Voorwoord bij de eerste uitgave

De tekst van het vierde deel van de cursus “Structuur en organisatie van Computersystemen” sluit dicht aan bij de drie voorgaande delen van deze cursus. Er wordt verder gebouwd op de definitie van de DRAMA model-computer uit het eerste deel. Ook bepaalde aspecten van het tweede deel, zoals de werking van de randapparatuur, worden hier gekend verondersteld. Tenslotte wordt ook van de lezer verwacht dat hij de kennis heeft van de 10-complementsvoorstelling, die in het derde deel behandeld wordt.

Om de stof uit hoofdstuk 4 niet te abstract te houden, wordt de werking van vertalers, laders en binders uitgelegd via voorbeelden voor de DRAMA-computer. Vooral de aspecten “wat doet een dergelijk programma” en “waarom is het nodig” komen aan bod, maar ook enkele implementatie details worden besproken. Het blijft echter een inleidende tekst. Een grondige studie van vertalers komt in andere cursussen aan bod.

In hoofdstuk 5 wordt de DRAMA-computer verder uitgebreid met een programma-onderbrekingsmechanisme en verschillende processortoestanden. Op deze wijze zijn we in staat de deelaspecten van een besturingsprogramma bevattelijk voor te stellen. Vooral het programmeren van de in- en uitvoerapparatuur komt hier aan bod. Wat de andere componenten van een besturingsprogramma betreft, wordt vooral de nadruk gelegd op de functionaliteit (het “wat”). De verschillende algoritmes die aangewend worden (het “hoe”) komt in de cursus “Besturingssystemen” uitgebreid aan bod.

In de tekst zijn regelmatig opgaven opgenomen. Enerzijds testen zij de opgedane (theoretische) kennis, anderzijds proberen zij deze kennis toe te passen of verder uit te diepen. Studenten mogen deze opgaven beschouwen als voorbeelden van “examenvragen”.

Tot slot nog een woordje over de DRAMA-simulator. Voor de modelcomputer is een simulator ontworpen, die voor verschillende besturingssystemen (waaronder DOS) beschikbaar is. Met deze simulator kan de lezer zelf experimenteren, voorbeelden uit proberen en opgaven oplossen en uittesten. Helaas komt de huidige implementatie van de simulator nog niet volledig overeen met de definitie van de DRAMA-machine zoals zij beschreven is in de tekst, zodat sommige voorbeelden en opgaven nog niet kunnen uitgevoerd worden op de simulator (of tot een verkeerd resultaat leiden). We hopen dit zo snel mogelijk te kunnen verhelpen.

Er werd gepoogd om de concepten en de voorbeelden vrij gedetailleerd uit te werken zodat deze cursustekst ook als zelfstudie handboek gebruikt kan worden. Een cursustekst is echter steeds vatbaar voor verbeteringen. Opmerkingen en suggesties zijn steeds welkom bij de auteur op volgend adres: Prof. B. De Decker, Departement Computerwetenschappen, Celestijnenlaan 200A, B-3001 Heverlee (België), of op het volgend e-mail adres: `Bart.DeDecker@cs.kuleuven.ac.be`.

Alvast veel studie- en leesplezier,

Prof. B. De Decker

Voorwoord bij de tweede uitgave

De organisatie van de tekst is hier en daar gewijzigd, hoofdzakelijk in het vijfde hoofdstuk. Zo zijn er bijvoorbeeld twee nieuwe paragrafen die handelen over de “Processortoestanden” en “De kosten en baten van een besturingsprogramma”. De volgorde van de voorbeelden in de paragraaf “Invoer en uitvoer” is soms veranderd. Daarnaast is het aantal figuren, voorbeelden en opgaven aanzienlijk toegenomen. Een aantal belangrijke begrippen zijn wat meer uitgewerkt. De paragrafen over de “Macro-voorvertaler”, “Het speurprogramma”, “Invoer en uitvoer” en “Taken van een besturingsprogramma” zijn het meest aangepast.

In de tekst is zoveel mogelijk gekozen voor een Nederlandstalige terminologie, maar steeds is ook de Engelstalige vorm vermeld. Achteraan zijn een Nederlandse en Engelse lijst met veel gebruikte afkortingen en een Nederlands-Engelse en Engels-Nederlandse woordenlijst opgenomen. Tenslotte is in appendix A een volledige lijst met DRAMA-bevelen samen met hun semantische betekenis opgenomen.

Voorwoord bij de derde uitgave

Het grootste verschil met de vorige uitgave ligt in het feit dat een andere tekstverwerker werd gebruikt, die meer tekenmogelijkheden biedt. De figuren zijn dan ook herwerkt. Voor het overige werden ook een aantal fouten verbeterd en zijn enkele stukken tekst herschreven. Vooral de paragrafen “De macro-voorvertaler” en “Het speurprogramma” zijn het meest bijgewerkt.

Voorwoord bij de vierde uitgave

Enkele storende fouten werden verbeterd. Hier en daar werd wat extra uitleg en nieuwe voorbeelden toegevoegd. De in- en uitvoerbewerkingen INV en UTV zijn iets gewijzigd om meer randapparaten toe te laten. Tenslotte werd de nummering van de hoofdstukken aangepast. Hoofdstukken 3 en 4 zijn de vroegere hoofdstukken 4 en 5.

Voorwoord bij de vijfde uitgave

Hoofdstuk drie (Vertalers, ...) heeft heel wat wijzigingen ondergaan. In het deel over ‘vertalers’ is elke verwijzing naar ZEDRA verdwenen, aangezien deze voorganger van DRAMA niet meer in hoofdstuk 1 besproken wordt. Verder zijn er veranderingen aangebracht aan de macro-voorvertaler, de lader, de binder, voorvertaler en het speurprogramma. Er is naar gestreefd een eenduidige structuur voor uitvoerbare programma’s te definiëren en gebruiken.

.

In hoofdstuk 4 zijn alle Pascal programma's omgezet naar C.
Tenslotte wordt de code voor de DRAMA-simulator getoond in appendix B.3 (op pag. 211).

Voorwoord bij de zesde uitgave

Hoofdstuk vier (Besturingssystemen) heeft een aantal wijzigingen ondergaan. In het deel over in- en uitvoer werd een pseudo-C notatie ingevoerd om de algoritmes gemakkelijker te kunnen beschrijven.

Voorwoord bij de zevende uitgave.

Enkele fouten in de vorige uitgave zijn verbeterd en bepaalde concepten worden extra toegelicht.

Dankwoord

Het ontwerp van een modelcomputer is geen sinecure. Enerzijds moet het model didactisch verantwoord zijn: het moet mogelijk zijn alle details van de computer uit te leggen in een beperkt aantal uren; anderzijds moet de modelcomputer de eigenschappen van een moderne computer hebben, zodat ook niet-triviale toepassingen kunnen uitgevoerd worden.

De DRAMA-machine is dan ook het werk van heel wat "enthousiaste" mensen. Ik dank hen voor de vele uren die zij hieraan gespendeerd hebben, en in het bijzonder de collega's Pierre Verbaeten en Ludo Buyst, André Mariën, Marc Snijers, Geert Van Damme, Rudi Derkinderen en Henk Van Wulpen.
De studenten Hans-Kristiaan De Splenter, Kurt Haenen en Yves Peeters implementeerden voor hun thesiswerk de simulator, de vertaler en het visualisatie-programma.
Ten slotte nog een woord van dank aan collega Ludo Buyst voor de talrijke suggesties ter verbetering van de tekst.

Inhoudstafel

Inhoudstafel .... ***iv***

Lijst van figuren .... ***viii***

Lijst van tabellen .... ***xi***

HOOFDSTUK 3 Basisprogrammatuur **1**

3.1 De vertaler 2
3.1.1 DRAMA-machine-instructies 3
3.1.2 De mini-drama-vertaler 5
3.1.3 De DRAMA-vertaler 10
3.1.4 Vertaler voor een hogere programmeertaal 18
Opgaven ***21***

3.2 De MACRO-voorvertaler 23
3.2.1 Enkel MACRO- en MCREINDE-directieven. 24
3.2.2 Extra macrodirectieven 29
Opgaven ***41***

3.3 De lader 46
3.3.1 De absolute lader 47
3.3.2 De relocerende lader 52
3.3.3 Dynamische relocatie 55
Opgaven ***60***

3.4 De binder 64
3.4.1 Waarom een binder? 64
3.4.2 Modules 65
3.4.3 Bestanden 66
3.4.4 Hoofdprogramma 67
3.4.5 De taken van de binder 70
3.4.6 Programmabibliotheken 74
3.4.7 Vertaler-, binder- en laderdirectieven 75
3.4.8 Van bronprogramma tot uitvoering 75
Opgaven ***77***

3.5 De vertolker 79
3.5.1 Werking van een vertolker 80
3.5.2 Voordelen 82
3.5.3 Nadelen van vertolking 83
3.5.4 Toepassingen van vertolking 84
Opgaven ***85***

3.6 Het speurprogramma . . . 86
Opgaven . . . *90*

3.7 Het opstarten van de computer . . . 92
Opgaven . . . *94*

HOOFDSTUK 4 Besturingssystemen . . . 95

4.1 Inleiding . . . 96
4.2 Het programma-toestandswoord . . . 98
Opgaven . . . *100*

4.3 Programma-onderbrekingen . . . 101
4.3.1 Inleiding . . . 101
4.3.2 De bevelencyclus . . . 103
4.3.3 Het programma-onderbrekingsmechanisme . . . 105
4.3.4 Soorten onderbrekingen . . . 110
4.3.5 Verbieden van programma-onderbrekingen . . . 113
4.3.6 Programma-onderbrekingsroutines . . . 118
4.3.7 Geprogrammeerde programma-onderbreking . . . 121
Opgaven . . . *123*

4.4 Invoer en uitvoer . . . 125
4.4.1 Randapparaten . . . 125
4.4.2 Intermezzo: I/O en PO-routines in C. . . . 129
4.4.3 In- en uitvoer organisatievormen . . . 131
Geprogrammeerde in- en uitvoer . . . 131
In- en uitvoer m.b.v. programma-onderbrekingen . . . 137
Directe geheugentoegang . . . 141
Kanaalbestuurders . . . 150
Satelliet-computers . . . 154
4.4.4 Randapparatuur-besturingsroutines . . . 156
Opgaven . . . *157*

4.5 Processortoestanden . . . 159
4.5.1 Halttoestand en uitvoeringstoestand . . . 159
4.5.2 Probleemtoestand en supervisietoestand . . . 162
4.5.3 Geprivilegieerde bevelen . . . 164
4.5.4 Supervisie-oproep . . . 166
4.5.5 De volledige bevelencyclus . . . 172
Opgaven . . . *173*

4.6 Multiprogrammatie . . . 175
4.6.1 Het principe . . . 175
4.6.2 De doorvoer . . . 177

4.6.3 Het STP-bevel 179
4.6.4 Het besturingsprogramma 179
Opgaven ***180***

4.7 Soorten besturingssystemen 181
Opgaven ***183***

4.8 Taken/diensten v/h besturingsprogramma 184
4.8.1 Geheugenbeheer 184
4.8.2 Processorbeheer 186
4.8.3 Het beheer van de randapparaten 190
4.8.4 Bestandenbeheer 191
4.8.5 Informatiebeheer 192
4.8.6 Boekhouding 193
Opgaven ***194***

4.9 Kosten/baten aspecten 195
4.9.1 De baten 195
4.9.2 De kosten 195
Apparatuurkosten 196
Programmatuurkosten 196
Overhead aan tijd 196
4.9.3 Het uiteindelijke rendement 197
Opgaven ***198***

Appendix A Drama-Machinedefinities 199

Appendix B Programma's in pseudo-Pascal of C. 207

B.1 De DRAMA-vertaler 207
B.2 De reloceerbare DRAMA-lader 209
B.3 De DRAMA-simulator 211

Appendix C Macro's in een willekeurige context 216

Appendix D De randapparaten van de drama-machine . . . 218

D.1 Het scherm 220
D.2 Het toetsenbord 222
D.3 De schijf 224
D.4 De drukker 226
D.5 De klok 227
D.6 De wekker 227

Appendix E Woordenlijst 228

Appendix F Gebruikte Afkortingen 232

F.1 Nederlandse afkortingen 232
F.2 Engelse afkortingen 234

Index 235

Lijst van figuren

Hoofdstuk 3

FIGUUR 3-1. De werking van de vertaler . . . 2
FIGUUR 3-2. De drama-machine . . . 3
FIGUUR 3-3. Een mini-drama-programma. . . . 5
FIGUUR 3-4. mini-drama-programma met het uitvoerbaar programma (machinebevelen) . . . 9
FIGUUR 3-5. De code na de eerste vertalingsstap. . . . 13
FIGUUR 3-6. De code na de tweede vertalingsstap. . . . 17
FIGUUR 3-7. Omzetting van C naar DRAMA. . . . 18
Van een Pascal-programma tot een uitvoerbaar DRAMA-programma. . . . 19
FIGUUR 3-8. Van een C-programma tot een uitvoerbaar DRAMA-programma. . . . 19
FIGUUR 3-9. De macrovoorvertaler . . . 23
FIGUUR 3-10. Een programma met een macrodefinitie en -oproep. . . . 24
FIGUUR 3-11. Gegevensstructuur voor de macro SOM . . . 26
FIGUUR 3-13. Toestandsdiagram van de macrovoorvertaler . . . 28
FIGUUR 3-12. Het resultaat van de MACRO-voorvertaling. . . . 28
FIGUUR 3-14. Globale variabelen gebruikt als programma-constanten. . . . 31
FIGUUR 3-15. Gebruik van een globale VV-variabele voor het genereren van unieke etiketten. 32
FIGUUR 3-16. Voorvertaling van programma uit figuur 3-15. . . . 33
FIGUUR 3-17. Gebruik van VV-variabelen en macrovariabelen. . . . 34
FIGUUR 3-18. Globale voorvertaleretiketten. . . . 36
FIGUUR 3-19. Globale en lokale voorvertaler etiketten. . . . 37
FIGUUR 3-20. Globale VV-variabelen en lokale macrovariabelen. . . . 44
FIGUUR 3-21. De lader . . . 46
FIGUUR 3-22. De DRAMA-code voor de absolute lader. . . . 48
FIGUUR 3-23. Drie programma's in het geheugen. . . . 50
FIGUUR 3-24. Een DRAMA-programma, dat geladen moet worden vanaf geheugen-register 0300. . . . 51
FIGUUR 3-25. De drie uitvoerbare programma's, respectievelijk te laden vanaf 0000, 0100 en 0300. . . . 52
FIGUUR 3-26. Een reloceerbaar DRAMA-programma. . . . 53
FIGUUR 3-27. De relocerende lader . . . 54
FIGUUR 3-28. De geheugenbeheereenheid . . . 57
FIGUUR 3-29. De binder of het koppelprogramma . . . 64
FIGUUR 3-30. Twee modules A en B . . . 65
FIGUUR 3-31. EXTERN- en GLOBAAL-directieven . . . 66
FIGUUR 3-32. Vertaling van Modules A en B uit voorbeeld 3-5 . . . 69
FIGUUR 3-33. Het uitvoerbaar programma resulterend uit objectmodules A en B . . . 72

FIGUUR 3-34. Van broncode tot geladen uitvoerbaar programma 76
FIGUUR 3-35. De werking van de vertolker. 79
FIGUUR 3-36. De geheugeninhoud tijdens de vertolking. 80
FIGUUR 3-37. Het speurprogramma. 86
FIGUUR 3-38. Het besturingspaneel van de DRAMA-machine 93

Hoofdstuk 4

FIGUUR 4-1. De toestand van de DRAMA machine. 98
FIGUUR 4-2. Het programmatoestandswoord van de DRAMA machine 98
FIGUUR 4-3. Automatische onderbreking bij een uitzonderingstoestand. 102
FIGUUR 4-4. Een oneindige lus in C en DRAMA. 103
FIGUUR 4-5. Blokschema van de bevelencyclus. 104
FIGUUR 4-6. De uitvoering van een sprongbevel. 105
FIGUUR 4-7. Een onderbrekingsmechanisme (Analogie). 106
FIGUUR 4-8. Het programma-toestandswoord en de onderbrekingsvlaggen. 107
FIGUUR 4-9. Blokschema van de bevelencyclus met een programma-onderbrekingsmechanisme 108
FIGUUR 4-10. Het effect van een programma-onderbreking op de bevelenteller. . . 109
FIGUUR 4-11. Het verbieden of uitstellen van onderbrekingen. 113
FIGUUR 4-12. Het maskerwoord in het DRAMA-PTW. 114
FIGUUR 4-13. Globale maskers in het DRAMA-maskerwoord. 115
FIGUUR 4-14. Prioriteiten bij onderbrekingen. 116
FIGUUR 4-15. Het huidige onderbrekingsniveau. 117
FIGUUR 4-16. Een typische PO-behandelingsroutine. 119
FIGUUR 4-17. De uitvoering van programma’s gedurende een zeker tijdsinterval. . 120
FIGUUR 4-18. Initialisatie van het PTW bij de drama-machine. 121
FIGUUR 4-19. Initialisatie van de DRAMA machine (vervolg). 122
FIGUUR 4-20. Inpassing van poorten in het adresbereik. 126
FIGUUR 4-21. Een typische bestuurder van een randapparaat. 127
FIGUUR 4-22. Het toestandsdiagram voor een bestuurder. 128
FIGUUR 4-23. Tijdsdiagram bij uitvoer met actief wachten. 134
FIGUUR 4-24. Tijdsdiagram bij invoer met actief wachten. 137
FIGUUR 4-25. Tijdsdiagram bij lezen van schijf met DGT. 148
FIGUUR 4-26. De bestuurder (via dgt) steelt geheugencycli van de processor. 149
FIGUUR 4-27. Het toestandsdiagram voor een bestuurder die directe geheugen-toegang gebruikt. 150
FIGUUR 4-28. Processor met kanaalbestuurder. 151
FIGUUR 4-29. Een kanaalprogramma. 152
FIGUUR 4-30. Een grote computerinstallatie met front-end en back-end computers. 154

FIGUUR 4-31. Randapparaat-besturingsroutines en PO-routines in het besturingsprogramma. 157
FIGUUR 4-32. De halt-/uitvoeringsindicator. 160
FIGUUR 4-33. De overgangen tussen de halt- en uitvoeringstoestand. 161
FIGUUR 4-34. De supervisie-/probleemtoestand indicator. 163
FIGUUR 4-35. De overgangen tussen de verschillende processor-toestanden. 163
FIGUUR 4-36. Opdeling van de machine-instructies op de DRAMA-computer. 165
FIGUUR 4-37. Een supervisie-oproep. 170
FIGUUR 4-38. Multiprogrammatie. 176
FIGUUR 4-39. Mono- versus multiprogrammatie. 178
FIGUUR 4-40. Geheugentoekenning aan verschillende programma's. 184
FIGUUR 4-41. Geheugenbeschermregisters. 185
FIGUUR 4-42. Elk geheugenadres wordt door de hardware gecontrolleerd. 185
FIGUUR 4-43. Het toestandsdiagram van een programma. 187
FIGUUR 4-44. Het afwisselend uitvoeren van verschillende programma's. 189
FIGUUR 4-45. Verschillende bestanden op schijf. 191
FIGUUR 4-46. Het besturingsprogramma houdt de huidige datum en de tijd bij. . . . 192

Appendices

FIGUUR B-1. Pseudo pascal-code voor de vertaler 208
FIGUUR B-2. Pseudo-Pascal-code voor de relocerende lader 210
FIGUUR D-1. Een typische bestuurder van een randapparaat. 218
FIGUUR D-2. Het toestandsdiagram voor een bestuurder. 219

Lijst van tabellen

Hoofdstuk 3

TABEL 3-1. De machine instructie tabel voor MINI-DRAMA. 7
TABEL 3-2. De Symbooltabel (na stap 1). 15
TABEL 3-3. De symbooltabel in ascii-voorstelling 15
TABEL 3-4. De argumententabel. 26
TABEL 3-5. Macrodirectieven, hun context en betekenis 29
TABEL 3-6. Globale VV-variabelen tabel. 35
TABEL 3-7. Argumenten-en-lokale -variabelentabel. 35
TABEL 3-8. Globale Etiketten Tabel. 38
TABEL 3-9. De gegevensstructuur van een gedefinieerde macro. 38
TABEL 3-10. De gegevensstructuren van de macrovoorvertaler. 39
TABEL 3-11. Instructies voor het manipuleren van de GBE 58
TABEL 3-12. De allocatie- en globale symbooltabel. 71
TABEL 3-13. Relocatie/bindingstabellen in de modules en het uitvoerbaar bestand. 73
TABEL 3-14. Vertalerdirectieven en informatie voor binder en lader. 75
TABEL 3-15. De breekpuntentabel van het speurprogramma. 88

Hoofdstuk 4

TABEL 4-1. Programma-onderbrekingen in DRAMA. 112
TABEL 4-2. Instructies voor het plaatsen/testen van maskers en testen van vlaggen. 115
TABEL 4-3. De toestanden van een randapparaat bestuurder. 127
TABEL 4-4. Toestandsovergangen met een programma-onderbrekingsaanvraag. 129
TABEL 4-5. De opdrachten voor de schermbestuurder. 132
TABEL 4-6. Enkele opdrachten voor de schijfbestuurder. 142
TABEL 4-7. De uitvoering van instructies in de verschillende uitvoeringstoestanden. 165
TABEL 4-8. “Dienstnummers”' voor het DRAMA-besturingsprogramma. 169
TABEL 4-9. De geheugen-allocatie tabel. 186
TABEL 4-10. De programma-toestandstabel. 187
TABEL 4-11. De randapparaat-toestandstabel. 191
TABEL 4-12. De inhoudstafel van de schijf. 192

Appendices

TABEL A-1. De **drama**-machine-instructietabel. 200
TABEL A-2. De DRAMA-instructieklassen 200
TABEL A-3. Informatie tabel over de instructieklassen 201
TABEL A-4. De mogelijke waarden voor de twee modus-cijfers 201
TABEL A-5. De vertaling van de VSP-voorwaarden. 202
TABEL A-6. De betekenis van de VSP-voorwaarden 202
TABEL A-7. De vertaling van de onderbrekingsnummers 203
TABEL A-8. De vertaling van de register- en poortnummers 203
TABEL A-9. De drama-bevelen en hun betekenis (semantiek) 204
TABEL A-10. De drama-bevelen en hun betekenis (semantiek) (vervolg) 205
TABEL A-11. De ASCII-code tabel. 206

HOOFDSTUK 3
Basisprogrammatuur

Dit hoofdstuk is eerst gewijd aan de werking van een ***vertaalprogramma.*** De taak van de MINI-DRAMA-vertaler is vrij eenvoudig en houdt weinig meer in dan een lijn-per-lijn vertaling. Voor DRAMA-programma's moet de vertaler ook symbolische adressen verwerken, waardoor hij heel wat complexer wordt.

In de tweede paragraaf wordt de vertaling van macro's onder de loep genomen. In de DRAMA-omgeving werd hiervoor een aparte ***voorvertaler*** ontworpen, die de macro-oproepen expandeert naar gewone DRAMA-instructies.

Nadat een programma vertaald is, moet het eerst in het werkgeheugen van de computer gebracht worden alvorens het kan uitgevoerd worden. Dit gebeurt door een speciaal programma: de ***lader***. In de derde paragraaf van dit hoofdstuk zullen we enkele verschillende laders bestuderen.

Grotere programma's worden vaak opgedeeld in kleinere stukken, modules genoemd, die elk in een afzonderlijk bestand worden opgeslagen en ook afzonderlijk worden vertaald. Het is de taak van de ***binder*** om deze afzonderlijk vertaalde modules samen te voegen tot één groot uitvoerbaar programma. De vierde paragraaf is aan de binder gewijd.

In plaats van programma's eerst te vertalen alvorens ze uit te voeren, kan men ook gebruik maken van een ***vertolker*** die rechtstreeks het bronprogramma interpreteert. De vijfde paragraaf gaat dieper in op de werking van een vertolker.

