peripheral Operation On Line) genoemd. In afbeelding 173 is deze situatie voor een printer getekend.



Afbeelding 173 spooling

De getrokken lijnen geven aan dat de verbonden delen tot hetzelfde proces behoren, de gestippelde pijlen geven aan dat de verbonden processen elkaar synchroniseren met behulp van semaforen en grijze pijlen geven de weg aan die de data aflegt. Er is te zien dat de I/O procedure van het userprocess niet alleen de disk, maar ook de printerspooler een SIGNAL geeft als er iets te doen valt. Nadat de data op disk staat geeft de DISK handler een SIGNAL naar het userprocess, dat vervolgens weer gedeblokkeerd wordt om verder te gaan. Vanaf dat moment is de printerspooler verantwoordelijk voor het afhandelen van de printopdracht.