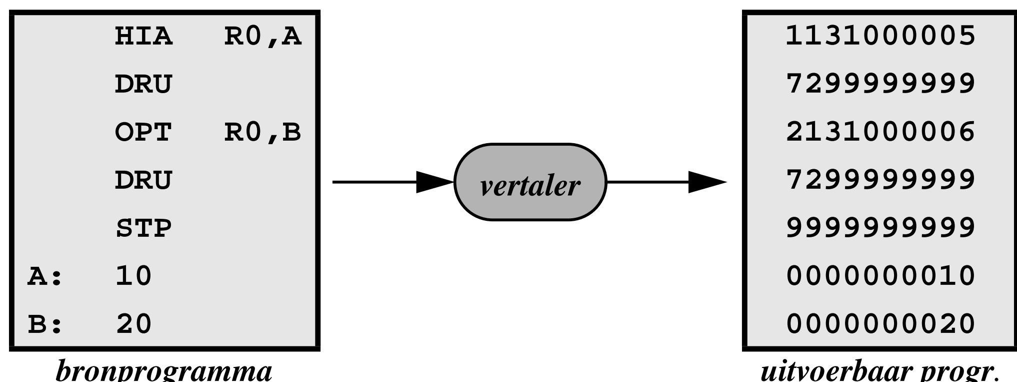
Weinig programma's zijn foutloos. Om het opsporen van fouten te vergemakkelijken kan men beroep doen op een ***speurprogramma*** dat een programma onder begeleiding van de gebruiker laat uitvoeren. De zesde paragraaf van dit hoofdstuk is gewijd aan speurprogramma's.

Tot slot, en als overgang naar het volgende hoofdstuk, gaan we na wat er gebeurt bij het opstarten van het computersysteem, wanneer het werkgeheugen nog 'leeg' is. Een speciale techniek, ***bootstrap*** genoemd, zal beschreven worden.

3.1 De vertaler

Programma's geschreven in een lagere (of hogere) programmeertaal moeten eerst omgezet ***(vertaald)*** worden naar ***machinetaal***, vermits de computer alleen deze laatste taal begrijpt. Voor kleine programma's kan de programmeur zelf deze omzetting doen. Het vereist echter een volledige kennis van de bevelenopmaak, van de verschillende velden in een machinebevel, alsook van de betekenis van de verschillende waarden die deze velden kunnen hebben. Met andere woorden, de programmeur moet de machine door en door kennen.

Toen de computer nog in de kinderschoenen stond, hadden de eerste programmeurs al snel door dat de computer het geschikte medium was om zelf voor deze vertaling te zorgen. Een speciaal programma, de ***vertaler*** of het ***assembleerprogramma*** genoemd (*assembler* in het Engels), zal deze taak uitvoeren. Figuur 3-1 schetst schematisch de werking van de vertaler: de invoer is een DRAMA-programma, de uitvoer hetzelfde programma in machinetaal. De invoer noemen het ***bronprogramma***, de uitvoer het ***uitvoerbaar programma***.



FIGUUR 3-1. De werking van de vertaler

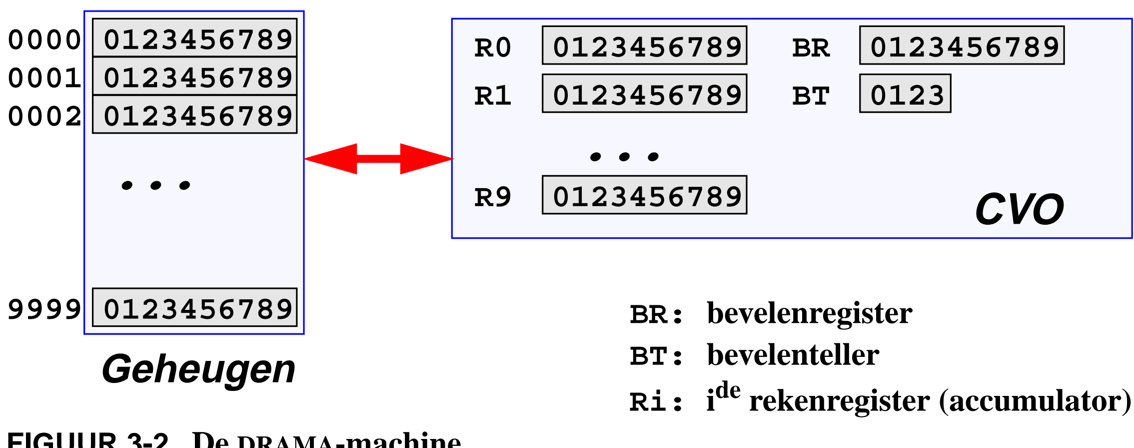
We zullen eerst de werking van de vertaler uitleggen voor programma's geschreven in een vereenvoudigde DRAMA-taal (MINI-DRAMA). Daarna zullen we de vertaler uitbreiden zodat hij ook DRAMA-programma's kan omzetten. Tenslotte bekijken we de werking van een compilator voor een hogere programmeertaal.

In de tekst zullen we ons beperken tot een informele beschrijving van de werking en de gebruikte gegevensstructuren. In de meeste gevallen zullen we vertrekken van een voorbeeldprogramma.

Voor we de vertaler onder de loep nemen, geven we eerst nog een bondige samenvatting van de bevelenopmaak van de DRAMA-machine. Hierbij zullen we verwijzen naar bevelen die nog niet gekend zijn: **`MKL`**, **`INV`**, **`UTV`**, …; deze bevelen worden in het volgende hoofdstuk uitgelegd. Ze hebben te maken met invoer- en uitvoerbewerkingen, en met het programmaonderbrekingsmechanisme.

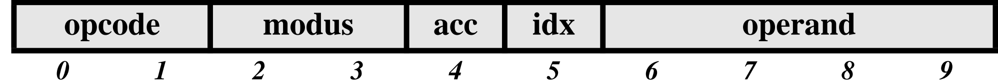
3.1.1 DRAMA-machine-instructies

Op de DRAMA-machine zijn alle geheugenregisters 10 decimale cijfers lang. Elk geheugenregister kan één machine-instructie of één decimaal getal bevatten. Zowel instructies als getallen worden decimaal[1] voorgesteld, zodat we achteraf niet kunnen afleiden of een geheugenregister nu een instructie of een getal bevat. Wel is het zo dat bepaalde getalvoorstellingen geen voorstelling van een bevel kunnen zijn: bv. op de DRAMA-machine bestaan er geen instructies die beginnen met `00...`, `01...`, enz. Indien de processor toch zou proberen dit 'getal' uit te voeren, zal de 'apparatuur' een fout detecteren en een ***programmaonderbreking*** veroorzaken (*interrupt* in het Engels). Programmaonderbrekingen worden in paragraaf 4.3.3 (op pag. 105)) besproken.



FIGUUR 3-2. De DRAMA-machine

In een DRAMA-instructie kunnen we verschillende velden onderscheiden, bestaande uit 1, 2 of 4 cijfers (de cijfers zullen we van links naar rechts nummeren van `0` t.e.m. `9`):



1. Het **opcode**-veld (of operatiecodeveld), bestaat uit twee decimale cijfers en geeft aan welke ***bewerking*** moet uitgevoerd worden. Voor elke mnemotechnische code is er een overeenkomstige decimale operatiecode. Bijvoorbeeld, opcode `11` staat voor `HIA`, opcode `99` daarentegen betekent `STP`. Tabel A-1 (pag. 200) geeft een volledig overzicht van de beschikbare operatiecodes samen met de overeenkomstige mnemotechnische naam.
2. Het **modus**-veld geeft aan hoe de operand van de instructie moet berekend en geïnterpreteerd worden. Het bestaat uit twee decimale cijfers, die elk hun specifieke betekenis hebben:
   a. Het **tweede modus-cijfer** geeft aan hoe de operandwaarde (of adres) berekend moet worden: met of zonder indexatie (eventueel met auto-increment of -decrement).

1. Op een reële computer worden de bevelen en getallen natuurlijk in het binair voorgesteld.

b. Het **eerste modus-cijfer** geeft aan hoe de berekende operand[1] gebruikt moet worden: als een waarde in het interval **`[-5000,4999]`**, een waarde in het interval **`[0,9999]`**, een adres of een indirect adres. In de eerste twee gevallen kan uitvoering van de instructie onmiddellijk gestart worden; in de laatste twee gevallen zal de inhoud van één of twee (bij indirectie) geheugenregisters opgehaald moeten worden.

Tabel A-4 (pag. 201) geeft een gedetailleerd overzicht van de mogelijke waarden.

3. Het **acc**-veld (of accumulatorveld) bestaat uit één decimaal cijfer en geeft aan op welk rekenregister de instructie van toepassing is. Het kan waarden van **`0`** tot en met **`9`** aannemen. Voor sommige instructies (zoals **`SPR`**) heeft dit veld geen betekenis. In dit geval wordt per conventie de waarde **`9`** genomen. Andere instructies zoals **`VSP`** of **`MKL`** gebruiken het acc-veld om respectievelijk de 'voorwaarde' of het 'onderbrekingsnummer' aan te duiden. Tabellen A-5 (pag. 202) en A-7 (pag. 203) bevatten de mogelijke waarden.

4. Het **idx**-veld (of indexregisterveld) bestaat ook uit één decimaal cijfer en geeft aan welk register als indexregister gebruikt moet worden. Soms is er helemaal geen indexregister nodig: het tweede modus-veld geeft aan of dit veld enige betekenis heeft of niet. Ook hier zullen we per conventie de waarde **`9`** in het idx-veld plaatsen indien dit veld (voor deze opcode) nooit gebruikt wordt (d.w.z. indexatie is nooit mogelijk, bijv. bij **`LEZ`**, **`STP`**, ...; het tweede modus-cijfer is dan ook **`9`**); we zullen daarentegen een **`0`** invullen indien er geen indexregister gebruikt wordt (het tweede modus-cijfer is dan **`1`**).

5. Tenslotte bestaat het **operand**-veld uit vier decimale cijfers. Het kan het adres van een geheugenregister bevatten, of een getal. Vooraleer dit adres of getal gebruikt wordt door de processor kan het nog gewijzigd worden d.m.v. indexatie.
`INV` en **`UTV`** zullen dit veld gebruiken om het poortnummer aan te duiden (zie ook tabel A-8 (pag. 203)).

Alhoewel de bevelenopmaak vast is, zullen bepaalde velden voor sommige instructies betekenisloos zijn. Daarom worden een aantal ***klassen*** gedefinieerd. In DRAMA zijn er zeven, genummerd K_1 tot K_7. Twee worden er hier afgebeeld, de overige kan je in tabel A-2 (pag. 200) vinden. K_1 is de bevelenopmaak voor **`HIA`**, **`OPT`**, ..., terwijl K_6 de bevelenopmaak is voor de bevelen die geen operanden hebben, zoals **`STP`**, **`LEZ`**, …

K_1 :	opcode		modus		acc	idx	operand			
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

K_6 :	opcode		9	9	9	9	9	9	9	9
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Merk op dat voor klasse K_6 alleen de eerste 2 cijfers enig nut hebben!

Met onze ***expertenkennis*** van de DRAMA-machine, kunnen we de werking van het vertaalprogramma uitleggen!

1. Dit is de oorspronkelijke operand, eventueel geïndexeerd.

3.1.2 De MINI-DRAMA-vertaler

De MINI-DRAMA-taal is een vereenvoudiging van de de DRAMA-taal. Ze kent alleen **mnemotechnische functiecodes** eventueel gevolgd door een registeraanduiding[1] en een **decimaal adres** (maximaal 4-cijfers). De taal kent geen ***symbolische adressen***, noch ***geheugenreservaties***. Ook van verschillende ***interpretaties*** van de bevelen is geen sprake.

Constanten[2] laten we wel toe, maar de programmeur moet zelf het adres bijhouden van het geheugenregister waarin het weggeborgen zal worden. Tenslotte kan het programma gedocumenteerd worden via **commentaar**, die steeds begint na de vertikale streep (het '|' teken).

Het programma wordt afgesloten met een **EINDPR**-directief. Alles wat eventueel nog na dit directief volgt, wordt niet meer bekeken door de vertaler.

Het volgende MINI-DRAMA-programma berekent de som van de kwadraten van twee getallen en drukt het resultaat af. Voor de duidelijkheid is elk bevel/constante voorafgegaan door het geheugenadres waarin het bevel of de getalwaarde geplaatst zal worden. Aangezien we slechts over één register (**R0**) beschikken, zijn we verplicht tussenresultaten in het geheugen weg te bergen (zie bijvoorbeeld **tmp** op adres 10).

```
       | A [= 10] op adres 8
       | B [= 20] op adres 9
       | tmp [= tussenresultaat] op adres 10
       | Bereken A*A+B*B en druk het resultaat af
0000:  HIA  R0,8        | R0 = A
0001:  VER  R0,8        | R0 = A*A
0002:  BIG  R0,10       | tmp = A*A
0003:  HIA  R0,9        | R0 = B
0004:  VER  R0,9        | R0 = B*B
0005:  OPT  R0,10       | R0 = A*A+B*B
0006:  DRU
0007:  STP
0008:  10               | A
0009:  20               | B
       EINDPR
       Deze lijn wordt niet gelezen door de vertaler!
```

FIGUUR 3-3. Een MINI-DRAMA-programma.

1. De enige toegelaten registeraanduiding is **R0**. Voor de **VSP**-instructie wordt de registeraanduiding vervangen door de voorwaarde: de enige toegelaten voorwaarde is **NNEG** (of **GRG**).
2. Als we heel nauwkeurig zijn, dan is de term constante niet juist. Het gaat immers eerder om een of meerdere geïnitialiseerde variabelen, waarvan de waarde door een **BIG** kan gewijzigd worden.

Werking van de vertaler

De taak van de MINI-DRAMA-vertaler is vrij eenvoudig: hij leest het programma lijn-per-lijn in, en voor elke lijn voert hij het volgende algoritme uit:

1. hij verwijdert eventueel aanwezige commentaar, (alles na het '|'-teken tot het einde van de lijn wordt weggelaten),

2. hij gaat na wat er van de lijn overblijft:

  2.a. niets (de lijn bevatte alleen commentaar),

  2.b. een decimaal getal (d.i. een constante of een geïnitialiseerde variabele),

  2.c. een instructie

  2.d. het **`EINDPR`**-directief: de vertaler stopt.

3. hij genereert het equivalente in DRAMA-machinetaal (indien **2.b** of **2.c**).

Lege lijnen hoeven niet vertaald te worden; de MINI-DRAMA-vertaler zal de lijnen echter wel meetellen, zodat eventuele fouten met het juiste lijnnummer kunnen weergegeven worden.

Constanten die een negatieve waarde voorstellen, worden omgezet naar hun overeenkomstige ***10-complementsvoorstelling*** (zie hoofdstuk 6, pag. 105); alle constanten worden tenslotte met tien cijfers uitgeschreven, eventueel vooraan aangevuld met nullen; indien het getal te groot is zal de vertaler een foutmelding geven. De volgende tabel geeft enkele voorbeelden:

broncode	machinecode
`12345`	`0000012345`
`-528`	`9999999472`
`-1234567890`	`8765432110`

MINI-DRAMA-instructies kunnen gemakkelijk omgezet worden naar machinetaal. Daartoe gebruikt de vertaler een tabel, de ***machine-instructietabel***, die aangeeft hoe elke 'mnemotechnische naam' wordt omgezet naar zijn 'decimale equivalent'. Tabel 3-1 geeft de inhoud van deze tabel weer. Merk op dat modus-, acc- en idx-veld ook reeds vastliggen, vermits geen interpretaties noch indexering toegelaten zijn en er slechts één registeraanduiding mogelijk is, namelijk **`R0`**.

In het geheugen wordt de mnemotechnische functiecode, die uit drie letters bestaat, volgens de ASCII-code voorgesteld (zie hoofdstuk 6, par. 6.8.2 (op pag. 164)); in het decimaal talstelsel betekent dit dat we drie cijfers voor elke letter nodig hebben. Een mnemotechnische functiecode kunnen we dus in één geheugenregister van tien decimale cijfers bewaren (voor het eerste cijfer nemen we **`0`**). (De ASCII-tabel vind je in tabel A-11 (pag. 206)). Dus **`HIA`** wordt voorgesteld als `0 072 073 065`, **`BIG`** als `0 066 073 071`, enz.

De tabel is gesorteerd, zodat het zoeken eenvoudiger en efficiënter wordt. Aangezien in de ASCII-voorstelling opeenvolgende letters door opeenvolgende getallen worden voorgesteld, is alfabetisch sorteren eenvoudig: de decimale representatie is van klein naar groot gesorteerd. Merk op dat deze bewering niet langer opgaat wanneer kleine en hoofdletters door elkaar

gebruikt worden. De laatste kolom (onder de hoofding '**Operand?**' geeft aan of de functiecode een operand mag hebben. De '***booleaanse waarde***' (ja of neen) wordt natuurlijk ook decimaal voorgesteld: 'neen' wordt door '`0`' voorgesteld, en 'ja' door '`1`'.

TABEL 3-1. De ***machine instructie tabel*** voor MINI-DRAMA.

Mnemo-technische functiecode	Decimale voorstelling functiecode	Opcode	Cijfers 3-6	Operand?	
AFT	0065070084	22	3100	1	ja
BIG	0066073071	12	2100	1	ja
DEL	0068069076	24	3100	1	ja
DRU	0068082085	72	9999	0	neen
HIA	0072073065	11	3100	1	ja
LEZ	0076069090	71	9999	0	neen
MOD	0077079068	25	3100	1	ja
NWL	0078087076	73	9999	0	neen
OPT	0079080084	21	3100	1	ja
SPR	0083080082	32	2100	1	ja
STP	0083084080	99	9999	0	neen
VER	0086069082	23	3100	1	ja
VSP	0086083080	33	2120	1	ja

Op onze DRAMA-computer zou deze tabel in 26 opeenvolgende geheugenregisters opgeslagen worden. Per instructie hebben we 2 geheugenregisters nodig: één voor de decimale voorstelling van de mnemotechnische functiecode en één geheugenregister voor de overige informatie (opcode, …). Indien we de matrix volgens de 'kolomvoorstelling[1]' lineariseren, ziet ze er als volgt uit:

0065070084	*'AFT'*
0066073071	*'BIG'*
...	
0086083080	*'VSP'*
0002231001	*opcode,... van AFT*
0001221001	*opcode,... van BIG*
...	
0003321201	*opcode,... van VSP*

1. Waarom is in dit geval de kolomvoorstelling beter dan de rijvoorstelling?

Een uitgewerkt voorbeeld

Laten we de theorie even toepassen op een kort MINI-DRAMA-programma.

Voorbeeld 3-1. *Gegeven het volgende MINI-DRAMA-programma:*

```
| Lees X in
| Bereken Y = X**2 - 5
| Druk X, Y af
LEZ            | X in Acc
DRU
BIG  R0,100
VER  R0,100
OPT  R0,8      | Acc = X**2 - 5
DRU
NWL
STP
-5
EINDPR
```

Probeer zelf eerst — met behulp van tabel 3-1 — de code te genereren. Wanneer geen adresoperand nodig is, vul je de waarde `9999` in (bijvoorbeeld bij `LEZ`, `DRU`, `NWL` en `STP`). Ook andere velden die geen betekenis hebben voor een bepaalde instructie, werden in de tabel reeds ingevuld met negens.

De vertaler leest het programma lijn per lijn in en genereert ***onmiddellijk***, d.w.z. in één stap, de overeenkomstige machinecode. Bij de DRAMA-vertaler zal dit niet meer het geval zijn! De vertaler zal het **uitvoerbare programma** (d.w.z. de machinebevelen) genereren zoals aangegeven is in figuur 3-4 (b). Het bronprogramma is hernomen zodat je duidelijk kan zien, hoe elk MINI-DRAMA-bevel omgezet is in een machinebevel. Het uitvoerbaar programma is de (donkergekleurde) laatste kolom. Merk op dat `9999999995` (in de laatste lijn) de 10-complementvoorstelling is voor `-5`.

De vertaler zal ook het programma nakijken op **fouten**:

- het gebruik van ***onbestaande mnemotechnische functiecodes***,
- ***constanten die niet met tien cijfers kunnen voorgesteld worden***,
- ***adresoperanden die buiten het interval [0000,9999] vallen***,
- het opgeven van een ***operand bij functiecodes die geen operand mogen hebben***.

In voorbeeld 3-2 (pag. 9) wordt een MINI-DRAMA-programma getoond dat een aantal fouten bevat, alsook de foutboodschappen die door de vertaler gegenereerd worden. Indien de vertaler fouten vindt, zal hij uiteraard geen uitvoerbaar programma aanmaken.

*(a) mini-*DRAMA*-bron programma*

```
| Lees X in
| Bereken Y = X**2 - 5
| Druk X, Y af
LEZ          | X in Acc
DRU
BIG  100
VER  100
OPT  8       | Acc = X**2 - 5
DRU
NWL
STP
-5
EINDPR
```

(b) Machinebevelen

```
7199999999
7299999999
1221000100
2331000100
2131000008
7299999999
7399999999
9999999999
9999999995
```

FIGUUR 3-4. mini-drama-programma met het uitvoerbaar programma (machinebevelen)

In het volgende voorbeeld wordt een foutief MINI-DRAMA-programma getoond.

Voorbeeld 3-2. *Gegeven het volgende bronprogramma, waarin vier fouten staan:*

```
(L1)  | mini-drama-progr. met fouten
(L2)  |
(L3)  LEZ          5
(L4)  OPS          4
(L5)  AFT          10239
(L6)  DRU
(L7)  STP
(L8)  -8000000000
(L9)  EINDPR
```

De vertaler zal de volgende foutboodschappen op het scherm uitschrijven:

```
*** fout ***  lijn 3:  LEZ mag geen operand hebben.
*** fout ***  lijn 4:  OPS is geen bestaande functiecode.
*** fout ***  lijn 5:  adres 10239 is groter dan 9999.
*** fout ***  lijn 8:  -8000000000 kan niet met 10 cijfers
                       voorgesteld worden.
```

3.1.3 De DRAMA-vertaler

De DRAMA-taal kent (net zoals MINI-DRAMA) vier verschillende ***invoerlijnen***:

1. **commentaarlijnen**, dit zijn lijnen die enkel bestaan uit commentaar (na het '`|`' symbool),
2. **instructies** die door de DRAMA-machine kunnen uitgevoerd worden (bv. `HIA.w  R1,0`),
3. **constanten**, ook wel geïnitialiseerde geheugenplaatsen (of variabelen) genoemd,
4. **vertalerdirectieven**, zoals `RESGR` en `EINDPR`.

Commentaarlijnen kunnen zonder meer overgeslagen worden.

Constanten hebben we reeds bij de MINI-DRAMA-vertaler behandeld. Ook hier weer zal de vertaler negatieve constanten omzetten naar de 10-complementnotatie, en de constanten voorstellen met exact 10 cijfers. Nieuw is echter dat constanten ook een naam (d.i. een symbolisch adres) mogen hebben. (Zie verder.)

Er zijn ook twee **directieven**: `RESGR` en `EINDPR`. Het `RESGR` kan men opvatten als een verkorte notatie voor opeenvolgende `0`-constanten. (`RESGR` reserveert een niet geïnitialiseerde geheugenplaats; ze met nul initialiseren is dus zeker niet foutief.) `EINDPR`[1] geeft het einde aan van het bronprogramma, en zal bijgevolg niet vertaald worden.

De DRAMA-**instructies** zelf zijn moeilijker te vertalen dan MINI-DRAMA-instructies. We geven een aantal verschilpunten:

- Er zijn tien verschillende registeraanduidingen mogelijk, van `R0` t.e.m. `R9`. Register `R9` heeft bovendien een alternatieve naam: `SR` (stapelregister).
- Sommige operanden zijn alleen ***geldig*** bij bepaalde instructies. Zo zijn poortnummers `P0`, ..., `P9999` enkel ***toegelaten*** bij `INV` en `UTV`, en mogen voorwaarden zoals `KL`, `POS`, ... enkel bij de `VSP`-instructie voorkomen. Dus de operanden moeten geïnterpreteerd worden in de **context** van de functiecode zelf.
- **Interpretaties** (`.a`, `.w`, `.d` of `.i`) hebben een invloed op de inhoud van het **modus-veld**. Sommige zijn niet toegelaten bij bepaalde instructies (vb. `BIG.a` is niet geldig; `STP` mag helemaal niet gevolgd worden door een interpretatie.) De ***verzuiminterpretatie*** is normaal gezien '`.d`', tenzij de instructie een register-register operatie is, dan is de verzuimwaarde '`.w`'.

  Erger nog, een bepaalde interpretatie wordt niet altijd tot hetzelfde modus-cijfer vertaald. Bijvoorbeeld in `HIA.d` zal het eerste modus-cijfer 3 zijn; bij `BIG.d` of `SPR.d` zal het de waarde `2` hebben (bij deze instructies moet immers niet de inhoud van een geheugenregister opgehaald worden).

  Bij de invoer- en uitvoerbevelen (`INV` en `UTV`), bij de bevelen die het programmaonderbrekingsmasker plaatsen of testen (`MKH`, `MKL`, `TSM` en `TSO`) en bij `OND` mag geen interpretatie opgegeven worden. (Deze bevelen worden in het volgende hoofdstuk uitvoerig besproken.)

1. Het `EINDPR`-directief hoeft niet aanwezig te zijn. Het einde van het bestand waarin het programma opgeslagen is, geeft dan het einde van het programma aan. Indien het directief echter wel aanwezig is, zal alles wat volgt niet meer bekeken worden door de vertaler. Na dit directief mag de programmeur dus om het even wat schrijven: extra uitleg over het programma, testinvoer en verwachte resultaten, enz.

- De moeilijkste opdracht voor de vertaler is echter het behandelen van **symbolische adressen**. Zij zijn er de oorzaak van dat het vertaalprogramma niet meer in staat zal zijn elke lijn onmiddellijk te vertalen.

In de rest van deze paragraaf zullen we eerst het probleem van de ‘symbolische adressen’ aanbrengen, waarna we een gepaste oplossing zullen formuleren.

Symbolische adressen.

Indien het bronprogramma symbolische adressen bevat, kan het vertaalprogramma niet meer elke lijn onmiddellijk vertalen. De reden hiervoor is dat reeds naar bepaalde symbolische adressen verwezen kan worden nog vóór ze gedefinieerd zijn. Men spreekt in dit geval over ***voorwaartse referenties***. Hun tegenhangers (verwijzingen naar symbolische adressen die reeds gedefinieerd zijn) noemt men ***achterwaartse referenties***. Deze leveren geen problemen op.

Voorbeeld 3-3. *Beschouw het volgende* DRAMA*-programma dat de veelterm 6 X^4 + 3 X^2 - 4 evalueert voor X = 8 volgens het schema van Horner. (Het programma werd tamelijk algemeen gehouden zodat we gemakkelijk de coëfficienten van de veelterm en de waarde van de veranderlijke X kunnen wijzigen.*

```
(L1)  |
(L2)  | bereken Y = V(X) = 6X**4+3X**2-4  voor  X = 8
(L3)  |
(L4)       HIA     R5,N        | N = graad v/d veelterm
(L5)       HIA.w   R2,0        | R2 --> cumulatieve som
(L6)  LUS: VER     R2,X
(L7)       OPT     R2,V(R5-)   | verlaag R5 na gebruik
(L8)       VGL.w   R5,0
(L9)       VSP     GRG,LUS     | zolang R5 >= 0, ga verder
(L10)      BIG     R2,Y
(L11)      STP
(L12)   X: 8
(L13)   N: 4
(L14)   V: -4; 0; 3; 0; 6
(L15)   Y: RESGR   1           | plaats voor het resultaat
(L16) EINDPR
```

We hebben de lijnen van het programma genummerd om er gemakkelijk naar te kunnen verwijzen in het vervolg van de tekst. De nummers stellen geenszins adressen voor, vermits we ook de commentaarlijnen een nummer hebben gegeven.

Na het overslaan van de commentaarlijnen ***(L1)-(L3)***, leest de vertaler de eerste echte opdracht ***(L4)***:

`HIA R5,N`

- De decimale operatiecode voor '**`HIA`**' kan opgezocht worden in de DRAMA-***machine-instructietabel***[1], namelijk '**`11`**'.
- '**`R5`**' wordt vertaald tot '**`5`**'.
- Er zijn geen indexregisters dus het tweede modus-cijfer is '**`1`**'.
- De tweede operand is geen register, dus de verzuiminterpretatie is '**`.d`**', waardoor het eerste modus-cijfer '**`3`**' is.
- Alleen het adres-gedeelte van de te genereren machine-instructie is nog onbekend. De vertaler weet op dit ogenblik **niet** waar '**`N`**' zich ergens in het geheugen zal bevinden. Deze informatie is slechts beschikbaar na het lezen van lijn ***(L13)***. Het symbolisch adres '**`N`**' is in lijn ***(L4)*** een ***voorwaartse referentie***. Bijgevolg zal de vertaler in een eerste fase dit bevel slechts kunnen omzetten naar:

  `HIA R5,N` ☞ **`113150????`**

  De vraagtekens duiden op cijfers die nog niet gekend zijn. Dit bevel zal uiteindelijk in geheugenregister met adres **`0000`** geladen worden. De lezer stelt zich hierbij waarschijnlijk de vraag hoe deze vraagtekens in de bevelenopmaak kunnen opgenomen worden. Dit is eigenlijk niet mogelijk omdat elke combinatie van vier decimale cijfers als een adres kan geïnterpreteerd worden. We zullen hier dus later op moeten terugkomen; voorlopig laten we de vraagtekens staan.

De volgende lijn ***(L5)***, kan onmiddellijk omgezet worden tot een machine-instructie (alle gegevens zijn bekend). Ga zelf na dat de vertaling correct is:

`HIA.w R2,0` ☞ **`1111200000`**

Dit bevel zal geladen worden in geheugenregister met adres **`0001`**.

Lijn ***(L6)*** bevat opnieuw een voorwaartse referentie, namelijk het symbolisch adres '**`X`**', met als gevolg dat ook hier weer het adres-veld nog niet kan ingevuld worden. Bovendien wordt in deze lijn het symbolisch adres '**`LUS`**' gedefinieerd. Dit betekent dat '**`LUS`**' overeenkomt met het adres **`0002`**. Immers, de **`VER`**-instructie zal in geheugenregister met adres **`0002`** geladen worden. Indien '**`LUS`**' in een van de volgende lijnen gebruikt wordt, kan het assembleerprogramma dit symbolisch adres dus meteen vervangen door **`0002`**.

Figuur 3-5 (pag. 13) toont het verder verloop van de vertaling. De vraagtekens wijzen op informatie die nog niet gekend is.

Het '**`RESGR`**' directief kan het eenvoudigst vertaald worden naar een aantal nul-constanten, alhoewel dit strikt genomen niet noodzakelijk is.

1. Zie ook tabel A-1 (pag. 200).

(*Lijn*)	Programma in DRAMA-LPT			Adres:	Machinecode
(L4)		HIA	R5,N	*0000:*	113150????
(L5)		HIA.w	R2,0	*0001:*	1111200000
(L6)	LUS:	VER	R2,X	*0002:*	233120????
(L7)		OPT	R2,V(R5-)	*0003:*	213625????
(L8)		VGL.w	R5,0	*0004:*	3111500000
(L9)		VSP	GRG,LUS	*0005:*	3321200002
(L10)		BIG	R2,Y	*0006:*	122120????
(L11)		STP		*0007:*	9999999999
(L12)	X:	8		*0008:*	0000000008
(L13)	N:	4		*0009:*	0000000004
(L14)	V:	-4;0;3;0;6		*0010:*	9999999996
				0011:	0000000000
				0012:	0000000003
				0013:	0000000000
				0014:	0000000006
(L15)	Y:	RESGR	1	*0015:*	0000000000

FIGUUR 3-5. De code na de eerste vertalingsstap.

Tweestapsvertaling.

Uit het voorgaande zou duidelijk moeten zijn dat de vertaler heel wat opdrachten nog niet volledig kan vertalen. De ‘onbekenden’ zijn de verwijzingen (de voorwaartse referenties) naar symbolische adressen die verder in het programma gedefinieerd worden. Wanneer de vertaler het volledige programma gelezen heeft, dan zijn alle symbolische adressen gekend, en kunnen de ontbrekende stukken aangevuld worden. Om te weten waar precies adressen moeten ingevuld worden, zal de vertaler nog een keer het bronprogramma moeten overlopen[1].

Vermits er toch twee stappen (of fasen) nodig zijn, kunnen we de functies van elke stap beter gescheiden houden. (Dit zal dan ook het probleem van de vraagtekens oplossen.)

1. In de eerste stap zal de vertaler de absolute tegenhanger (d.i. het geheugenadres) van elk symbolisch adres bepalen, en opslaan in de ***symbooltabel***
2. In de tweede stap zal de vertaler de ***code genereren*** m.b.v. de symbooltabel.

1. Dit kan vermeden worden indien de vertaler tijdens de eerste stap extra informatie bijhoudt: bijvoorbeeld een extra tabel waarin voor elke onvolledige instructie het adres van die instructie en een verwijzing naar het nog niet gekende symbolisch adres staat.

In hetgeen volgt zullen we het absoluut adres dat met een bepaald symbolisch adres overeenkomt ook de ***waarde van dit symbolisch adres*** noemen. We kunnen het voorgaande ook als volgt parafraseren:

- de eerste fase behandelt de ***definities*** van de symbolische adressen;
- de tweede fase doet de eigenlijke vertaling, waarbij de ***referenties*** (dit zijn verwijzigingen) naar de symbolische adressen worden behandeld.

Aangezien na de eerste stap alle symbolische adressen bepaald zijn, kunnen er in de tweede stap geen onbekende voorwaartse referenties meer voorkomen (tenzij we een niet-gedefinieerd symbool zouden gebruikt hebben).

Het opstellen van de symbooltabel.

In de **eerste stap** zal de vertaler de waarde van alle symbolische adressen bepalen. Hiervoor heeft hij ***drie gegevensstructuren*** nodig:

1. een **lijnteller**, die de lijnen van de invoer telt, en onder meer gebruikt wordt om in foutboodschappen te verwijzen naar de foutieve lijn,
2. een **programmateller**[1], die het ***adres*** bijhoudt van het bevel (of constante) dat vertaald wordt, d.w.z. het adres van het geheugenregister waarin dit bevel (of constante) zal opgeslagen (of geladen) worden.
3. een **symbooltabel**, waarin de symbolische adressen en hun overeenkomstige waarde worden bijgehouden.

Het opstellen van de symbooltabel (zie bijvoorbeeld tabel 3-2 (pag. 15)) is het resultaat van een reeks bewerkingen. Hierna volgt wat het vertaalprogramma hiervoor moet doen.

De gegevenstructuren worden als volgt **geïnitialiseerd**:

1. de lijnteller krijgt initieel de waarde `1`,
2. de programmateller de waarde `0000` (of een andere waarde indien we het programma op een andere plaats in het geheugen willen laden[2]);
3. de symbooltabel is initieel leeg.

De vertaler zal de broncode **lijn per lijn inlezen**, waarbij voor elke lijn het volgende algoritme uitgevoerd wordt:

- ***Commentaarlijnen*** worden onmiddellijk genegeerd. Volledig blanco lijnen worden beschouwd als commentaarlijnen.

---

1. Deze programmateller is een interne variabele van de vertaler en mag niet verward worden met de bevelenteller van het besturingsorgaan.
2. Zie paragraaf 3.3 “De lader” op pag. 46 en volgende.

- In alle andere gevallen wordt nagekeken of er een symbolisch adres (ook etiket of label genoemd) ***gedefinieerd*** wordt (d.w.z. of er iets van de vorm **`NAAM`** ***gevolgd door een dubbele punt*** staat). Indien dit het geval is wordt dit symbolisch adres samen met de huidige waarde van de programmateller in de symbooltabel opgenomen.
  - Bij gewone DRAMA-***instructies*** wordt tenslotte de programmateller met één opgehoogd.
  - Bij ***constanten*** en ***geheugenreservaties***, echter, zal de programmateller met het aantal constanten respectievelijk het aantal reservaties opgehoogd worden.
- Na de behandeling van elke lijn, wordt de lijnteller met één opgehoogd.

Bij het toevoegen aan de symbooltabel moet wel nagekeken worden of het symbolisch adres niet eerder gedefinieerd was. In dit geval moet een foutboodschap uitgeschreven worden:

```
*** fout *** lijn 7: etiket `NAAM' is reeds vroeger gedefinieerd.
```

Na de eerste vertalingsstap bevat de symbooltabel alle symbolische adressen met hun overeenkomstige waarden. Voor ons modelprogramma zou de tabel er als volgt uitzien (indien de tabel tamelijk lang wordt, kan het nuttig zijn deze tabel te sorteren, zodat het zoeken in de tabel efficiënter kan gebeuren):

TABEL 3-2. De Symbooltabel (na stap 1).

Symbolisch adres	Adres
LUS	0002
X	0008
N	0009
V	0010
Y	0015

In de definitie van de DRAMA-taal werd gesteld dat een symbolisch adres ten hoogste uit vijftien symbolen mocht bestaan, waarvan het eerste geen cijfer is. In de symbooltabel zou je dus best plaats voorzien voor vijftien ASCII symbolen (elk voorgesteld door drie decimale cijfers); hiervoor volstaan dus vijf geheugenregisters. Niet-gebruikte posities worden met nullen gevuld. Ook voor het adresgedeelte zullen we een volledig geheugenregister voorzien, aangezien dit de kleinste adresseerbare geheugeneenheid is op de DRAMA-computer. Op de DRAMA-machine kan tabel 3-2 dus op de volgende wijze in het geheugen opgeborgen zijn (de matrix is volgens de kolom- of rijvoorstelling gelineariseerd):

TABEL 3-3. De symbooltabel in ascii-voorstelling

symbool				adres
0076085083	0000000000	...	0000000000	0000000000002
0088000000	0000000000	...	0000000000	0000000000008
0078000000	0000000000	...	0000000000	0000000000009
0086000000	0000000000	...	0000000000	0000000000010
0089000000	0000000000	...	0000000000	0000000000015

Dergelijke tabellen zijn echter niet erg handig voor een menselijke gebruiker. Daarom zullen we steeds de andere voorstelling (zoals in tabel 3-2) blijven gebruiken.

Codegeneratie.

In de **tweede stap** zal de vertaler opnieuw de bronbevelen lijn per lijn inlezen, en de bevelen omzetten naar machinebevelen. Alle symbolische referenties kunnen met behulp van de symbooltabel omgezet worden naar hun overeenkomstige waarde.
Ook tijdens deze fase kunnen zich fouten voordoen:

1. de gebruikte mnemotechnische code bestaat niet (bijvoorbeeld: **`HAI`**),
2. de syntaxis van de instructie is foutief (bijvoorbeeld: we geven slechts *één* operand op i.p.v. twee, of omgekeerd),
3. een operand kan betekenisloos zijn voor een bepaalde instructie, (bijvoorbeeld, een registeraanduiding als voorwaarde in een voorwaardelijke sprong: **`VSP R1,LUS`**),
4. er wordt een referentie naar een symbolisch adres gebruikt dat niet voorkomt in de symbooltabel (omdat het nergens gedefinieerd is).

In de tweede stap zal de vertaler voor het veelterm-evaluatieprogramma de machinecode genereren zoals die voorkomt in de meest rechtse (donkergekleurde) kolom van figuur 3-6 (pag. 17). De tabel zelf geeft het ***listingbestand*** weer dat voor de programmeur bedoeld is. Het toont duidelijk aan hoe elke lijn in het bronprogramma vertaald werd, en in welk geheugenregister het getal geladen moet worden. Het uitvoerbare programma zelf bestaat alleen uit de meest rechtse kolom. (Ga zelf na dat deze machinebevelen inderdaad correct zijn).

Op de listing merk je ook dat de inhoud van de symbooltabel werd afgedrukt. Vaak zal de vertaler deze symbooltabel ook achteraan toevoegen aan het uitvoerbaar programma. Deze symbooltabel wordt natuurlijk niet in het geheugen geladen wanneer het programma uitgevoerd wordt. Sommige programma's kunnen echter wel deze informatie nuttig gebruiken. Een voorbeeld is een ***speurprogramma*** (in het Engels '*debugger*') dat een uitvoerbaar programma onder de leiding van de gebruiker laat uitvoeren. Zo kan de programmeur aan het speurprogramma vragen om de uitvoering te stoppen bij '**`LUS`**'. Speurprogramma's worden besproken in paragraaf 3.6 (op pag. 86).

Samenvatting.

De aanwezigheid van symbolische adressen (voornamelijk als ze voorkomen als voorwaartse referenties) bemoeilijkt de vertaling van DRAMA-programma's. Hierdoor kan de vertaler niet meer elke ingelezen lijn van het bronprogramma onmiddellijk omzetten naar de overeenkomstige machinecode. Bijgevolg zal de vertaler in twee stappen werken:

1. in de eerste stap wordt de ***symbooltabel*** opgesteld, waarin alle symbolische adressen met hun overeenkomstige waarde worden bewaard;
2. in de tweede stap zal de vertaler m.b.v. deze symbooltabel de ***machinecode genereren***.

Paragraaf B.1 (op pag. 207) geeft in pseudo-Pascal-notatie de werking weer van de vertaler.

(Lijn)	Instructies in DRAMA-LPT	Adres:	Machinecode
(L1)	|		
(L2)	| bereken Y = V(X) = …		
(L3)	|		
(L4)	HIA R5,N	0000:	1131500009
(L5)	HIA.w R2,0	0001:	1111200000
(L6)	LUS: VER R2,X	0002:	2331200008
(L7)	OPT R2,V(R5-)	0003:	2136250010
(L8)	VGL.w R5,0	0004:	3111500000
(L9)	VSP GRG,LUS	0005:	3321200002
(L10)	BIG R2,Y	0006:	1221200015
(L11)	STP	0007:	9999999999
(L12)	X: 8	0008:	0000000008
(L13)	N: 4	0009:	0000000004
(L14)	V: -4;0;3;0;6	0010:	9999999996
		0011:	0000000000
		0012:	0000000003
		0013:	0000000000
		0014:	0000000006
(L15)	Y: RESGR 1	0015:	0000000000
	EINDPR		# symbolen
			LUS 0002
			X 0008
			N 0009
			V 0010
			Y 0015

FIGUUR 3-6. De code na de tweede vertalingsstap.

3.1.4 Vertaler voor een hogere programmeertaal

Programma's geschreven in een hogere programmeertaal, zoals C, moeten ook eerst omgezet worden naar machine-instructies alvorens ze kunnen uitgevoerd worden. De omzetting wordt door een **compilator** (in het Engels '*compiler*') verricht. Het is geenszins de bedoeling om hier de werking van een compilator in detail te bespreken. Deze wordt in andere cursussen behandeld.

Een hogere programmeertaal staat tamelijk ver van de laag-niveau-instructies die door de computer begrepen worden. In de oefenzittingen werd vaak vertrokken van een C-programma, dat dan omgezet werd naar het overeenkomstige DRAMA-programma. In heel wat gevallen is deze omzetting tamelijk eenvoudig, soms (denk aan procedure-oproepen) is er meer werk nodig. In figuur 3-7 worden enkele C-constructies omgezet naar equivalente DRAMA-instructies.

C	DRAMA		
int a, b;	a:	RESGR	1
	b:	RESGR	1
x = (4 * (y - z))		HIA.w	R1,4
/ (3 * (s - t)) - 7;		HIA	R2,y
		AFT	R2,z
		VER	R1,R2
		HIA.w	R2,3
		HIA	R3,s
		AFT	R3,t
		VER	R2,R3
		DEL	R1,R2
		AFT.w	R1,7
		BIG	R1,x
if (a > b)		HIA	R1,a
		VGL	R1,b
		VSP	KLG,ELSE3
max = a		HIA	R6,a
		BIG	R6,max
		SPR	ENDIF3
else max = b;	ELSE3:	HIA	R6,b
		BIG	R6,max
	ENDIF3:		

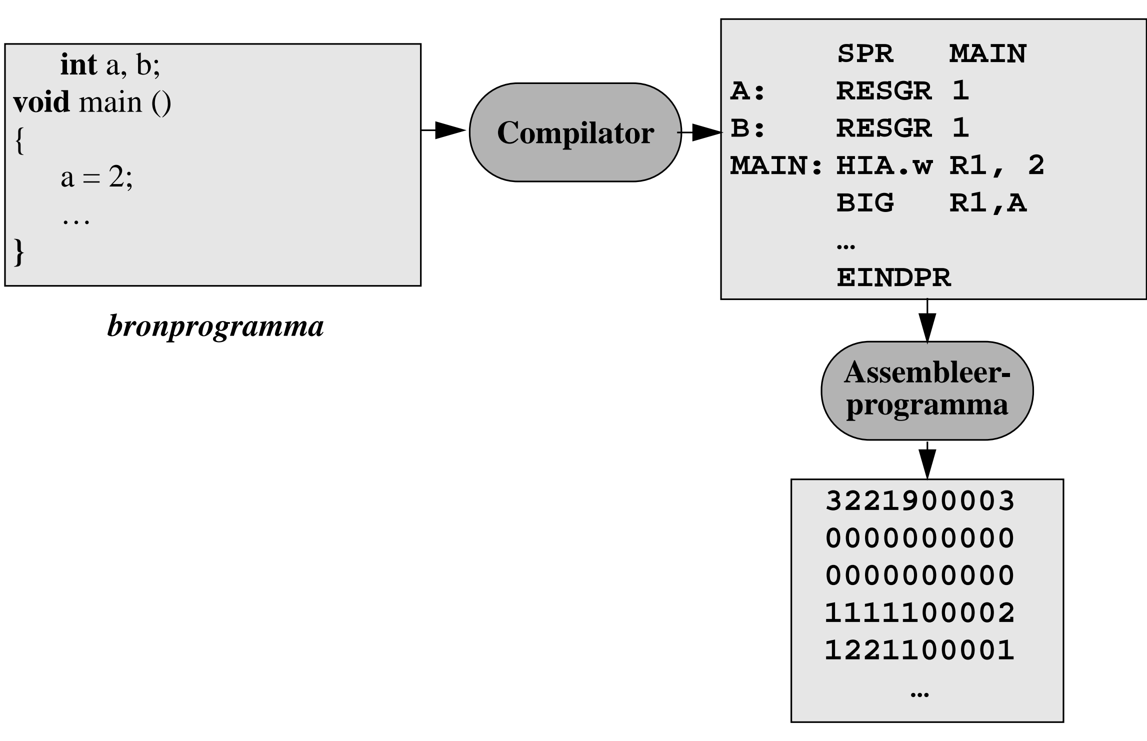
FIGUUR 3-7. Omzetting van C naar DRAMA.

Het spreekt vanzelf dat een compilator veel complexer is dan een assembleerprogramma. De **taken** die een compilator moet vervullen zijn:

1. het herkennen van de woorden van de taal (ook wel 'tokens' genoemd), zoals '`4`', '`*`', '`;`', '**if**', ...; dit noemt men de ***lexicale analyse***,
2. het herkennen van de individuele opdrachten, zoals '`b = 3*a;`'
   dit noemt men de ***syntactische analyse***, omdat elke opdracht in de taal een bepaalde structuur heeft, die bepaald wordt door de 'syntaxis' van de taal,
3. het begrijpen van de individuele opdrachten, zoals '`b = 3*a;`' ***betekent*** '3' vermenigvuldigen met de inhoud van '`a`' en het resultaat wegbergen in '`b`';
   dit noemt men de ***semantische analyse***, omdat de hogere programmeertaal voor elk 'soort' opdracht de 'betekenis' of 'semantiek' heeft vastgelegd,
4. tenslotte de equivalente ***code genereren.***

De compilator kan één of meer stappen nodig hebben om de omzetting te doen, afhankelijk van de complexiteit van de taal. Vaak zal de compilator eerst het bronprogramma omzetten naar een **boomstructuur** (de syntaxisboom) die gemakkelijker te behandelen is dan de oorspronkelijke invoer.

Alhoewel het perfect mogelijk is om onmiddellijk de inwendige (decimale) DRAMA-instructies te genereren, zal een compilator eerder **code genereren in een lagere programmeertaal** (of een intermediaire taal van laag niveau). Het resultaat moet dan vertaald worden door een assembleerprogramma (zie figuur 3-8).



FIGUUR 3-8. Van een C-programma tot een uitvoerbaar DRAMA-programma.

Dit heeft twee voordelen:

- de compilator is eenvoudiger uit te testen; het resultaat is leesbaar (een tekstbestand) en men kan het bijgevolg zelf nakijken,
- indien we een complexere DRAMA-II-machine zouden ontwerpen, waarvoor de inwendige bevelenopmaak een klein beetje verschilt van de oorspronkelijke DRAMA-machine, maar die dezelfde lagere DRAMA-programmeertaal heeft, hoeven we alleen het assembleerprogramma te wijzigen en niet de compilator.

De meeste compilatoren zullen automatisch na de omzetting van het bronprogramma naar een programma in lagere programmeertaal het assembleerprogramma oproepen, zodat het lijkt alsof de compilator zelf de machinecode gegenereerd heeft.

Een C-compilator voor de DRAMA-machine?!?

De allereerste compilatoren werden in een lagere programmeertaal geschreven. Vandaag worden compilatoren nog uitsluitend in een **hogere programmeertaal**[1] ontworpen. Wil je die compilator ook kunnen uitvoeren op je computer, dan moet deze eerst omgezet worden naar de inwendige machine-instructies van je computer.

Stel dat je voor de DRAMA-machine een C-compilator wil ontwikkelen en veronderstel dat je die in C hebt geschreven. Indien je de compilator op de DRAMA-machine wil uitvoeren, moet je hem eerst vertalen met … een C-compilator voor de DRAMA-machine! Maar die compilator heb je nog niet, want die ben je net aan het ontwikkelen. Gelukkig bestaan er vandaag reeds heel veel C-compilatoren (weliswaar niet voor de DRAMA-machine, maar wel voor tal van andere machines). Je kan dus je C-compilator (die in C is geschreven) vertalen met *één* van die andere compilatoren (bijvoorbeeld een C-compilator voor het LINUX-besturingssysteem op een PC). Wat is het resultaat van die vertaling? Een uitvoerbaar programma, dat C als invoer aanvaardt en DRAMA-programma's genereert. Dit programma kan echter alleen op de PC (met LINUX) uitgevoerd worden en niet op de DRAMA-machine! Een dergelijke vertaler wordt een ***kruiscompilator*** genoemd (in het Engels '*cross-compiler*'), omdat hij code genereert voor een andere machine dan deze waarop hij zelf uitgevoerd wordt. Met deze kruisvertaler kan je dus op je PC elk C-programma omzetten naar een DRAMA-programma dat je dan overbrengt naar de DRAMA-machine om het verder te vertalen en uit te voeren.

Indien je nu opnieuw je C-compilator (oorspronkelijk geschreven in C) laat vertalen door deze kruiscompilator, dan … bekom je een C-compilator, die:

- DRAMA-code genereert en
- na vertaling door het assembleerprogramma, kan uitgevoerd worden op je DRAMA-machine!

1. Vaak gebruikt men een taal van ***zeer hoog niveau***. Speciale programma's, ook wel compiler-compilers genoemd, zetten deze hoog-niveaubeschrijvingen om naar een programma, geschreven in een van de conventionele hogere programmeertalen, zoals Pascal, C, C++ of Java.

Opgaven

1. Leg het verschil uit tussen bronprogramma en uitvoerbaar programma. Welke omzetting moet er gebeuren?
2. Beschrijf bondig de werking van de MINI-DRAMA-vertaler. Pas dit toe op het volgende MINI-DRAMA-programma: (Gebruik hiertoe de tabellen uit appendix A.)

```
| Druk -20000 * A + B af
| met A in 0100, B in 0200
HIA    R0,6
VER    R0,100
OPT    R0,200
DRU
NWL
STP
-20000
EINDPR
```

3. Som enkele redenen op waarom de DRAMA-vertaler heel wat complexer is dan de MINI-DRAMA-vertaler.
4. Wat is de betekenis van ‘stap’ in ‘de vertaler heeft twee stappen nodig om een programma te vertalen’?
5. Waarom kan de MINI-DRAMA-vertaler in één stap een MINI-DRAMA-programma vertalen terwijl de DRAMA-vertaler vaak twee stappen nodig heeft om een DRAMA-programma te vertalen?

   Welke beperking(en) zou je moeten opleggen aan DRAMA-programma’s opdat de DRAMA-vertaler ook maar één stap nodig zou hebben?
6. Wat zijn voorwaartse en achterwaartse referenties? Welke zijn het moeilijkst te behandelen door de vertaler?
7. Beschrijf bondig de taken die de DRAMA-vertaler in elke stap uitvoert.
8. Wat is een symbooltabel? Waartoe dient ze? Welke informatie bevat ze? In welke stap wordt ze ingevuld?

   Wat is de symbooltabel voor het volgende DRAMA-programma?

```
|
| Bereken en druk N! af voor N = 0, ..., 10
|
FAC:    HIA.w   R1,0        | N bijhouden in R1
        HIA.w   R2,1        | N! berekenen in R2
LUS:    HIA     R0,R1
        DRU     | Druk N
```

```
        HIA     R0,R2
        DRU                | Druk N!
        NWL
        VGL.w   R1,10      | N >= 10?
        VSP     GRG,EINDE
        VER.w   R2,0(+R1)  | N = N+1; N! = N! * N
        SPR     LUS
EINDE:  STP
EINDPR
```

9. Genereer (m.b.v. de tabellen van appendix A (op pag. 199 e.v.)) de code voor het **FAC** programma uit vraag 8.

10. Wat is een compilator? Welke taken moet een compilator uitvoeren?

11. Wat betekenen de volgende termen:

- semantische analyse,
- lexicale analyse en
- syntactische analyse?

12. Je beschikt over een Fortran- en een C-compilator voor DOS op je PC. Je zou graag een Fortran-compilator voor de DRAMA-machine hebben. Hoe ga je te werk?
Hoe ver geraak je als je enkel over een C-compilator beschikt?

13. De methode die voorgesteld werd om een C-compilator te schrijven voor de DRAMA-machine gaat niet op voor het DRAMA-assembleerprogramma. Dit assembleerprogramma kan je wel in C schrijven en nadien vertalen op een PC, maar verder dan een kruisvertaler kom je niet. Bedenk zelf een werkwijze om een DRAMA-assembleerprogramma te bekomen dat rechtstreeks uitgevoerd kan worden op de DRAMA-machine.

3.2 De MACRO-voorvertaler

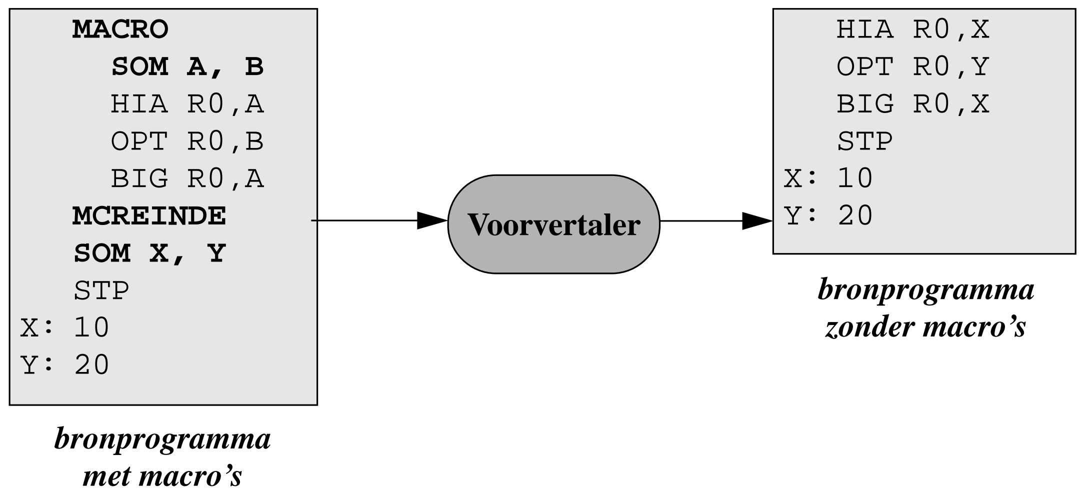
Macro's werden ingevoerd als een hulpmiddel bij het programmeren. De programmeur hoeft veelvoorkomende stukjes code slechts eenmaal te schrijven (en te definiëren als een macro); daarna kan dit codefragment op elke plaats ingelast worden door deze macro op te roepen.

De taak van de ***voorvertaler*** (*'pre-processor'* in het Engels) bestaat erin de **macrodefinities** te verwijderen uit het programma, en alle **macro-oproepen** te vervangen door de overeenkomstige code. Dit laatste wordt ook ***macro-expansie*** genoemd.

In principe zou de vertaler ook de macro-verwerking op zich kunnen nemen. Dit heeft het voordeel dat betekenisvollere foutboodschappen kunnen gegeven worden. Nochtans bestaan er goede redenen om een aparte voorvertaler te voorzien:

- Er bestaat een groot verschil tussen 'macroverwerking', wat een vorm van ***symboolmanipulatie*** is, en 'codegeneratie'. Vanwege de modulariteit is het aangewezen om die verschillende taken gescheiden te houden.
- Macro's kunnen in om het even welke context gebruikt worden, dus ook in andere dan DRAMA-programma's. Voor een voorbeeld hiervan, zie appendix C (op pag. 216).

De werking van de voorvertaler komt grosso modo neer op ***symboolmanipulatie***, d.w.z. het kopiëren van tekens (symbolen) of het vervangen van een rij van tekens door een sequentie van andere tekens. (Zie ook figuur 3-9.)



FIGUUR 3-9. De macrovoorvertaler

Indien alleen de directieven **MACRO** en **MCREINDE** voorzien worden[1], wordt de taak van de voorvertaler sterk vereenvoudigd. We zullen eerst dit geval bespreken, en daarna aangeven hoe de voorvertaler kan uitgebreid worden met de extra directieven, die ook voorwaardelijke macro-opbouw en recursieve macro-oproepen moeten mogelijk maken.

1. Dus geen **MEVA**, **MVGL**, **MVSP**, **MSPR**, **MNTS** en **MFOUT**.

3.2.1 Enkel MACRO- en MCREINDE-directieven.

In figuur 3-10 staat een eenvoudig DRAMA-programmaatje waarin een macro '**SOM**' eerst gedefinieerd wordt en daarna opgeroepen wordt.

```
(L1)  | Begin van het programma
(L2)  MACRO
(L3)          SOM     RES,A,N
(L4)          | <RES> := som(i=0 tot <N>-1) <A>[i]
(L5)          HIA.w   R0,0
(L6)          HIA.w   R1,0
(L7)   $LUS:  OPT     R0,<A>(R1+)
(L8)          VGL.w   R1,<N>
(L9)          VSP     KL,$LUS
(L10)         BIG     R0,<RES>
(L11) MCREINDE
(L12) |  Begin van het programma
(L13)         SOM     C,X,3
(L14)         HIA     R0,C
(L15)         DRU
(L16)         STP
(L17)     C:  RESGR   1
(L18)     X:  10;20;30
(L19) EINDPR
```

FIGUUR 3-10. Een programma met een macrodefinitie en -oproep.

De voorvertaler moet ***drie verschillende taken*** kunnen uitvoeren:

1. alles wat niets met macro's te maken heeft (d.w.z. geen macrodefinitie of geen macro-oproep is) gewoon kopiëren,
2. macrodefinities verwijderen uit het bronbestand en ze tijdelijk bewaren (zodat een latere macro-oproep correct geëxpandeerd kan worden),
3. macro-oproepen expanderen (d.w.z. vervangen door de *eventueel lichtjes gewijzigde* macrodefinitie).

Om duidelijk te maken wat de voorvertaler op elk ogenblik precies moet doen, zullen we drie verschillende toestanden voorzien: ***kopieermodus***, ***definitiemodus*** en ***expansiemodus***. In elke toestand zal de voorvertaler iets anders doen. De overgang van de ene toestand naar de andere wordt geregeld door sleutelwoorden (bijvoorbeeld een voorvertalerdirectief zoals **MACRO** of de naam van een eerder gedefinieerde macro) of door het einde van de invoer of het einde van bewaarde gegevens (bijvoorbeeld het einde van een opgeslagen macrodefinitie).

De voorvertaler start zijn uitvoering in de kopieermodus.

1. In de **kopieermodus** zal de voorvertaler de invoer lijn per lijn inlezen, en in de meeste gevallen de lijn letterlijk kopiëren naar het uitvoerbestand. Er zijn drie mogelijkheden:

a. ofwel bevat de lijn het `MACRO`-directief. Dit directief duidt aan dat een nieuwe macro gedefinieerd zal worden. De voorvertaler gaat bijgevolg naar de definitiemodus.

b. de lijn kan ook van de volgende vorm zijn:

```
XXX   arg ...
```

waarbij '`XXX`' de naam is van een eerder gedefinieerde macro.

De voorvertaler zal het verband vastleggen tussen de formele parameters van de macro en de ***actuele parameters***. Dit verband wordt in een aparte gegevensstructuur, de ***argumententabel***, bijgehouden (per macro-oproep wordt een nieuwe argumententabel opgesteld). De voorvertaler gaat nu naar de expansiemodus, aangezien de macro-oproep moet geëxpandeerd worden.

c. in alle andere gevallen wordt de lijn letterlijk gekopieerd naar het uitvoerbestand.

In het voorbeeld zal de eerste lijn `(L1)` gewoon gekopieerd worden naar het uitvoerbestand. De tweede lijn `(L2)` bevat het `MACRO`-directief; deze lijn wordt niet gekopieerd maar de voorvertaler gaat wel naar de definitiemodus.

2. In de **definitiemodus** wordt een nieuwe macro gedefinieerd. De eerstvolgende lijn die gelezen wordt in deze toestand bevat de ***hoofding*** van de macro en bepaalt de naam van de macro, alsook de namen van de argumenten. De voorvertaler zal een gegevensstructuur creëren waarin deze hoofding en het ***lichaam*** opgeslagen zullen worden. Voor de volgende lijnen zijn er twee mogelijkheden:

a. de lijn bevat het `MCREINDE`-directief. Dit betekent dat de macrodefinitie beëindigd is. De voorvertaler gaat terug naar de kopieermodus.

b. in alle andere gevallen wordt de lijn zonder meer toegevoegd aan de gegevensstructuur waarin de macrodefinitie opgeslagen wordt.

In het voorbeeld zal de voorvertaler lijn `(L3)` analyseren als de hoofding van de nieuwe macrodefinitie '`SOM`', en een gegevensstructuur creëren waarin de macrodefinitie opgeslagen kan worden. Lijnen `(L4)` tot en met `(L10)` behoren tot het lichaam van de macro. Figuur 3-11 toont schematisch de gegevensstructuur die door de voorvertaler voor de macro '`SOM`' wordt bijgehouden. Naast de ***informatie uit de hoofding*** (naam van de macro '`SOM`' en de namen van de formele parameters '`RES`', '`A`' en '`N`'), wordt een ***exacte kopie van het lichaam*** van de macro bijgehouden. Het 'lengteveld' wordt na het lezen van het `MCREINDE`-directief ingevuld en geeft aan uit hoeveel lijnen het lichaam van de macro bestaat. In dit voorbeeld, is het lichaam `7` lijnen lang.

.

SOM	RES A N	**lengte: 7**

```
| <RES> := som(i=0 tot <N>-1) <A>[i]
             HIA.w     R0,0
             HIA.w     R1,0
$LUS:        OPT       R0,<A>(R1+)
             VGL.w     R1,<N>
             VSP       KL,$LUS
             BIG       R0,<RES>
```

FIGUUR 3-11. Gegevensstructuur voor de macro SOM

Bij het lezen van lijn `(L11)`, het `MCREINDE`-directief, zal de voorvertaler de lengte van de macrodefinitie invullen (7), en terug naar de kopieermodus overgaan. Het directief zelf wordt niet toegevoegd aan de definitie. **Vanaf nu** is de naam van de gedefinieerde macro (i.c. `SOM`) een **sleutelwoord**[1] voor de voorvertaler. Telkens wanneer deze naam voorkomt als eerste woord op een lijn, zal de voorvertaler dit interpreteren als een macro-oproep. De taal (dit is de verzameling sleutelwoorden) die de voorvertaler begrijpt, breidt zich dus tijdens de uitvoering van de voorvertaler uit: telkens een nieuwe macro gedefinieerd is, heeft de voorvertaler een nieuw sleutelwoord bijgeleerd!

Lijn `(L12)` wordt letterlijk gekopieerd naar het uitvoerbestand.

Bij het lezen van lijn `(L13)` detecteert de voorvertaler het sleutelwoord `SOM` (dit is de naam van een eerder gedefinieerde macro) als eerste woord op deze lijn. De overeenkomst tussen formele en actuele parameters wordt in een ***argumententabel*** opgeslagen (de formele namen vindt de voorvertaler in de gegevensstructuur waarin de definitie van `SOM` opgeslagen is):

TABEL 3-4. De argumententabel.

formele naam	actuele parameter
RES	C
A	X
N	3

Zoals eerder vermeld, wordt voor elke oproep van '`SOM`' een nieuwe argumententabel opgesteld (dit is belangrijk indien we te maken hebben met recursieve macro-oproepen). Na het opstellen van de argumententabel gaat de voorvertaler over naar de expansiemodus.

1. De voorvertalerdirectieven MACRO en MCREINDE zijn ook sleutelwoorden (*keywords* in het Engels) voor de voorvertaler.

3. In de **expansiemodus** wordt een macro-oproep vervangen door het (eventueel licht gewijzigde) lichaam van die macro. De wijzigingen worden veroorzaakt door de parameters en door eventuele genestelde macro-oproepen. In plaats van het invoerbestand te lezen, zal de voorvertaler nu het in de gegevensstructuur opgeslagen lichaam lijn per lijn lezen. Er moeten twee soorten substituties doorgevoerd worden:

a. Elk symbool van de vorm **<NAAM>** wordt vervangen door de overeenkomstige 'actuele parameter' uit de argumententabel. Bijvoorbeeld, de eerste lijn van het lichaam van '**SOM**':

```
| <RES> := som(i=0 tot <N>-1) <A>[i]
```

zal gewijzigd worden tot:

```
| C := som(i=0 tot 3-1) X[i]
```

Zoals je merkt, worden zelfs commentaarlijnen mee geëxpandeerd. De voorvertaler kent immers het verschil niet tussen een gewone DRAMA-opdracht en een commentaarlijn. Hij herkent immers alleen macrodefinities (die beginnen bij het sleutelwoordje '**MACRO**' en eindigen bij '**MCREINDE**') en macro-oproepen (van reeds gedefinieerde macro's). Al het overige wordt door hem als een willekeurige symbolenrij beschouwd.

Indien de naam van de parameter niet voorkomt in de argumententabel (d.w.z. de gebruikte symbolische naam is geen parameter), dan zal de voorvertaler een foutboodschap afdrukken.

b. Een symbool van de vorm **$ETIKET** is een ***lokaal symbolisch adres***, en moet vervangen worden door een ***uniek globaal symbolisch adres***.

Hier stelt zich dus een probleem: het etiket '**$ETIKET:**' kan in verschillende macrodefinities voorkomen; ook de naam '**ETIKET**' zelf kan ergens in het hoofdprogramma voorkomen om een geheugenregister aan te duiden. Bovendien kunnen de macro's waarin '**$ETIKET**' voorkomt verschillende keren opgeroepen worden. Na de macrovoorvertaling moeten echter al deze etiketten ***unieke*** (d.i. verschillende) namen geworden zijn. Volgens de syntaxisbeschrijving van DRAMA, moeten alle namen beginnen met een letter. Ten behoeve van de macrobehandeling wordt ook het onderlijningsstreepje '_' als een letter beschouwd. De macrovoorvertaler zal bovendien een interne teller bijhouden die bij elke macro-oproep opgehoogd wordt. Unieke namen worden nu als volgt gegenereerd: het **$**-teken in '**$ETIKET**' wordt vervangen door ***_teller***, waarbij ***teller*** de waarde van de interne macro-oproepteller is. Bijvoorbeeld, de derde lijn van het lichaam van '**SOM**':

```
$LUS:   OPT R0,<A>(R1+)
```

zal gewijzigd worden tot:

```
_1LUS:  OPT R0,X(R1+)
```

omdat dit de eerste macro-oproep is. Indien de macro een tweede keer zou opgeroepen worden (wat in dit voorbeeld niet het geval is), dan zou de voorvertaler het etiket '**$LUS**' omzetten tot het etiket '**_2LUS**', enzovoort.

Na het uitvoeren van alle substituties, zal de voorvertaler nagaan wat er met de resulterende lijn moet gebeuren. Er zijn weer twee mogelijkheden:

a. ofwel is de lijn een macro-oproep (een ingenestelde macro-oproep). In dit geval zal de voorvertaler tijdelijk de expansie van de huidige macro uitstellen, en eerst de nieuwe macro-oproep expanderen. Dit houdt in dat de voorvertaler bijhoudt waar hij gekomen is

bij de expansie van de huidige macro-oproep, daarna een nieuwe argumententabel opstelt, en tenslotte begint met het lezen van het lichaam van de zojuist opgeroepen macro. Uiteraard blijft de voorvertaler in de expansiemodus. Na de expansie van deze laatste macro-oproep (en eventueel alle verdere ingenestelde macro-oproepen) zal hij verdergaan met de huidige expansie.

b. in alle andere gevallen wordt de gewijzigde lijn naar het uitvoerbestand gekopieerd.

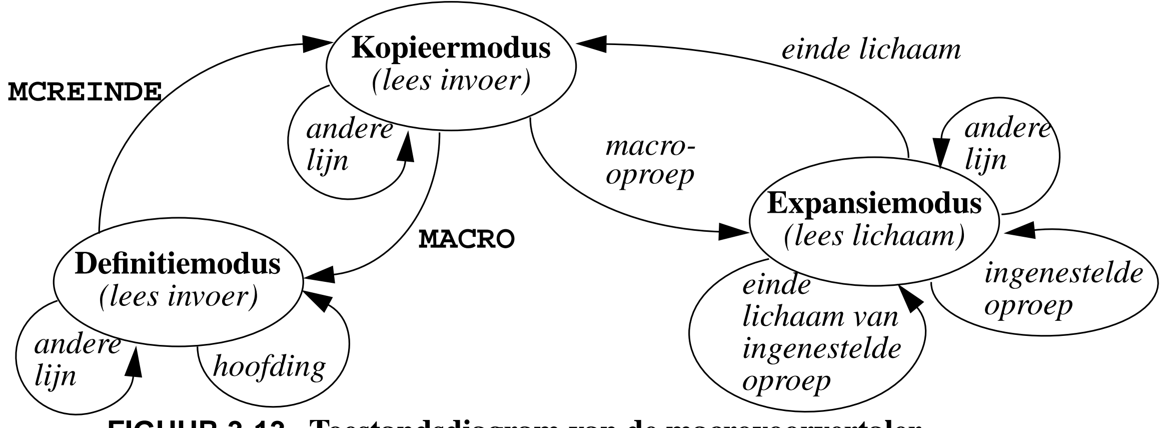
Indien het volledige lichaam gelezen is, wordt de argumententabel vernietigd en indien deze expansie geen genestelde expansie was, gaat de voorvertaler terug naar de kopieermodus.

Het uitvoerbestand voor het voorbeeld ziet er dus uit zoals weergegeven is in figuur 3-12.

```
|  Begin van het programma
| C := som(i=0 tot 3-1) X[i]
          HIA.w   R0,0
          HIA.w   R1,0
  _1LUS:  OPT     R0,X(R1+)
          VGL.w   R1,3
          VSP     KL,_1LUS
          BIG     R0,C
          HIA     R0,C
          DRU
          STP
      C:  RESGR   1
      X:  10;20;30
EINDPR
```

FIGUUR 3-12. Het resultaat van de MACRO-voorvertaling.

Figuur 3-13 geeft het ***toestandsdiagram*** voor de macrovoorvertaler weer. Het beschrijft de verschillende toestanden (modi) en de mogelijke overgangen; naast elke overgang wordt de ***oorzaak*** voor die overgang geschreven.



FIGUUR 3-13. Toestandsdiagram van de macrovoorvertaler

3.2.2 Extra macrodirectieven

De extra macrodirectieven **MEVA**, **MNTS**, **MFOUT**, **MSPR**, **MVSP** en **MVGL** werden voorzien om aan voorwaardelijke macro-opbouw te kunnen doen. Het is echter interessant ze ook toe te laten in het hoofdprogramma, dus zowel binnen als buiten een macrodefinitie. Het worden dus gewoon voorvertalerdirectieven. Tabel 3-5 geeft een overzicht van alle beschikbare voorvertalerdirectieven met een aanduiding van hun toepassingsgebied.

TABEL 3-5. Macrodirectieven, hun context en betekenis

Directief	Ook buiten macro definitie?	Betekenis
MACRO	uitsluitend	begin nieuwe macrodefinitie
MCREINDE	neen	beëindig een macrodefinitie
MEVA NAAM,expressie	ja	**NAAM** := **expressie** en geef **MCC** de gepaste waarde
MFOUT boodschap	ja	druk boodschap af en stop
MSPR $ETIKET	ja	ga verder bij **$ETIKET**
MVSP voorw,$ETIKET	ja	ga verder bij **$ETIKET** indien **MCC** aan **voorw** voldoet
MVGL expr1,expr2	ja	vergelijk beide expressies en geef **MCC** de gepaste waarde
MNTS	ja	negeer directief (nuttig om een etiket voor te plaatsen)

Zoals uit de tabel blijkt mogen de meeste directieven zowel **binnen** als **buiten** een macrodefinitie gebruikt worden. Dit verhoogt het ‘programmeercomfort’. Samen met die directieven voegen we nog twee faciliteiten toe:

- het gebruik van globale voorvertalervariabelen,
- het gebruik van globale voorvertaleretiketten.

We zullen eerst deze nieuwe mogelijkheden bespreken. Daarna zullen we aangeven welke gegevensstructuren de voorvertaler hiervoor zal bijhouden en welke aanpassingen aan het ontwerp van de voorvertaler nodig zijn. De voorvertaler zal dus iets complexer worden.

Lokale macrovariabelen en globale voorvertalervariabelen

Het **MEVA**-directief kent een nieuwe waarde toe aan een macrovariabele of aan een formele parameter van een macro. Tot hiertoe waren alle variabelen lokaal t.o.v. de macro waarin ze gebruikt werden. Er bestaan echter goede redenen om naast deze lokale macrovariabelen, ook globale voorvertalervariabelen te voorzien:

- een lokale macrovariabele houdt op te bestaan na de macro-oproep, terwijl een globale variabele zijn waarde blijft behouden; dit kan nuttig zijn om doorlopende tellers of doorlopende wijzers te implementeren;
- een globale variabele kan door meerdere macro's gebruikt worden;
- een globale variabele kan ook buiten een macro gebruikt worden (bijvoorbeeld om programmaconstanten voor te stellen; zie verder).

Opgelet, met globale variabelen bedoelen we hier variabelen die enkel betekenis hebben voor de voorvertaler en dus niet de gewone variabelen die in het programma voorkomen en waarvoor een geheugenplaats voorzien is (bijv. via het **RESGR**-directief). Daarom zullen we in het vervolg liever spreken over ***globale voorvertalervariabelen*** (afgekort tot VV-variabelen). Een ***lokale macrovariabele*** is ook een voorvertalervariabele, maar ze bestaat enkel gedurende de expansie van een oproep van de macro waarbinnen ze gedefinieerd is.

Impliciet hebben we reeds een globale VV-variabele gezien: de ***teller*** die verhoogd wordt bij elke nieuwe macro-oproep en die gebruikt wordt om unieke symbolische adressen te genereren.

Figuur 3-14 illustreert het gebruik van globale VV-variabelen om programmaconstanten voor te stellen. In programma's komen vaak constanten voor. In DRAMA kunnen kleine constanten als directe operand ingelast worden in een instructie. Het nadeel van deze werkwijze is wel dat als je later die constanten wil aanpassen (bijvoorbeeld het aantal stappen wil verhogen van 50 naar 100, of de sentinelwaarde 0 maken i.p.v. -1, of de nilwijzer wil voorstellen door 0 i.p.v. -1, ...) dat je dan je ganse programma moet nakijken. Het is echter niet altijd evident welke constanten je dan moet aanpassen. Indien op verschillende plaatsen in je programma de waarde '0' gebruikt wordt, gaat het dan om de sentinelwaarde of de nilwijzer, of misschien gewoon om de getalwaarde 0? Aan een constante waarde kun je immers niet zien wat zijn betekenis is! Daarom is het beter een aantal globale voorvertalervariabelen te voorzien die deze constanten voorstellen. In dit voorbeeld voorzien we drie constanten, die we met de symbolische namen **N**, **NOK** en **NIL** voorstellen. In het programma gebruiken we nergens de constante waarden, maar wel de symbolische namen **<N>**, **<NOK>** en **<NIL>**. Willen we nu het aantal stappen verhogen, dan hoeven we enkel de eerste **MEVA** aanpassen. Overal zal de correcte waarde gebruikt worden.

```
| Definitie van enkele constanten
|    N:   Aantal Stappen
|    NOK: Sentinel-waarde
|    NIL: Nil-wijzer
        MEVA      N,50
        MEVA      NOK,-1
        MEVA      NIL,-1
| Hoofdprogramma
        HIA.w     R1,<N>
        ...
        LEZ
        VGL.w     R0,<NOK>
        ...
        HIA.w     R6,<NIL>
        BIG       R6,A+1
        ...
        VGL.w     R1,<N>
        ...
```

FIGUUR 3-14. Globale variabelen gebruikt als programma-constanten.

Zoals uit dit voorbeeld blijkt, bestaat er **geen** directief voor het declareren van VV-variabelen in DRAMA. VV-variabelen bestaan zodra er voor de eerste keer een waarde aan toegekend wordt via het '**MEVA**'-directief. Zodra een toekenning gebeurt aan een VV-variabele ***buiten de context van een macro*** wordt deze VV-variabele **globaal** (en blijft ze voor altijd globaal). Zolang er ***enkel toekenningen*** gebeurd zijn ***tijdens een macro-expansie***, is de VV-variabele **lokaal** en noemen we ze ook een macrovariabele; na de macro-oproep houdt de variabele op te bestaan.

Figuur 3-15 illustreert een andere toepassing van globale VV-variabelen. We definiëren drie macro's **IF**, **THEN** en **ENDIF** die samen een **if-then-endif**-constructie mogelijk maken. We hebben hier een globale VV-variabele nodig omdat we unieke etiketten moeten vormen waarnaar verwezen wordt vanuit andere macro's dan deze waarin ze gedefinieerd worden. In dit voorbeeld wordt er vanuit de **IF**-macro verwezen naar etiketten die pas in **THEN** en **ENDIF** gedefinieerd worden. We kunnen dus geen gebruik maken van lokale macro-etiketten. De **IF**-macro begint met het verhogen van de variabele **L**. Op die manier zijn we ervan verzekerd dat elke **if-then-endif**-constructue een ***unieke set*** van etiketten zal gebruiken. Met behulp van deze variabele zullen twee etiketten gedefinieerd worden (in de macro's **THEN** en **ENDIF**), namelijk **THEN<L>** en **END<L>**. Naar deze etiketten wordt er echter reeds verwezen in de **IF**-macro!

.

```
MEVA        L,0
-----------------------------------
MACRO
IF          A,VW,B
            MEVA   L,<L>+1
            HIA    R0,<A>
            VGL    R0,<B>
            VSP    <VW>,THEN<L>
            SPR    END<L>
MCREINDE
-----------------------------------
MACRO
THEN
 THEN<L>:   SPR    THEN<L>+1
MCREINDE
-----------------------------------
MACRO
ENDIF
 END<L>:    SPR    END<L>+1
MCREINDE
-----------------------------------
            | Hoofdprogramma
            IF     A,GR,B
            THEN
            BIG    R0,MAX
            ENDIF
            ...
            IF     C,GRG,MAX
            THEN
      LUS:  AFT    R0,MAX
            ...
            ENDIF
            ...
```

FIGUUR 3-15. Gebruik van een globale VV-variabele voor het genereren van unieke etiketten.

Merk op dat de **if-then-endif**-constructie, zoals ze hier gedefinieerd is, niet mag ineengenesteld worden (tussen de **THEN** en de **ENDIF** mag je geen nieuwe **if-then-endif**-constructie inlassen). Er is ook geen **else**-gedeelte voorzien. De macro's zijn ook niet echt robuust geschreven. Er wordt bijvoorbeeld niet getest of een **THEN** door een **IF** is voorafgegaan, en of

er zich geen ineennesteling voordoet. We laten deze uitbreidingen over aan de lezer als oefening. (Zie ook oefening 12 (op pag. 43)).

Figuur 3-16 toont het resultaat van de voorvertaling van figuur 3-15.

```
        | Hoofdprogramma
        HIA   R0,A
        VGL   R0,B
        VSP   GR,THEN1
        SPR   END1
THEN1:  SPR   THEN1+1
        BIG   R0,MAX
END1:   SPR   END1+1
        ...
        HIA   R0,C
        VGL   R0,MAX
        VSP   GRG,THEN2
        SPR   END2
THEN2:  SPR   THEN2+1
LUS:    AFT   R0,MAX
        ...
END2:   SPR   END2+1
        ...
```

FIGUUR 3-16. Voorvertaling van programma uit figuur 3-15.

De sprongbevelen die naast de etiketten **THEN1** en **END1** staan, zijn zogenaamde **NOP**-bevelen[1] — dit zijn bevelen die geen enkel effect hebben. We kunnen deze bevelen vermijden door de etiketten in de macrodefinities van **THEN** en **ENDIF** te schrijven vóór het **MCREINDE** directief. Het nadeel is dan wel dat de eerste instructie die een **THEN**- of een **ENDIF**-oproep volgt zelf geen etiket mag bevatten (zoals hier het geval is in het tweede **then**-gedeelte, waar reeds het etiket **LUS** gebruikt wordt), tenzij de vertaler toelaat dat een instructie door meer dan een etiket mag voorafgegaan worden.

1. Een NOP-bevel of No-OPeration bevel is een bevel dat niets doet. Het kan dus op elke plaats ingelast worden zonder dat het resultaat van het programma gewijzigd wordt. Aangezien de DRAMA-machine geen NOP-bevel kent, moeten we het simuleren m.b.v. een sprongbevel naar de volgende instructie.

Lokaal, Globaal of Onzichtbaar

Het is mogelijk dat ***tijdens*** een macro-expansie een VV-variabele lokaal is (d.w.z. een macro-variabele is), terwijl bij een latere expansie dezelfde variabele globaal is, omdat er tussen de twee oproepen een 'globale toekenning' gebeurd is. Of dergelijke constructies al dan niet nuttig zijn, laten we hier buiten beschouwing. We gaan hier niet verder op in. Een voorbeeld kan je vinden in de oefeningen (oefening 13 (op pag. 43)). Om dit soort dubbelzinnigheden te vermijden, doe je er best aan om alle globale VV-variabelen een waarde te geven (m.b.v. **MEVA**) helemaal vooraan in het programma, dus ***vóór*** de macrodefinities of programmacode.

Globale variabelen zijn niet altijd ***zichtbaar*** binnen een macro. Indien een van de formele parameters van de macro dezelfde naam heeft als een globale VV-variabele, dan kan je binnen de macro die globale VV-variabele niet gebruiken.

Gegevenstructuren voor het bijhouden van VV-variabelen

Figuur 3-17 toont een eenvoudig DRAMA-programmaatje waarin zowel globale VV-variabelen (**NIL** en **N**) als een macrovariabele (**IDX**) gebruikt wordt.

```
MEVA        NIL,-1
MEVA        N,20
MACRO
      SET   REG,X,I,LEN,INT
            MEVA    IDX,<I>*<LEN>
            HIA.w   <REG>,<INT>
            BIG     <REG>,<X>(<IDX>)
MCREINDE
            | Hoofdprogramma
            ...
            SET     R0,A,3,2,<NIL>
            ...
            SET     R3,B,7,10,100
            ...
            STP
       A:   RESGR   <N>*2
       B:   RESGR   <N>*10
EINDPR
```

FIGUUR 3-17. Gebruik van VV-variabelen en macrovariabelen.

De globale VV-variabelen worden bijgehouden in de **globale-VV-variabelentabel** (afgekort tot GVT). Bijvoorbeeld, na de uitvoering van het macrodirectief **`MEVA N,20`** (zie licht gekleurde lijn in figuur 3-17) ziet deze tabel er als volgt uit:

TABEL 3-6. Globale VV-variabelen tabel

Naam	Waarde
NIL	-1
N	20

De lokale VV-variabelen, d.w.z. de macrovariabelen, worden bijgehouden in een lokale-macro-variabelentabel en zo is er één per macro-oproep. Aangezien we per macro-oproep ook een argumententabel hebben, zullen we deze twee tabellen samenvoegen tot de **argumenten-en-lokale-variabelentabel** (afgekort tot ALVT). Links wordt de inhoud van deze tabel getoond op het einde van de eerste macro-expansie, rechts op het einde van de tweede expansie van '**`SET`**'. De eerste vijf rijen in de tabel stellen de parameters voor, de laatste rij de lokale macrovariabele.

TABEL 3-7. Argumenten-en-lokale -variabelentabel

Naam	Waarde
REG	R0
X	A
I	3
LEN	2
INT	-1
IDX	6

(a) Einde 1ste expansie

Naam	Waarde
REG	R3
X	B
I	7
LEN	10
INT	100
IDX	70

(b) Einde 2de expansie

Lokale macro-etiketten en globale voorvertaleretiketten

Voorvertaleretiketten (afgekort VV-etiketten) worden wel expliciet gedefinieerd, dus hier is geen verwarring mogelijk over het lokaal of globaal zijn van een VV-etiket: een VV-etiket dat gedefinieerd wordt binnen een macrodefinitie is steeds lokaal (en wordt dan ook wel een macro-etiket genoemd); wordt het VV-etiket gedefinieerd buiten een macrodefinitie dan is het globaal.

Het gebruik van deze etiketten in het **`MVSP`**- of het **`MSPR`**-directief is echter onderworpen aan de volgende ***beperking***: het is ***verboden*** 'in of uit een macrodefinitie te springen'. Met andere woorden, indien **`MVSP`** of **`MSPR`** gebruikt wordt binnen een macrodefinitie, dan moet het etiket (**`$ETIKET`**) ook binnen deze macro gedefinieerd zijn; indien deze directieven echter buiten een macrodefinitie gebruikt worden, dan moet het etiket ook buiten een macrodefinitie gedefinieerd zijn.

Macro-etiketten zijn nodig indien we voorwaardelijke macro-opbouw willen voorzien of indien we recursieve macro-oproepen (die eindig zijn) willen toelaten. We verwijzen hiervoor naar hoofdstuk 1.

Globale VV-etiketten kunnen nuttig zijn om bepaalde stukjes code over te slaan (zonder ze echt weg te laten of zonder er het commentaarteken voor te plaatsen). Bijvoorbeeld, bij het ontwikkelen van een programma zal je vaak stukjes code schrijven die bedoeld zijn om het programma uit te testen. Eenmaal de procedures en/of functies correct bevonden zijn, zijn deze stukjes test-code niet meer nodig. Toch is het beter deze niet echt weg te laten, omdat je ze misschien later terug nodig zult hebben wanneer je programma moet aangepast worden. In figuur 3-18 worden globale VV-etiketten gebruikt om bepaalde stukjes code over te slaan (en dus niet te vertalen).

```
| Hoofdprogramma
               ...
| Procedure XXX
               ...
               MSPR     $TST_XXX
| Testprogramma voor Procedure XXX
               ...
$TST_XXX:      MNTS
| Procedure YYY
               ...
               MSPR     $TST_YYY
| Testprogramma voor Procedure YYY
               ...
$TST_YYY:      MNTS
               ...
```

FIGUUR 3-18. Globale voorvertaleretiketten.

Tijdens de voorvertaling van de uiteindelijke versie zullen de testprogramma's voor de procedures '`XXX`' en '`YYY`' weggelaten worden (zie donker gekleurde deel van figuur 3-18). Ze zullen dus niet door de vertaler vertaald worden en bijgevolg niet voorkomen in het uitvoerbaar programma.

Merk op dat globale VV-etiketten slechts ***eenmaal*** mogen gedefinieerd worden; lokale slechts ***eenmaal per macrodefinitie***. Globale VV-etiketten zijn niet zichtbaar binnen een macrodefinitie! Lokale macro-etiketten zijn niet zichtbaar buiten de macrodefinitie waarbinnen ze gedefinieerd zijn. Het is immers niet toegestaan om in of uit een macrodefinitie te springen. Zie ook figuur 3-19. De twee donker gekleurde lijnen illustreren niet toegelaten sprongen: de eerste springt in een macrodefinitie; de tweede springt uit een macro.

	Code			Toelichting
(L1)	MSPR	$NAAM		ga verder bij globaal VV-etiket **$NAAM**
(L2)	MSPR	$LUS		**niet toegestaan!** (**$LUS** ≠ globaal VV-etiket)
(L10)	MACRO			
(L11)		XYZ	A,B,C	
(L15)	$NAAM:	...		lokaal macro-etiket **$NAAM**
(L22)	$LUS:	...		lokaal macro-etiket **$LUS**
(L30)		MSPR	$NAAM	ga verder bij lokaal macro-etiket **$NAAM**
(L31)		MSPR	$EINDE	**niet toegestaan!** (**$EINDE** ≠ lokaal macro-etiket)
(L32)	MCREINDE			
(L40)	$NAAM:	...		globaal VV-etiket **$NAAM**
(L45)	$EINDE:	MNTS		globaal VV-etiket **$EINDE**

FIGUUR 3-19. Globale en lokale voorvertaler etiketten.

Tot slot nog de volgende bemerkingen:

- Indien er tijdens de voorvertaling gesprongen wordt ***over*** een macrodefinitie (zoals dat in het vorige voorbeeld gebeurt via '**MSPR $NAAM**'), dan wordt de definitie **niet** gelezen door de voorvertaler, en mag bijgevolg de macro (i.c. '**XYZ**') **niet** opgeroepen worden. Indien dit toch gebeurt, zal de voorvertaler hiervoor geen fout geven (aangezien hij de macro niet kent, zal hij de oproep letterlijk kopiëren naar het uitvoerbestand) maar de vertaler zal hiermee geen raad weten:

  ***** fout *** XYZ: onbekende functiecode**.

- Indien je m.b.v. '**MSPR**' of '**MVSP**' een lus maakt, moet je ervoor zorgen dat zich binnen die lus **geen** macrodefinities bevinden; anders zal de voorvertaler melden dat een macro meerdere keren gedefinieerd is.
- We hebben gezien dat **$ETIKET** ook gebruikt wordt om unieke DRAMA-etiketten te genereren. Om geen onnodige unieke etiketten te genereren, plaats je VV-etiketten best voor het **MNTS**-directief. Zie ook oefening 14 (op pag. 44).

Gegevensstructuren voor het bijhouden van voorvertaleretiketten

Globale VV-etiketten worden in een **globale-etikettentabel** (GET) bijgehouden. De tabel bevat de namen van deze etiketten samen met de nummers van de lijnen waarop ze gedefinieerd werden. Voor het programmafragment van hierboven zal deze tabel (op het einde van de verwerking) er als volgt uitzien:

TABEL 3-8. Globale Etiketten Tabel.

Etiket	Lijnnummer
`$NAAM`	40
`$EINDE`	45

Lokale macro-etiketten worden in een gelijkaardige tabel bijgehouden, de lokale-macro-etikettentabel (LET). Voor de tabel zelf wordt plaats voorzien in de gegevensstructuur van de macro waarbij ze hoort. De lijnnummers verwijzen dan naar de lijnen in het lichaam van de macro. Bijvoorbeeld, nadat de macrodefinitie[1] is ingelezen door de voorvertaler, zal de gegevensstructuur van de macro '**`XYZ`**' er als volgt uitzien (de lijnnummers zijn relatief t.o.v. het begin van de macro):

TABEL 3-9. De gegevensstructuur van een gedefinieerde macro.

<table>
<tr><td rowspan="5"></td><td rowspan="5">XYZ</td><td rowspan="5">A
B
C</td><td colspan="2">lengte: 19</td></tr>
<tr><td colspan="2">Lokale-etikettentabel</td></tr>
<tr><td>Etiket</td><td>Lijn</td></tr>
<tr><td>$NAAM</td><td>4</td></tr>
<tr><td>$LUS</td><td>11</td></tr>
<tr><td>1:</td><td colspan="4"></td></tr>
<tr><td>4:</td><td colspan="4">$NAAM: ...</td></tr>
<tr><td>11:</td><td colspan="4">$LUS: ...</td></tr>
<tr><td>19:</td><td colspan="4">MSPR $NAAM</td></tr>
</table>

1. We veronderstellen dat de voorvertaler niet over de definitie van **`XYZ`** gesprongen is; dus in figuur 3-19 (pag. 37) zijn de eerste twee lijnen weggelaten alsook de laatste foutieve lijn van de macro.

Samenvatting

Tabel 3-10 geeft een volledig overzicht van de gegevensstructuren die door de voorvertaler bijgehouden worden.

TABEL 3-10. De gegevensstructuren van de macrovoorvertaler.

Afkorting	Inhoud	Aantal
GVT	globale voorvertaler variabelen	1
ALVT	macroparameters en lokale macrovariabelen	1 per oproep
GET	globale voorvertaler etiketten	1
MACRO-DEF	macrodefinitie (hierin wordt een gedefinieerde macro opgeslagen)	1 per definitie
LET	lokale macro-etiketten (is een onderdeel van MACRO-DEF)	—
MCC[a]	**m**acro**c**onditie**c**ode, wordt bewaard in GVT	—
teller	telt de macro-oproepen, wordt bewaard in GVT	—

a. De macroconditiecode is een variabele van de voorvertaler die bij de uitvoering van een **MEVA**- of **MVGL**-directief een nieuwe waarde krijgt.

Aanpassingen aan de macrovoorvertaler

We zullen in het kort bespreken in welk opzicht de voorvertaler aangepast moet worden om alle directieven correct te kunnen uitvoeren.

De voorvertaler kan zich nog steeds in drie mogelijke toestanden bevinden: ***kopieermodus***, ***definitiemodus*** en ***expansiemodus***.

1. In de **definitiemodus** verandert er weinig: tijdens het kopiëren van het lichaam wordt de lokale-etikettentabel (LET) opgesteld.
2. In de **kopieermodus** zijn er drie belangrijke wijzigingen:
   - **a.** Indien in de invoer een globaal macro-etiket gedefinieerd wordt, dan wordt de naam en het lijnnummer toegevoegd aan de GET.
   - **b.** In de lijnen die naar het uitvoerbestand gekopieerd moeten worden, moeten alle symbolen van de vorm **<NAAM>** vervangen worden door de waarde die in de GVT gevonden wordt.
   - **c.** Tenslotte moet de voorvertaler ook de extra macrodirectieven kunnen verwerken:
     - **MEVA**: de expressie wordt geëvalueerd en samen met de naam van de variabele toegevoegd aan de GVT. Indien de variabele reeds gedefinieerd is, wordt alleen de waarde aangepast. De macroconditiecode (MCC) krijgt een nieuwe waarde.
     - **MNTS**: hiermee moet niets gebeuren.
     - **MFOUT**: er wordt een foutboodschap afgedrukt en de macrovoorvertaler stopt.

- **`MSPR`**: de voorvertaler zoekt in de GET het lijnnummer van het etiket; het kopiëren gaat verder bij de lijn met dit lijnnummer. Indien het nog niet opgenomen is in de tabel, blijft de voorvertaler lijnen overslaan tot het etiket gevonden is. (Merk op dat de voorvertaler geen twee stappen nodig heeft zoals de vertaler! Wel is het mogelijk dat door die sprongen de invoer meer dan een keer gelezen zal worden.)
- **`MVGL`**: de twee expressies worden geëvalueerd en vergeleken. De macroconditiecode (MCC) krijgt een nieuwe waarde.
- **`MVSP`**: analoog aan **`MSPR`**, maar de sprong wordt maar uitgevoerd indien de voorwaarde overeenkomt met de waarde in de MCC.

3. Ook de **expansiemodus** moet substantieel uitgebreid worden.

a. Bij de substitutie van **`<NAAM>`**-symbolen, wordt de waarde eerst in de ALVT gezocht, en indien het symbool daar niet voorkomt moet er ook in de GVT gezocht worden. Deze volgorde van behandeling heeft als gevolg dat een globale VV-veranderlijke met dezelfde naam als een parameter “onzichtbaar” wordt gedurende de expansie van de macro-oproep.

b. Tenslotte moeten alle extra macrodirectieven herkend en verwerkt worden:

- **`MEVA`**: de expressie wordt geëvalueerd. Indien de variabele reeds voorkomt in de ALVT, dan wordt de waarde daar aangepast; zoniet, indien de variabele voorkomt in de GVT, dan wordt de waarde daar aangepast; zoniet wordt de variabele en de waarde toegevoegd aan de ALVT. De macroconditiecode variabele (MCC) krijgt een nieuwe waarde.
- **`MNTS`**: hiermee moet niets gebeuren.
- **`MFOUT`**: er wordt een foutboodschap afgedrukt en de macrovoorvertaler stopt.
- **`MSPR`**: de voorvertaler zoekt in de LET het lijnnummer van het etiket; de expansie gaat verder bij de lijn met dit lijnnummer. (De LET is opgesteld tijdens het inlezen van de macrodefinitie; dus moet het etiket in de LET voorkomen, zoniet wordt een foutboodschap afgedrukt en stopt de voorvertaler).
- **`MVGL`**: de twee expressies worden geëvalueerd en vergeleken. De macroconditiecode (MCC) krijgt een nieuwe waarde.
- **`MVSP`**: analoog aan **`MSPR`**, maar de sprong wordt slechts uitgevoerd indien de voorwaarde overeenkomt met de waarde in de MCC.

Opgaven

1. Weeg de voor- en nadelen af van een aparte macrovoorvertaler.
2. Wat wordt bedoeld met “De werking van de voorvertaler komt grosso modo neer op symboolmanipulatie”?
3. In welke toestanden kan de voorvertaler zich bevinden? Teken het toestandsdiagram.
4. Beschrijf bondig de werking van de DRAMA-macrovoorvertaler indien alleen **MACRO-** en **MCREINDE**-directieven gebruikt worden.
5. Wat is een argumententabel? Waartoe en wanneer wordt ze opgesteld? Welke informatie bevat ze?
6. Wat is het onderscheid tussen globale VV-variabelen en lokale macrovariabelen? Welke toepassingen kan je bedenken voor globale VV-variabelen? Waar wordt informatie over deze variabelen bewaard? Is het mogelijk dat een globale VV-variabele onzichtbaar is binnen een macro? Zo ja, geef een voorbeeld.
7. Wat is het onderscheid tussen globale VV-etiketten en lokale macro-etiketten? Welke toepassing(en) kan je bedenken voor globale VV-etiketten? Zijn er beperkingen bij het gebruik van globale en lokale VV-etiketten? Waar wordt informatie over deze etiketten bewaard?
8. Welke aanpassingen zijn er nodig aan de voorvertaler, indien ook de andere directieven zoals **MEVA**, **MVSP**, enz. toegelaten worden?
9. Wat is het resultaat van de voorvertaling van het volgende DRAMA-programma:

```
          MEVA     <REG>,0
$LUS:     MNTS
          HIA.w    R<REG>,0
          MEVA     REG,<REG>+1
          MVGL     <REG>,10
          MVSP     KL,$LUS
          STP
EINDPR
```

10. Wat is het resultaat van de voorvertaling van het volgende drama-programma:

```
MEVA       <MAX>,15     | 15! is het maximum
                        | dat we kunnen berekenen
                        | op de DRAMA-machine
|  Macro FAC berekent N!
MACRO
              ETIKET:  FAC     N,REG
                       MVGL    <N>,0
                       MVSP    GRG,$POS
                       MFOUT   'FAC: N moet >= 0'
                $POS:  MVGL    <N>,<MAX>
```

```
                     MVSP    KLG,$OK
                     MFOUT   'FAC: N moet <= <MAX>'
                $OK: MNTS
           <ETIKET>: HIA.w   R<REG>,1
                     RFAC    2,<N>,<REG>
MCREINDE
|
| Macro RFAC is recursief en zorgt
| voor de nodig VER-instructies
|
MACRO
          RFAC       START,EINDE,REG
          MVGL       <START>,<EINDE>
          MVSP       GR,$KLAAR
          VER.w      R<REG>,<START>
          MEVA       <START>,<START>+1
          RFAC       <START>,<EINDE>,<REG>
$KLAAR:   MCREINDE
|
| Hoofdprogramma
|
          MEVA       RG,0
  HOOFD:  FAC        5,<RG>
          DRU
          NWL
          MEVA       RG,1
   DRIE:  FAC        3,<RG>
          HIA        R0,R<RG>
          DRU
          NWL
          STP
EINDPR
```

11. Schrijf een macro (**PLAATS**) die een aantal opeenvolgende geheugenregisters reserveert, en deze een unieke naam geeft. Als enige parameter wordt het aantal te reserveren plaatsen opgegeven. De macro genereert ook code die ervoor zorgt dat er over die plaats gesprongen wordt. Aangezien de DRAMA-machine geen **NTS** instructie kent (die niets doet), gebruik je hiervoor **ETIKET: SPR ETIKET+1**. De unieke namen zijn van de vorm **_V*getal*** waarbij ***getal*** bij elke reservatie opgehoogd wordt. Je voorziet ook een globale VV-variabele (**<V>**) waarin dit ***getal*** bijgehouden wordt, zodat in het programma naar de laatst gegene-

reerde naam kan verwezen worden. Probeer geen andere VV-etiketten te gebruiken, maar genereer zelf de nodige etiketten m.b.v. globale VV-variabelen. Het volgende fragment illustreert het gebruik van deze macro:

```
...
PLAATS   20
HIA      R1,_V<V>+2
PLAATS   30
OPT      R1,_V<V>+4
...
```

Na de voorvertaling ziet het programma er als volgt uit:

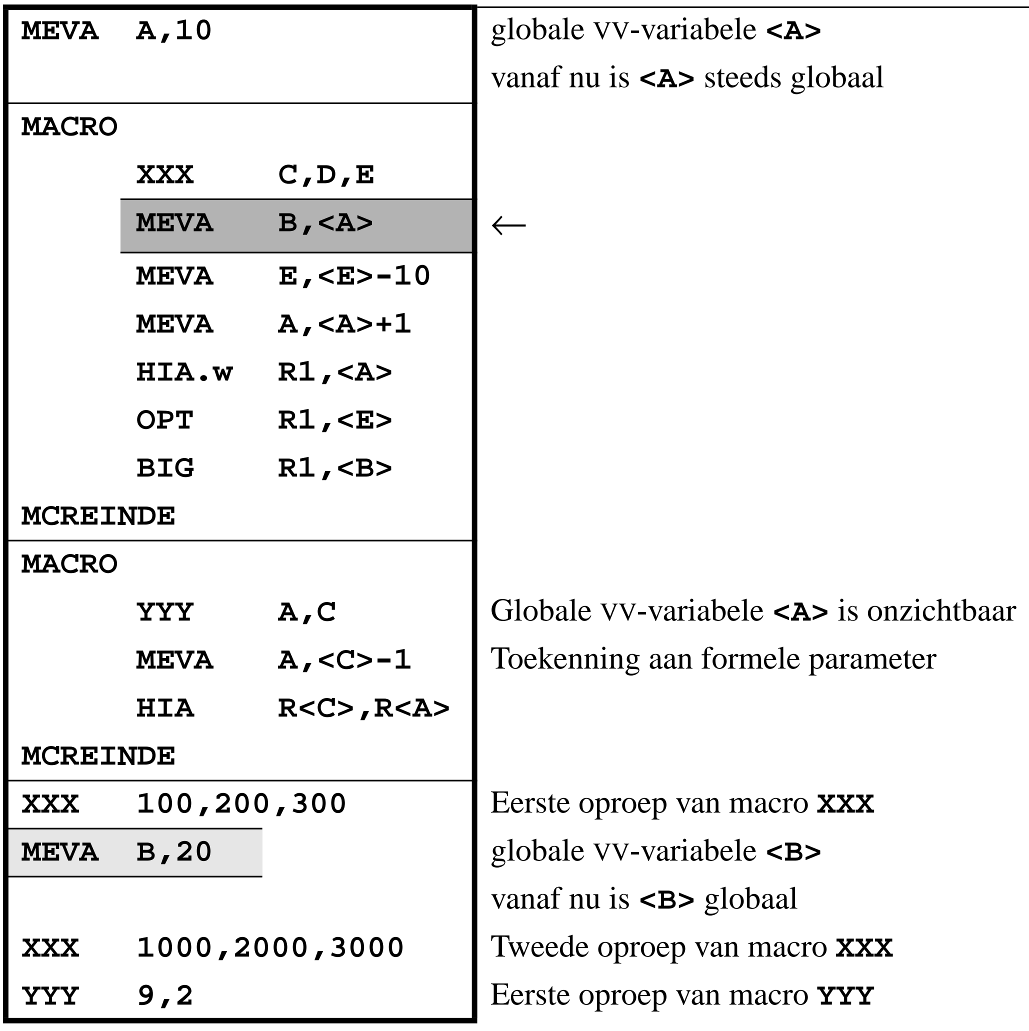
```
        ...
        SPR       _E0
_V0:    RESGR     20
_E0:    SPR       _E0+1
        HIA       R1,_V0+2
        SPR       _E1
_V1:    RESGR     30
_E1:    SPR       _E1+1
        OPT       R1,_V1+2
        ...
```

12. Breid de **if-then-endif**-constructie uit figuur 3-15 (pag. 32) uit met een **else**-gedeelte. Maak je macrodefinities ook robuuster. Zo moet je in de **THEN**-macro testen of een **IF**-oproep gebeurd is (gebruik hiervoor een globale variabele **TOESTAND**). In de **ELSE**-macro test je of er een voorafgaandelijke **THEN**-oproep gebeurd is, en tenslotte test je in de **ENDIF** of er een **THEN**- of een **ELSE**-oproep gebeurd is. Bovendien moet je ook nagaan of er zich geen ineennesteling van the **if-then-else**-constructies voordoet. Of beter nog, pas je oplossing aan zodat ineennesteling geen probleem vormt. Is dit laatste gemakkelijk te realiseren?

13. Het is mogelijk dat tijdens een macro-expansie een VV-variabele lokaal is, terwijl bij een latere expansie dezelfde variabele globaal is, omdat er tussen de twee oproepen een 'globale toekenning' gebeurd is.

Figuur 3-20 (pag. 44) illustreert dit. Tijdens de expansie van de **eerste** macro-oproep van '**XXX**' bestaat er nog geen globale VV-variabele **B**, dus is de VV-variabele lokaal t.o.v. de macro (zie donker gekleurde lijn in figuur 3-20). Na de expansie bestaat **B** niet meer. De **tweede** oproep van '**XXX**' gebeurt na de '**MEVA**', waarbij de VV-variabele **B** voortaan globaal wordt. Dus na de expansie blijft **B** bestaan.
Gedurende de expansie van de macro-oproep van '**YYY**' komt **A** voor als formele parameter. Hierdoor verliest **A** tijdelijk zijn status als globale VV-variabele. Men zegt dat de globale VV-variabele **A** onzichtbaar is gedurende de expansie van de macro-oproep **YYY**.

Wat is het resultaat van de voorvertaling? Wat is de inhoud van de GVT na de uitvoering van '**MEVA B,20**'? Wat is de inhoud van de ALVT op het einde van de eerste oproep van **XXX** en op het einde van de tweede oproep van **XXX**?



Code			Commentaar
MEVA	A,10		globale VV-variabele **<A>**
			vanaf nu is **<A>** steeds globaal
MACRO			
	XXX	C,D,E	
	MEVA	B,<A>	←
	MEVA	E,<E>-10	
	MEVA	A,<A>+1	
	HIA.w	R1,<A>	
	OPT	R1,<E>	
	BIG	R1,<B>	
MCREINDE			
MACRO			
	YYY	A,C	Globale VV-variabele **<A>** is onzichtbaar
	MEVA	A,<C>-1	Toekenning aan formele parameter
	HIA	R<C>,R<A>	
MCREINDE			
XXX	100,200,300		Eerste oproep van macro **XXX**
MEVA	B,20		globale VV-variabele **<B>**
			vanaf nu is **<B>** globaal
XXX	1000,2000,3000		Tweede oproep van macro **XXX**
YYY	9,2		Eerste oproep van macro **YYY**

FIGUUR 3-20. Globale VV-variabelen en lokale macrovariabelen.

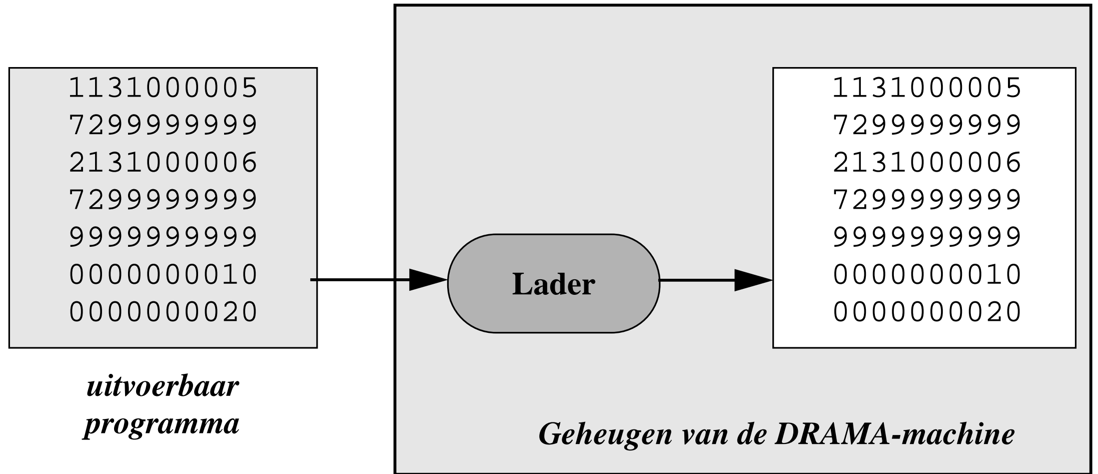
14. Wat is het resultaat van de voorvertaling van het volgende programma? Waarom zal de vertaler later een fout geven? Pas de macrodefinitie aan zodat deze fout in de toekomst niet meer voorkomt.

```
MACRO
        INCR   A
        | Tel <A> keer 1 bij R0 bij
$LUS:   OPT.w  R0,1
        MVGL   <A>,0
        MVSP   GR,$LUS
MCREINDE
        HIA.w  R0,0
        INCR   10
```

```
        DRU
        STP
EINDPR
```

3.3 De lader

Als het bronprogramma vertaald is naar machinecode, moet deze machinecode nog in het geheugen gebracht worden alvorens het uitgevoerd kan worden. In principe zou de vertaler de uitvoerbare code rechtstreeks in het geheugen kunnen plaatsen. Dit heeft echter als nadeel dat we vóór elke uitvoering het programma opnieuw moeten vertalen. We besparen heel wat tijd indien we het uitvoerbaar programma op een hulpgeheugen bewaren. Telkens als we het willen uitvoeren, moet het uitvoerbaar programma alleen maar in het geheugen gebracht worden. Hiervoor zal een speciaal programma instaan, de ***lader***. Zie figuur 3-21. Merk op dat de lader zelf ook in het geheugen zit!



FIGUUR 3-21. De lader

Bij een modern computersysteem maakt de lader deel uit van het besturingsprogramma, zodat we het ook in het volgende hoofdstuk hadden kunnen bespreken. Maar om het besturingsprogramma in het geheugen te kunnen laden heb je echter ook een lader nodig! We komen hierop terug in paragraaf *3.7 "Het opstarten van de computer" op pag. 92*.

Er bestaan verschillende ***laderalgoritmes*** of ***laderschema's***. We onderscheiden twee hoofdklassen:

1. de ***absolute*** lader, die het vertaalde programma (het uitvoerbaar programma) ***ongewijzigd*** in het geheugen plaatst, en
2. de ***relocerende*** lader, die in staat is het uitvoerbaar programma aan te passen aan de plaats die het in het geheugen zal innemen. Met name de adressen die in de machinebevelen gebruikt worden zullen soms moeten gewijzigd worden.

Ten slotte vermelden we ook de trend om apparatuur meer complexe taken te laten uitvoeren, zoals dynamische adresvertalingen. Hierdoor kan de programmatuur (i.c. de lader) heel wat vereenvoudigd worden.

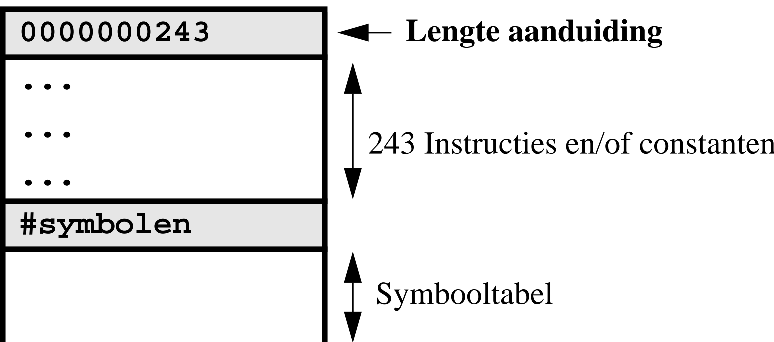
3.3.1 De absolute lader

De absolute lader brengt het uitvoerbaar programma (dat uitsluitend bestaat uit een opeenvolging van getallen[1]) ***ongewijzigd*** in het geheugen. Het laderprogramma is zo eenvoudig dat we hiervoor gemakkelijk de DRAMA-code kunnen geven.

Het meest elementaire geval

Vooraf maken we een aantal ***veronderstellingen***, waarvan we er later enkele zullen afzwakken:

- Het programma moet geladen worden vanaf geheugenregister met adres `0000`.
- Het te laden programma bevindt zich in het invoerorgaan (en kan dus m.b.v. **`LEZ`**-instructies ingelezen worden). Het invoerorgaan zou bijvoorbeeld een kaartlezer kunnen zijn, en de machinecode geponst in ponskaarten.
- Het eerste getal van het te laden programma is de ***lengte*** van het programma. De vertaler zal dit getal genereren vóór de eerste machine-instructie; hij weet immers reeds op het einde van de eerste stap wat de exacte lengte van het programma is, namelijk de waarde die de programmateller op dat ogenblik heeft. Bijvoorbeeld voor een programma van lengte **`243`**, zal de uitvoerbare code er als volgt uitzien:



- Het laderprogramma zelf is reeds geladen vanaf adres **`9000`**. Hoe het daar is terechtgekomen, bespreken we in paragraaf *3.7* .

De absolute lader zal de volgende **acties** ondernemen:

1. de lengte inlezen, (die heeft de lader nodig om te weten hoeveel getallen hij moet inlezen),

2. de opeenvolgende getallen van het programma inlezen en wegbergen in opeenvolgende geheugenplaatsen te beginnen bij adres `0000`,

3. tenslotte de uitvoering van het ingelezen programma starten, d.w.z. naar adres `0000` springen.

Figuur 3-22 geeft de DRAMA-code voor de lader. Indien de programmeur elk programma beëindigt met een **`SPR 9000`** i.p.v. een **`STP`**-bevel[2], zal de lader het volgende programma dat in het

1. De eventueel aanwezige symbooltabel wordt door de lader genegeerd.
2. De vertaler zou elk **`STP`**-bevel automatisch kunnen omzetten naar een **`SPR 9000`**.

invoer-orgaan is geplaatst, inladen en de uitvoering ervan starten[1], enz.
We verkrijgen aldus een heel eenvoudig besturingssysteem voor onze DRAMA-machine, dat voordurend nieuwe programma's zal inlezen en uitvoeren!.

```
| Programma voor de absolute DRAMA-lader
      LEZ                    | Lees lengte van programma in
      HIA     R2,R0          | R2 --> lengte van het programma
      HIA.w   R1,0           | R1 --> adres van geheugenregister
LUS:  VGL.w   R2,0
      VSP     KLG,0          | Klaar! --> Start uitvoering
      LEZ                    | Lees de opeenv. instructies in
      BIG     R0,0(R1+)      | Plaats ze in opeenv. geheugenreg.
      AFT.w   R2,1
      SPR     LUS
EINDPR
```

FIGUUR 3-22. De DRAMA-code voor de absolute lader.

Wil je dit programma ook werkelijk uitproberen op de DRAMA-simulator, dan zal je enkele ***'truukjes'*** moeten toepassen.

- Indien je deze lader vertaalt en uitvoert op de simulator, zal het fout lopen. **Waarom?** Het laderprogramma wordt zelf vanaf adres `0000` geladen en zal tijdens de uitvoering zichzelf overschrijven! Je moet er dus voor zorgen dat de lader ergens anders in het geheugen geplaatst wordt. Verander daartoe de eerste **`LEZ`**-opdracht door de volgende drie opdrachten, en ga zelf na dat de lader zichzelf nu niet meer zal overschrijven[2] (voor zover de lengte van het te laden programma niet groter is dan **`9000`**).

```
          SPR     LADER
          RESGR   8999
LADER:    LEZ
          ...
```

- De beschikbare DRAMA-vertaler genereert geen lengteaanduiding in het uitvoerbaar bestand. Je zal dit zelf vooraan moeten toevoegen. De ingevulde lengte moet correct zijn!
- Aan de simulator moet je meedelen dat de invoer voor jouw lader uit een bestand komt (en niet via het toetsenbord wordt ingegeven). Dit kan je met de '**-invoer bestandsnaam**' optie realiseren[3].

1. Dit werkt slechts indien de symbooltabel verwijderd is uit elk uitvoerbaar programma.
2. De eerste **`SPR`**-opdracht wordt wel overschreven maar dit is niet erg. Waarom niet?
3. Opgelet! Indien het te laden programma zelf **`LEZ-`**instructies bevat, zal je de symbooltabel uit het uitvoerbaar bestand moeten weglaten en vervangen door de in te lezen gegevens.

De interne lader van de DRAMA-simulator zal dan jouw (absolute) lader in het geheugen laden en uitvoeren. Tijdens de uitvoering van jouw lader wordt een ander programma geladen en tenslotte uitgevoerd.

Verfijningen

We laten eerst de veronderstelling vallen dat een DRAMA-programma ***altijd*** vanaf geheugenregister met adres ***0000*** geladen zal worden. De vertaler heeft van deze veronderstelling gebruik gemaakt om de symbooltabel op te stellen. Hij telt immers de DRAMA-instructies vanaf **0**. Indien de computer slechts één programma in zijn geheugen wil houden, vormt deze veronderstelling geen echte beperking[1]. Wanneer we echter meerdere programma's tegelijkertijd in het geheugen zouden willen houden (zoals bijvoorbeeld bij timesharing[2] systemen het geval is), dan kunnen die natuurlijk niet allemaal vanaf ***0000*** geladen worden.

Figuur 3-23 (pag. 50) schetst zo'n situatie waarbij driemaal hetzelfde programma in het geheugen geladen is, zij het op verschillende plaatsen: vanaf adres ***0000***, adres ***0100*** en adres ***0300***. Merk op dat de machinecode voor de drie programma's verschilt. Dit is normaal daar de variabelen '**A**' en '**B**' in de drie gevallen een ander adres hebben. Het verschil '**B-A**' is echter een constante. '**B-A**' is een 'afstand': namelijk het aantal geheugenregisters dat tussen **A** en **B** gelegen is plus één. In de figuur werden de verschillen onderlijnd, en donkerder afgebeeld.

Aangezien de absolute lader zelf niet in staat is om het vertaalde programma nog te wijzigen, moet de machinecode die door het assembleerprogramma geleverd wordt reeds de correcte adressen bevatten. Bijgevolg moet het assembleerprogramma door de ***programmeur*** op de hoogte gebracht worden van het feit dat het te vertalen programma **niet** vanaf geheugenregister ***0000*** zal geladen worden maar bijvoorbeeld vanaf geheugenregister ***0300***.
Dit kan gebeuren via een speciaal ***vertalerdirectief,*** **LAADGR** (Laad vanaf geheugenregister), dat als operand het beginadres van het programma bevat (i.c. ***0300***). Dit **LAADGR**-directief moet door de programmeur worden vermeld in zijn bronprogramma **vóór** de eerste DRAMA-instructie. Bij de **vertaling** zullen alle symbolische adressen met dat beginadres verhoogd worden; m.a.w. de vertaler zal de instructies beginnen tellen vanaf dit beginadres.
Het spreekt vanzelf dat het vertaalde programma alleen maar vanaf adres ***0300*** geladen mag worden. Zoniet, zullen de adressen in sommige bevelen niet kloppen. Dus deze informatie zal ook door het vertaalprogramma aan de lader doorgegeven worden, via een laadadresaanduiding (zie het tweede getal in de hoofding van het uitvoerbaar programma van figuur 3-24 (pag. 51)).

1. Merk op dat de lader ook permanent aanwezig is in het geheugen. Aangezien dit een heel speciaal programma is, tellen we dit niet mee.
2. Op een timesharing systeem kunnen verschillende gebruikers tegelijkertijd met de computer werken. Voor elke gebruiker worden een of meerdere programma's uitgevoerd. Zie hoofdstuk 4.

Symbolisch			Adres	Decimaal
	HIA	R1,A	0000:	1131100004
	OPT.w	R1,B-A	0001:	2111100001
	BIG	R1,B	0002:	1221100005
	STP		0003:	9999999999
A:	10		0004:	0000000010
B:	RESGR	1	0005:	0000000000
				...
	HIA	R1,A	0100:	1131100104
	OPT.w	R1,B-A	0101:	2111100001
	BIG	R1,B	0102:	1221100105
	STP		0103:	9999999999
A:	10		0104:	0000000010
B:	RESGR	1	0105:	0000000000
				...
	HIA	R1,A	0300:	1131100304
	OPT.w	R1,B-A	0301:	2111100001
	BIG	R1,B	0302:	1221100305
	STP		0303:	9999999999
A:	10		0304:	0000000010
B:	RESGR	1	0305:	0000000000
				...

Geheugen

FIGUUR 3-23. Drie programma's in het geheugen.

Zij gegeven het volgende DRAMA-bronprogramma en het overeenkomstig uitvoerbaar programma (zie figuur 3-24).

(a) DRAMA-bronprogramma

```
    LAADGR  300
    HIA     R1,A
    OPT.w   R1,B-A
    BIG     R1,B
    STP
A:  10
B:  RESGR   1
EINDPR
```

(b) Uitvoerbaar programma

lengte →	0000000006
laadadres →	0000000300
0300:	1131100304
0301:	2111100001
0302:	1221100305
0303:	9999999999
0304:	0000000010
0305:	0000000000
	#symbolen
	A 0304
	B 0305

FIGUUR 3-24. Een DRAMA-programma, dat geladen moet worden vanaf geheugenregister 0300.

In het rechtergedeelte van de figuur zie je het uitvoerbaar programma[1] dat door de lader in het geheugen gebracht moet worden. Het spreekt vanzelf dat een dergelijk programma alleen maar vanaf adres *`0300`* geladen mag worden. Je kan duidelijk zien dat het uitvoerbaar programma uit drie delen bestaat:

- een ***hoofding***, waarin de lengte en het laadadres vermeld worden; deze informatie heeft de lader nodig om het programma op de juiste plaats en op een correcte wijze (d.i. volledig) in het geheugen te brengen;
- de ***programmacode*** (de machinecode en data), die door de lader in het geheugen geladen zal worden;
- de ***symbooltabel***, die bedoeld is voor andere programma's, bijvoorbeeld het speurprogramma (zie paragraaf 3.6 (op pag. 86)) of de binder (zie paragraaf 3.4 (op pag. 64)).

Laten we even terugkeren naar figuur 3-23 (pag. 50). Om dit te realiseren hebben we dus driemaal hetzelfde programma laten vertalen door het assembleerprogramma, de eerste keer met het directief[2] '`LAADGR 0`', de tweede maal met '`LAADGR 100`', en een laatste keer met '`LAADGR 300`' in het bronprogramma. Het resultaat van die drie vertalingen is te zien in figuur 3-25 (pag. 52). Aan de lader werd dan gevraagd om deze drie uitvoerbare programma's in het geheugen te brengen.

1. Vóór elke instructie/constante is het adres weergegeven van het geheugenregister waarin dit decimaal getal geplaatst moet worden; uiteraard komen deze adressen niet voor in het uitvoerbaar programma.
2. Indien het bronprogramma geen `LAADGR`-directief bevat, wordt door de vertaler `LAADGR 0` verondersteld.

Pas zelf de DRAMA-code van de absolute lader (fig. 3-22 (pag. 48)) aan zodat hij in staat is programma's op een verschillende plaats in het geheugen in te laden.

0000000006	0000000006	0000000006
0000000000	0000000100	0000000300
1131100004	1131100104	1131100304
2111100001	2111100001	2111100001
1221100005	1221100105	1221100305
9999999999	9999999999	9999999999
0000000010	0000000010	0000000010
0000000000	0000000000	0000000000
#symbolen	#symbolen	#symbolen
A 0004	A 0104	A 0304
B 0005	B 0105	B 0305

FIGUUR 3-25. De drie uitvoerbare programma's, respectievelijk te laden vanaf 0000, 0100 en 0300.

3.3.2 De relocerende lader

In plaats van het bronprogramma te vertalen in functie van de plaats waar het in het geheugen zal geladen worden, zou de vertaler aan de lader kunnen meedelen ***hoe de machinecode moet aangepast worden*** (d.i. hoe sommige adressen moeten aangepast worden) ***indien het programma niet vanaf geheugenregister met adres 0000 geladen wordt***.

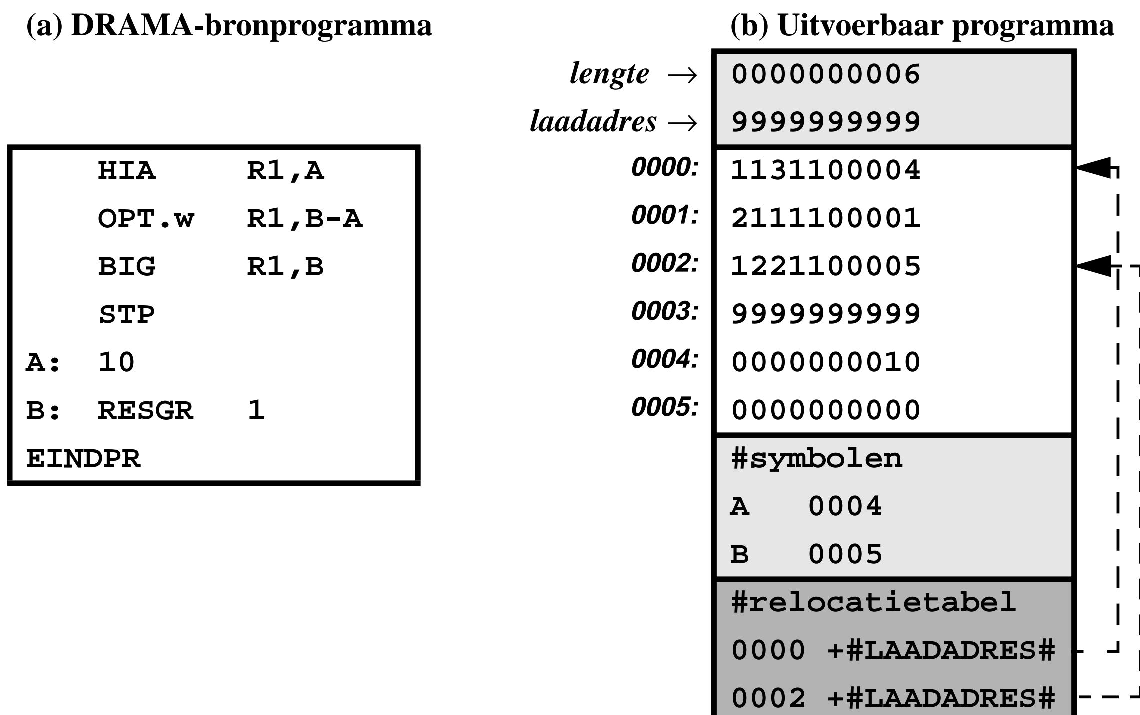
Een lader die in staat is de machinecode aan te passen noemt men een ***relocerende lader***. Natuurlijk moet in de eerste plaats de vertaler ***reloceerbare code*** genereren. Dit betekent dat het uitvoerbaar programma extra informatie zal bevatten. Hernemen we het voorbeeld van hierboven. De vertaler gaat ervan uit dat het programma geladen wordt vanaf adres ***0000***. De machinecode die in het uitvoerbaar programma staat zal dus correct zijn indien het programma ook werkelijk vanaf ***0000*** geladen zal worden.

Zou de lader dit programma op een ander adres (bijvoorbeeld ***0300***) willen laden, dan moeten sommige operanden (adressen) aangepast worden. De vertaler zal hiertoe de nodige instructies geven aan de lader: achteraan in het uitvoerbaar programma zal de vertaler een ***relocatietabel*** toevoegen aan het uitvoerbaar programma. Met deze tabel kan de lader de code op een correcte wijze aanpassen. Zie figuur 3-26 (pag. 53). In het voorbeeld wordt de relocatietabel via de symbolenrij, **#relocatietabel**, aangekondigd. Elke lijn in deze tabel is van de vorm:

relatief adres[1] **bewerking**

1. De instructies/constanten/geheugenreservaties worden vanaf adres 0 geteld. 'Relatief' betekent dus 't.o.v. het begin van het programma'

Om aan te geven dat een uitvoerbaar programma reloceerbaar is (d.w.z. een relocatietabel bevat, zal de vertaler '`-1`' (in 10-complement) als laadadres opgeven in de hoofding.



FIGUUR 3-26. Een reloceerbaar DRAMA**-programma.**

Het relatief adres duidt aan ***welke*** instructie moet aangepast worden; de bewerking geeft aan ***hoe*** het moet aangepast worden. Het is voor de vertaler niet moeilijk om deze relocatie-informatie te genereren (tijdens de tweede stap): telkens een symbolisch adres voorkomt in het operandgedeelte, zal de vertaler **`programmateller +/-#LAADADRES#`** uitschrijven, afhankelijk of het symbolisch adres al dan niet voorafgegaan is door een minteken.

Merk op dat **`B-A`** in de instructie `OPT.w R1,B-A` ***onafhankelijk*** is van de plaats waar het programma in het geheugen wordt geladen. Het is een constante term; dus hoeft ze niet gereloceerd te worden. In feite wordt deze term twee keer gereloceerd: eenmaal moet **`#LAADADRES#`** bijgeteld worden, en eenmaal afgetrokken. Een intelligente vertaler zal beide elkaar opheffende lijnen niet toevoegen aan de relocatietabel.

Toegepast op dit voorbeeld: bij het lezen van de eerste instructie van het DRAMA-programma (**`HIA R1,A`**), zal de vertaler

```
0000 +#LAADADRES#
```

uitschrijven, omdat de programmateller op dat ogenblik de waarde **`0`** heeft. Na de vertaling wordt de programmateller met één opgehoogd. Bij het lezen van de tweede instructie worden twee relocatieopdrachten uitgeschreven (die tegen elkaar wegvallen):

```
0001 +#LAADADRES#
0001 -#LAADADRES#
```

Voor de derde instructie (de programmateller heeft dan de waarde 2), genereert de vertaler:

```
0002 +#LAADADRES#.
```

De relocerende lader zal — net zoals de DRAMA-vertaler — ook in stappen werken:

- in de eerste stap wordt het uitvoerbaar programma letterlijk in het geheugen geladen, vanaf het laadadres,
- in de tweede stap, ***de relocatiestap***, zal de code aangepast worden m.b.v. de relocatietabel.

Stel dat de lader beslist het voorgaande programma vanaf ***0300*** in het geheugen te laden. Figuur 3-27 (a) geeft de inhoud van de geheugenregisters weer na het laden van de code (maar vóór de relocatiestap). In de tweede fase wordt de code aangepast. In het voorbeeld moeten geheugenregisters ***0300+0000=0300*** en ***0300+0002=0302*** gewijzigd worden. In beide gevallen moet er het laadadres (**0300**) bijgeteld worden. Dus:

1131100004 ⇨ 1131100304

1221100005 ⇨ 1221100305

Merk op dat we precies hetzelfde resultaat bekomen indien we het bronprogramma eerst met het **LAADGR 300** directief laten vertalen, en het daarna door de ***absolute lader*** in het geheugen laten inlezen. (Vergelijk de rechterkolom met de rechterkolom van figuur 3-23 (pag. 50).)

(a) vóór relocatie

	...
0300:	1131100004
0301:	2111100001
0302:	1221100005
0303:	9999999999
0304:	0000000010
0305:	0000000000
	...

(b) ná relocatie

	...
0300:	1131100304
0301:	2111100001
0302:	1221100305
0303:	9999999999
0304:	0000000010
0305:	0000000000
	...

FIGUUR 3-27. De relocerende lader

Het grote voordeel van een relocerende lader is dat de programmeur ontlast wordt van het opgeven van een laadadres. De keuze kan uitgesteld worden tot op het ogenblik dat het programma moet uitgevoerd worden, en zal afhankelijk zijn van wat er zich op dat ogenblik reeds in het geheugen bevindt. Op de meeste computers zullen immers verschillende programma's gelijktijdig in het geheugen geladen zijn, omdat op die manier het computersysteem efficiënter kan werken. We komen hierop terug in het vierde hoofdstuk wanneer we de taken van het besturingsprogramma bestuderen.

In paragraaf B.2 (op pag. 209) vind je het programma in pseudo-PASCAL-code voor de relocerende lader.

Gevolgen voor de vertaler

We kunnen slechts een relocerende lader gebruiken indien het uitvoerbaar programma ***relo-ceerbaar*** is, d.w.z. indien het een relocatietabel bevat. Deze tabel wordt door de vertaler tijdens de tweede stap opgesteld (namelijk bij de berekening van de adressen) en op het einde van deze stap toegevoegd aan het uitvoerbaar programma. De wijziging aan de vertaler is eerder gering. Indien je de vertaler echter aangekocht hebt, kan zo'n wijziging onmogelijk zijn, omdat je niet beschikt over de broncode van de vertaler. In de volgende paragraaf over binders zal de vertaler nog meer moeten aangepast worden. Hij wordt steeds complexer, maar de flexibiliteit die je hierdoor wint is groot. Het principe 'niets voor niets' is ook in de informatica van toepassing!

Er bestaat echter nog een andere manier om de flexibiliteit te verhogen: de hardware meer taken op zich laten nemen. Op deze wijze wordt de hardware complexer, maar blijft de programmatuur eenvoudig. Een voorbeeld wordt in de volgende paragraaf gegeven.

3.3.3 Dynamische relocatie

De techniek beschreven in vorige paragraaf wordt ***statische relocatie*** genoemd, omdat de relocatie eenmalig gebeurt vóór de uitvoering van het programma. Bij modernere computersystemen is de apparatuur (hardware) aangepast. Deze systemen beschikken over een **geheugenbeheereenheid** (afgekort GBE) (in het Engels '*memory management unit*', vaak afgekort tot MMU), die statische relocatie overbodig maakt. Relocatie zal nu ***dynamisch***, d.w.z. tijdens de uitvoering van het programma doorgevoerd worden. Men spreekt ook soms over ***dynamische adresvertaling*** (in het Engels '*dynamic address translation*', soms afgekort tot DAT).

In hoofdstuk 1 hebben we ook reeds een vorm van dynamische adresberekening gezien: namelijk bij het gebruik van indexregisters. Het adres (of de waarde) die in de instructie gebruikt zal worden, wordt berekend tijdens de uitvoering van het programma (d.i. bij het formele adres wordt de inhoud van het indexregister bijgeteld). Zo'n berekening hoeft alleen te gebeuren indien een indexregister aanwezig is in de instructie. Hier gaan we nog een stapje verder. De geheugenbeheereenheid zal **alle adressen**[1] herberekenen.

Dynamische relocatie kan op een redelijk eenvoudige wijze gerealiseerd worden m.b.v. een ***basisregister***. Dit register behoort tot het centraal verwerkingsorgaan[2] (afgekort CVO) en wordt geïnitialiseerd door het besturingsprogramma met het beginadres van het geladen programma (juist voor de uitvoering van dit programma begonnen wordt). Telkens het CVO de inhoud van een geheugenregister met een bepaald adres wil ophalen of wijzigen, zal de hardware automatisch de inhoud van het basisregister bijtellen bij dit adres. Het adres wordt dus dynamisch (d.i. tijdens de uitvoeringsfase) gereloceerd. Een voorbeeld zal dit mechanisme illustreren.

1. Alleen adressen die naar het geheugen worden gestuurd (voor een lees- of schrijfoperatie).
2. Met CVO bedoelen we de 'processor' ('*central processing unit*' in het Engels, of afgekort CPU). In het vervolg zullen we CVO, CVE (afkorting van Centrale VerwerkingsEenheid) en processor door elkaar gebruiken.

Voorbeeld 3-4. *Hernemen we het eenvoudige programma. Links staat de broncode, rechts de inhoud van de geheugenregisters. We veronderstellen dat het programma vanaf `0300` geladen werd.*

(a) DRAMA-bronprogramma

```
     HIA     R1,A
     OPT.w   R1,B-A
     BIG     R1,B
     STP
 A:  10
 B:  RESGR   1
EINDPR
```

(b) Geladen in het geheugen

Adres	Inhoud
	...
0300:	1131100004
0301:	2111100001
0302:	1221100005
0303:	9999999999
0304:	0000000010
0305:	0000000000
	...

Het basisregister (**br**) wordt geïnitialiseerd met de waarde *0300*. De bevelenteller (BT) heeft de waarde 0000.

a. Het CVO haalt de eerste instructie uit geheugenregister met adres *0300* (d.i. BT+**br** = *0000 +0300*) op. Ondertussen wordt de bevelenteller reeds met één opgehoogd en krijgt nu de waarde *0001*. Vervolgens:

- wordt de instructie 1131000004 geanalyseerd,
- wordt de operand berekend (0004) in het operandregister (OR),
- en wordt de opdracht uitgevoerd: de inhoud van geheugenregister *0304* (d.i. OR+**br** = *0004+0300*) wordt opgehaald en in register **R1** geplaatst. (**R1** krijgt de waarde 10.)

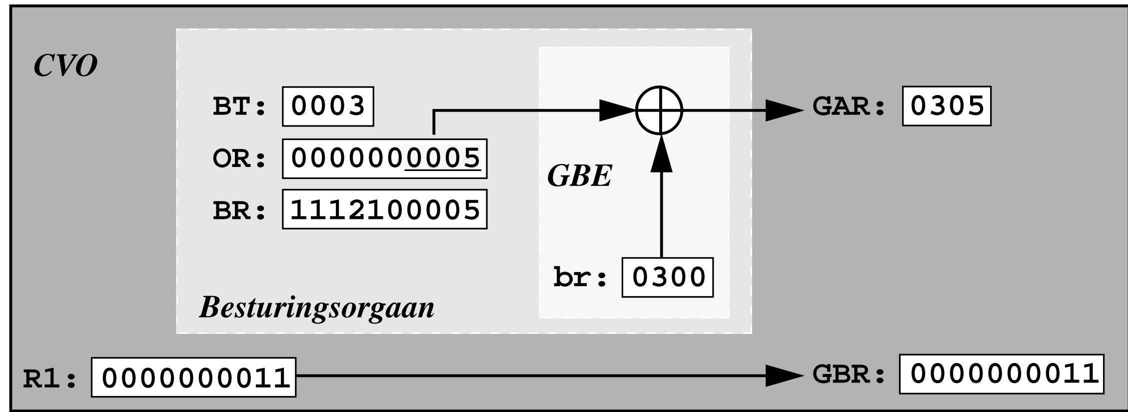
b. De volgende instructie wordt opgehaald uit geheugenregister met adres *0301* (d.i. BT+**br** = *0001+0300*). Ondertussen wordt de bevelenteller met één opgehoogd en krijgt nu de waarde *0002*. Vervolgens:

- wordt de instructie 2111100001 geanalyseerd,
- wordt de operand berekend (0001).
- en wordt de opdracht uitgevoerd: de inhoud van register **R1** wordt met één verhoogd.

c. De volgende instructie wordt gehaald uit geheugenregister met adres *0302* (d.i. BT+**br** = *0002+0300*). Ondertussen wordt de bevelenteller met één opgehoogd en krijgt nu de waarde *0003*. Vervolgens:

- wordt de instructie 1221100005 geanalyseerd,
- wordt de operand berekend (0005) in het operandregister (OR),
- en wordt de opdracht uitgevoerd: de inhoud van **R1** wordt in geheugenregister 0305 (d.i. OR+**br** = *0005+0300*) weggeborgen, enz.

Figuur 3-28 toont de inhoud van de verschillende registers op het ogenblik dat de **BIG**-instructie (**1221100005**) uitgevoerd wordt. De bevelenteller wijst op dit ogenblik reeds naar de volgende uit te voeren instructie (de **STP**-opdracht).



BT	= Bevelenteller	**GAR**	= Geheugen Adres Register
BR	= Bevelenregister	**GBR**	= Geheugen Buffer Register
OR	= Operandregister	**br**	= basisregister
CVO	= Centrale Verwerkingsorgaan	*GBE*	= Geheugenbeheereenheid

FIGUUR 3-28. De geheugenbeheereenheid

d. De volgende instructie wordt gehaald uit geheugenregister met adres ***0303*** (d.i BT+**br** = ***0003+0300***). Ondertussen wordt de bevelenteller met één opgehoogd en krijgt nu de waarde ***0004***. Vervolgens:

- wordt de instructie **9999999999** geanalyseerd,
- de opdracht wordt uitgevoerd: de bevelencyclus wordt stopgezet.

Dynamische relocatie versus indexering

Dynamische relocatie, zoals het hier uitgevoerd wordt m.b.v. een basisregister, vertoont enige gelijkenis met indexering, maar er zijn belangrijke verschillen:

- de inhoud van het basisregister blijft ***constant*** gedurende de uitvoering van het programma,
- de inhoud van het basisregister wordt —***transparant voor het programma***— bij het adres opgeteld juist vóór het in het geheugenadresregister (GAR) geplaatst wordt; merk op dat wanneer de operand niet naar het GAR gestuurd wordt, er geen dynamische relocatie plaatsvindt (bijvoorbeeld bij een **HIA.a** of een **SPR.d**, ...),
- voor het programma in uitvoering ***lijkt het alsof het vanaf adres 0000 geladen is***; op geen enkele wijze blijkt het tegendeel.

De geheugenbeheereenheid en de programmeur

Alhoewel ‘dynamische relocatie’ transparant (d.i. onmerkbaar voor de programmeur) gebeurt, zijn er wel machine-instructies nodig voor het invullen van het basisregister en het in- en uitschakelen van de geheugenbeheereenheid. In uitgeschakelde toestand worden de adressen ongewijzigd in het GAR geplaatst (dus zonder relocatie). Deze laatste optie is bijvoorbeeld bijlangrijk voor de lader zelf:

- het inladen van de instructies in het geheugen gebeurt best zonder dynamische relocatie,
- daarna zal de lader het basisregister invullen,
- de geheugenbeheereenheid wordt slechts geactiveerd op het ogenblik dat met het uitvoeren van het ingeladen programma begonnen wordt.

Tabel 3-11 toont drie extra instructies die nodig zijn voor het manipuleren van de geheugenbeheereenheid.

TABEL 3-11. Instructies voor het manipuleren van de GBE

Instructie	Verklaring
`HIB Adres`	`br` ← `Adres` `HIB` staat voor ‘**H**aal **I**n **B**asisregister’
`SGI Adres`	`BT` ← `Adres` en schakelt de geheugenbeheereenheid in: vanaf nu wordt er dynamisch gereloceerd. `SGI` staat voor ‘**S**pring en schakel de **G**eheugenbeheereenheid **I**n’. Het eerstvolgende bevel dat opgehaald wordt komt uit het geheugenregister met adres ‘`Adres`’ + `br.`
`SGU Adres`	`BT` ← `Adres` en schakelt de geheugenbeheereenheid uit: alle adressen worden ongewijzigd in het **GAR** geplaatst. `SGU` staat voor ‘**S**pring en schakel de **G**eheugenbeheereenheid **U**it’. Het eerstvolgende bevel dat opgehaald wordt komt uit het geheugenregister met adres ‘`Adres`’.

In de opgaven die volgen op deze paragraaf, wordt dieper ingegaan waarom het in- en uitschakelen van de geheugenbeheereenheid samen met een sprong gebeurt.
Op de DRAMA-machine is er geen geheugenbeheereenheid aanwezig, bijgevolg is er geen basisregister beschikbaar en bestaan deze instructies niet. Het is echter mogelijk dat in een nieuwe release van de simulator een geheugenbeheereenheid voorzien wordt.

Gevolgen voor de vertaler en lader

De zojuist beschreven techniek van dynamische relocatie heeft een grote invloed op de complexiteit van enkele programma's: de vertaler en de lader.

De **vertaler** is aanzienlijk vereenvoudigd[1]:

- hij mag steeds veronderstellen dat het programma vanaf adres `0000` geladen is;
- bovendien moet hij geen relocatietabel opstellen.

Het laderschema komt grotendeels overeen met die van de **absolute lader**[2], mits één kleine wijziging:

- het invullen van het basisregister en het inschakelen van de GBE juist voor de uitvoering van het geladen programma gestart wordt.

1. Helaas is de vereenvoudiging maar van korte duur! Indien we op de DRAMA-machine een binder voorzien, zal de vertaler toch nog steeds reloceerbare code moeten genereren. Zie volgende paragraaf.
2. We bedoelen hier de reeds gewijzigde absolute lader, namelijk deze die in staat is een programma op een willekeurige plaats in het geheugen in te lezen.

Opgaven

1. Wat is de taak van een lader? Waarom wordt hiervoor een apart programma voorzien?
2. Geef een bondig overzicht van de verschillende laderalgoritmes.
3. Beschrijf de taken die een absolute lader moet uitvoeren. Geef de DRAMA-code voor deze lader.
4. Het **`LAADGR`**-directief, de **`lengte`**-aanduiding en de **`laadadres`**-aanduiding werden ingevoerd. Voor wie zijn ze bedoeld? Wie genereert ze? Wat betekenen ze precies?
5. Gegeven het volgende uitvoerbaar programma. Geef aan hoe dit programma door de absolute lader in het geheugen geplaatst wordt, m.a.w. geef de inhoud van de geheugenregisters.

```
0000000011   ← lengte
0000000100   ← laadadres
1111700010
2331700108
2131700109
1221700110
1112070000
7299999999
7399999999
9999999999
0000001234
9999999990
0000000000
#symbolen
A   0108
B   0109
C   0110
```

Kijk even na of de adressen kloppen. Het uitvoerbaar programma is de vertaling van het volgende bronprogramma:

```
LAADGR  100
HIA.w   R7,10
VER     R7,A
OPT     R7,B
BIG     R7,C
HIA     R0,R7
DRU
NWL
```

```
    STP
A:  1234
B:  -10
C:  RESGR   1
EINDPR
```

Wat is het resultaat van de uitvoering van dit programma?

6. Wat is een relocerende lader? Geef een bondig overzicht van de taken die deze lader moet uitvoeren.
7. Wat is een relocatietabel? Door wie wordt ze opgesteld? Hoe en wanneer wordt ze opgesteld? Voor wie is ze bedoeld? Welke informatie bevat ze?
8. Gegeven het volgende uitvoerbaar programma. Geef aan hoe dit programma door de relocerende lader in het geheugen geplaatst wordt (vanaf geheugenadres 300), m.a.w. geef de inhoud van de geheugenregisters.

```
0000000013        ← lengte
9999999999        ← laadadres (= −1)
1111700010
2331700009
2131700010
1131800012
1222780000
1112070000
7299999999
7399999999
9999999999
0000001234
9999999990
0000000000
0000000011
#symbolen
A     0009
B     0010
C     0011
D     0012
#relocatietabel
0001  +#LAADADRES#
0002  +#LAADADRES#
```

```
0003  +#LAADADRES#
0012  +#LAADADRES#
```

Kijk even na of de adressen kloppen. Het uitvoerbaar programma is afkomstig van de volgende broncode:

```
    HIA.w   R7,10
    VER     R7,A
    OPT     R7,B
    BIG.i   R7,D
    HIA     R0,R7
    DRU
    NWL
    STP
A:  1234
B:  -10
C:  RESGR   1
D:  A+2
EINDPR
```

Merk op dat symbolisch adres **D** verwijst naar een adresconstante **A+2** (wat het adres is van **C**). Deze adresconstante, die later via de indirectie in een **BIG.i** wordt gebruikt, moet natuurlijk ook aangepast worden. Dus heeft de vertaler voor deze constante ook een relocatieopdracht in de relocatietabel geplaatst.

9. Wat is de relocatietabel voor het volgende DRAMA-programma:

```
    HIA     R3,A
    OPT.w   R3,D-B
    VER     R3,D
    HIA     R0,R3
    DRU
    STP
A:  100
B:  200
C:  RESGR   10
D:  300
EINDPR
```

10. Vergelijk statische met dynamische relocatie. Wat bedoelt men met 'relocatie'? Hoe of door wat wordt de relocatie uitgevoerd? Indien dynamische relocatie voorzien is, worden de taken van de lader eenvoudiger of complexer? En de taken van vertaler: eenvoudiger, ongewijzigd of complexer?

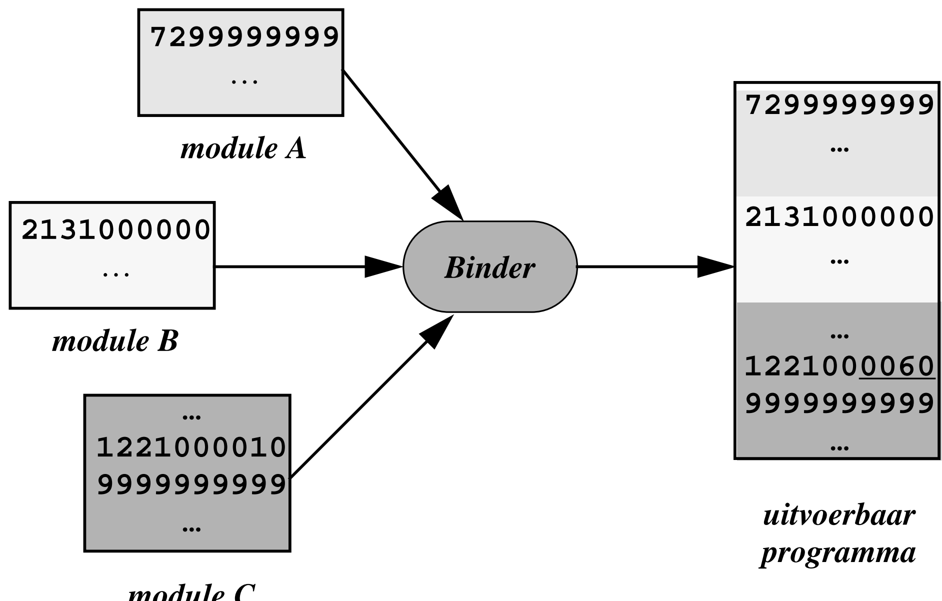
11. Veronderstel dat het in- en uitschakelen van de geheugenbeheereenheid gebeurt m.b.v. twee instructies (zonder operand):

- **`GBI`**: schakel de geheugenbeheereenheid in,
- **`GBU`**: schakel de geheugenbeheereenheid uit.

Ga na hoe de lader te werk moet gaan om de uitvoering van het net ingeladen programma te beginnen. Wat gebeurt er op het ogenblik dat de geheugenbeheereenheid ingeschakeld wordt via **`GBI`**? Welke instructie zal opgehaald worden? Zie je een probleem? Hoe zou je dit kunnen oplossen? Wordt het probleem omzeild met de **`SGI`** en **`SGU`**-instructies?

3.4 De binder

Grotere programma's worden vaak opgedeeld in kleinere stukken, ***modules*** genoemd, die elk in een afzonderlijk bestand worden opgeslagen en ook afzonderlijk worden vertaald. Het is de taak van de binder (in het Engels '*linkage editor*') om deze afzonderlijk vertaalde modules samen te voegen tot één groot uitvoerbaar programma, dat dan door de lader in het geheugen kan geladen worden. Daarom wordt de binder ook soms het 'koppelprogramma' genoemd. Zie figuur 3-29. Zoals uit de figuur blijkt, houdt het 'binden' meer in dan het achter elkaar plaatsen van de machinebevelen uit de verschillende modules. Bij sommige instructies (zie de onderlijning in het uitvoerbare programma) zijn ook de adresgedeelten aangepast.



FIGUUR 3-29. De binder of het koppelprogramma

3.4.1 Waarom een binder?

In grotere projecten wordt vaak het werk opgedeeld in kleinere deeltaken, die dan worden verdeeld over de verschillende programmeurs. Het eindresultaat wordt dan bekomen door al deze programmafragmenten samen te voegen tot één groot programma. In principe zou dit kunnen op broncodeniveau. Dit heeft echter een aantal nadelen:

- Het samenvoegen van modules tot één groot bronprogramma, dat in één keer door het assembleerprogramma wordt vertaald, is niet erg praktisch. Eén van de problemen vormen weeral de **symbolische adressen**. Elk symbolisch adres moet uniek gedefinieerd worden. Het lijkt weinig waarschijnlijk dat in deze modules, waar misschien verschillende personen

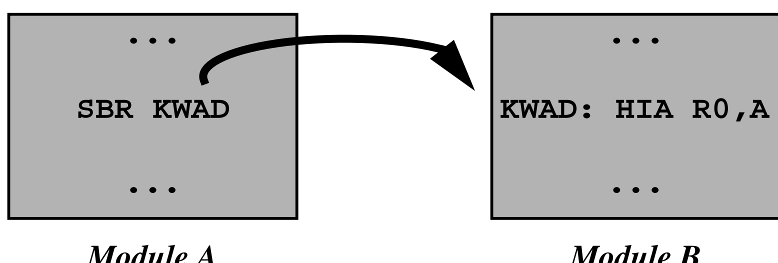
aan gewerkt hebben, geen twee keer hetzelfde etiket gebruikt zou zijn. Tenzij natuurlijk vooraf strikte afspraken gemaakt werden over de naamkeuze. Dit laatste komt echter de leesbaarheid van programma's niet altijd ten goede.

- Hoe groter het bronprogramma wordt, hoe kostelijker (in tijd) een eventuele **hervertaling** wordt na bijvoorbeeld een aanpassing van het programma (om bijvoorbeeld een fout te verbeteren). Indien de modules afzonderlijk konden vertaald worden, zou bij het aanbrengen van een verbetering in één module alleen die ene module moeten hervertaald worden. Uiteraard moet het 'binden' wel volledig worden overgedaan.
- Ook in de informaticawereld hoeven we niet steeds opnieuw het 'wiel' uit te vinden. Voor heel wat deelproblemen zijn reeds oplossingen beschikbaar. Zo zal men bij de aanschaf van een vertaler, ook een hele reeks deelprogramma's meegeleverd krijgen: subroutines die een rij van ASCII-tekens omzetten naar binaire getallen, subroutines die een aantal geheugenregisters wissen, enz. De leverancier zal echter **niet graag de broncode uit handen geven**. (Men wil immers de concurrentie niet 'helpen'.) Bovendien zijn er werkelijk honderden van dergelijke subroutines beschikbaar, zodat het niet erg praktisch is dit allemaal manueel samen te voegen. (Zie ook het begrip 'programmabibliotheek' dat in paragraaf 3.4.6 (op pag. 74) ter sprake komt.)

We kunnen dus besluiten dat het handig zou zijn indien we de modules afzonderlijk kunnen vertalen, en de vertaalde versies (die dus uit machinebevelen bestaan) laten samenvoegen door een speciaal programma, ***de binder***.

3.4.2 Modules

Een module bevat een gedeelte van het programma, bijvoorbeeld het hoofdprogramma, of een aantal procedures of functies. Ze zal dus ook een aantal ***verwijzingen*** (referenties) naar symbolische adressen bevatten die niet in de module zelf gedefinieerd zijn, maar in andere modules. Bijvoorbeeld, in de module ***A*** zou een subroutineoproep kunnen voorkomen naar een rouinte '**`KWAD`**', die tot een andere module ***B*** behoort. Figuur 3-30 schetst zo'n situatie.



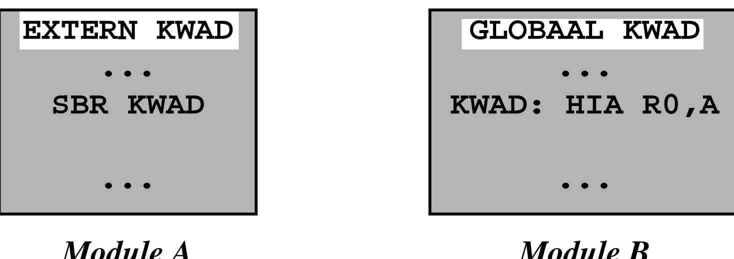
FIGUUR 3-30. Twee modules A en B

Bij het vertalen van module ***A***, zal de vertaler een foutmelding geven omdat verwezen wordt naar een niet gedefinieerd symbolisch adres '**`KWAD`**'. De vertaler moet ervan op de hoogte gebracht worden dat het symbolisch adres '**`KWAD`**' elders gedefinieerd is, **extern** aan deze module. Daarom voorzien we een extra ***vertalerdirectief***, **`EXTERN`**. Als operand mag een lijst van ***externe symbolische adressen*** opgegeven worden.

Symbolische adressen zijn meestal alleen belangrijk in de module waarbinnen ze gedefinieerd zijn. Men zegt dat deze etiketten ***lokaal*** zijn t.o.v. de module. Een uitzondering hierop zijn de symbolische adressen (zoals '**KWAD**' in module ***B*** waarnaar verwezen kan worden vanuit een andere module. Deze etiketten worden ***globaal*** genoemd. Om de vertaler toe te laten een onderscheid te maken tussen beide soorten, wordt het ***vertalerdirectief*** **GLOBAAL** ingevoerd. Als operand mag een lijst van ***globale symbolische adressen*** opgegeven worden.

In het voorbeeld van figuur 3-30 zal in module ***A*** een **EXTERN**-directief nodig zijn en in module ***B*** een **GLOBAAL**-directief. Zie figuur 3-31.

- **EXTERN KWAD**: (in module ***A***) duidt aan dat in deze module ***verwijzingen voorkomen naar een symbolisch adres (KWAD) dat buiten deze module gedefinieerd is***,
- **GLOBAAL KWAD**: (in module ***B***) duidt aan dat deze module een symbolisch adres (**KWAD**) ***definieert*** waarnaar vanuit een andere module mag gerefereerd worden.



FIGUUR 3-31. EXTERN- en GLOBAAL-directieven

Beide directieven worden door de programmeur opgegeven en zullen door de vertaler — in een gewijzigde vorm — doorgegeven worden aan de binder.

3.4.3 Bestanden

Elke module wordt in een ***afzonderlijk bestand*** bewaard. Het is de programmeur die uiteindelijk beslist hoe een programma verdeeld wordt in verschillende modules. We verkiezen de benaming 'module' boven deze van 'bestand' (alhoewel elke module in een bestand bewaard wordt), omdat een bestand ook andere informatie kan bevatten dan programma's, bijvoorbeeld een tekst, een foto, een geluidsfragment, een film of andere (binaire) gegevens.

Merk op dat programma's niet zomaar willekeurig opgesplitst mogen worden. Voor een programma dat in een ***lagere programmeertaal*** geschreven is, zijn er weinig beperkingen: elk bevel moet tot één module behoren. Het is bijvoorbeeld niet toegestaan **HIA R2,ABC** in twee (of meer) delen te splitsen, en elk deel in een aparte module te plaatsen; bijvoorbeeld: **HIA** in de ene module en **R2,ABC** in de andere. Natuurlijk moet de programmeur ervoor zorgen dat de nodige **EXTERN**- en **GLOBAAL**-directieven in elke module voorzien worden.

Voor programma's die in een ***hogere programmeertaal*** geschreven zijn, zijn er meer beperkingen. Bijvoorbeeld een procedure of functie in C mag niet gesplitst worden over twee (of meer) modules. Wel is het toegelaten om het hoofdprogramma en elke procedure (of functie) in een afzonderlijke module te plaatsen.

3.4.4 Hoofdprogramma

Wanneer twee of meer modules worden samengevoegd, is het niet direct duidelijk waar de uitvoering van dit samengevoegde programma precies moet beginnen. Met andere woorden, we moeten aangeven welke module het hoofdprogramma[1] bevat. Daarom wordt nog een extra vertalerdirectief, **STARTPR**, ingevoerd, die door de programmeur moet opgegeven worden. Het directief heeft als argument het (symbolisch) adres van de eerste instructie die moet uitgevoerd worden. De vertaler zal dit directief moeten vertalen naar informatie voor de binder/lader: de **start**-aanduiding. Merk op dat **STARTPR** ook nuttig is wanneer we slechts één module hebben. Indien we vooraan plaats reserveren voor de variabelen, hoeven we niet als eerste opdracht een **SPR** over die variabelen uit te voeren. Het volstaat met **STARTPR** de eerste uit te voeren instructie aan te duiden. De lader zelf moet slechts een kleine wijziging ondergaan: na laden (en eventueel relocering) zal een sprong uitgevoerd worden naar het (gereloceerde) adres dat in de **start**-aanduiding[2] is aangegeven. (Zie ook voorbeeld 3-5.)

Voorbeeld 3-5. *In de rest van deze paragraaf zullen we verwijzen naar de volgende twee modules* **A** *en* **B**. *Module* **B** *bevat twee zeer eenvoudige procedures die één invoerparameter verwachten (in register* **R1**) *en als resultaat respectievelijk de derdemacht en het kwadraat berekenen in register* **R0**. *(In de praktijk zullen dergelijke procedures nooit op zo'n manier geïmplementeerd worden.) Module* **A** *drukt eerst het verschil tussen de adressen van* **EIND** *en* **MACHT** *af en daarna het kwadraat van* **1000**.

```
L1:           STARTPR PRG
L2:           EXTERN  KWAD,EIND
L3:  X:       100;1000
L4:  PRG:     HIA.w   R0,EIND-MACHT
L5:           DRU
L6:           HIA     R1,X+1
L7:  MACHT:   SBR     KWAD
L8:           DRU
L9:           STP
L10:          EINDPR
```

module A

```
L1:           GLOBAAL DDM,KWAD,EIND
L2:  DDM:     SBR     KWAD
L3:           VER     R0,R1
L4:           KTG
L5:  KWAD:    HIA     R0,R1
L6:           VER     R0,R1
L7:  EIND:    KTG
L8:           EINDPR
```

module B

1. In een hogere programmeertaal heeft het hoofdprogramma een vaste naam (vb. **main** in C) of wordt het via een speciaal sleutelwoord aangeduid (vb. **program** in Pascal).
2. De **start**-aanduiding is het adres van de eerste uit te voeren instructie. Het is het derde getal in de hoofding. Indien de vertaler geen **STARTPR**-directief vindt, dan genereert hij (**-1**) als **start**-aanduiding, wat een ongeldige waarde is en aangeeft dat deze objectmodule geen hoofdprogramma bevat.

Deze twee modules (die elk in een afzonderlijk bestand bewaard worden) worden ***afzonderlijk vertaald*** door de vertaler. Het resultaat van deze vertaling is echter ***geen uitvoerbaar programma***, alhoewel het er erg op lijkt. Bij een aantal DRAMA-machinebevelen kon de vertaler nog niet het correcte operandgedeelte berekenen (omdat hiervoor de andere module nodig is). De vertaalde module zullen we ***objectmodule*** noemen. Ze bestaat uit machinecode en extra informatie voor de binder. Het spreekt vanzelf dat deze machinecode reloceerbaar moet zijn, vermits deze modules — na samenvoeging — niet allemaal vanaf adres `0000` kunnen beginnen. Om vergissingen uit te sluiten zal vooraan in een objectmodule en in een uitvoerbaar programma het type vermeld staan: de zogenaamde **`type`**-aanduiding, die zowel door de vertaler als door de binder gegenereerd wordt. Zo kan de lader weigeren een objectmodule te laden en uit te voeren. We voorzien twee types: **`OBJECTMODULE (-1)`** en **`UITVOERBAAR (-9)`**.

Vanaf nu veronderstellen we dat de vertaler steeds objectmodules produceert (het type zal steeds **`OBJECTMODULE`** zijn). Zelfs indien het programma slechts uit één module bestaat, zal de vertaler toch een objectmodule genereren, die dan door de binder omgezet wordt tot een uitvoerbaar programma. (In dit geval is de omzetting uiterst eenvoudig: alleen het type moet aangepast worden.) Aangezien objectmodules steeds reloceerbaar moeten zijn (zie verder) vind je geen laadadresaanduiding in de hoofding. Het resultaat van het binden kan echter wel of niet reloceerbaar zijn. De hoofding van het uitvoerbaar programma zal dus wel nog een laadadresaanduiding hebben.

Informatie voor de binder

Welke **extra informatie** moet de vertaler doorspelen aan de binder? Vermits elke objectmodule reloceerbaar is, zal ze ook een relocatietabel bevatten.

Aangezien de vertaler de ‘waarde’ van een extern symbolisch adres niet kent, zal hij hiervoor gemakshalve de waarde `0000` nemen. Omdat dit meestal niet correct is, zal de vertaler ook aangeven hoe het operandveld binnen een instructie (of een constante) moet gewijzigd worden met de echte waarde van dit symbolisch adres. Dit is volledig analoog aan de informatie die in de relocatietabel staat. Vandaar dat we beide soorten informatie (om te reloceren en om te binden) in dezelfde tabel zullen stoppen: de ***relocatie- en bindingstabel***[1].

Tenslotte zal de vertaler ook een ***uitgebreide symbooltabel*** toevoegen aan de objectmodule. Naast de symbolische naam en de waarde wordt ook het type van elk symbolisch adres vermeld: lokaal (d.w.z. alleen binnen deze module te gebruiken), globaal (d.w.z. hier gedefinieerd maar ook buiten deze module te gebruiken), en extern (d.w.z. hier gebruikt, maar in een andere module gedefinieerd).

Afspraak

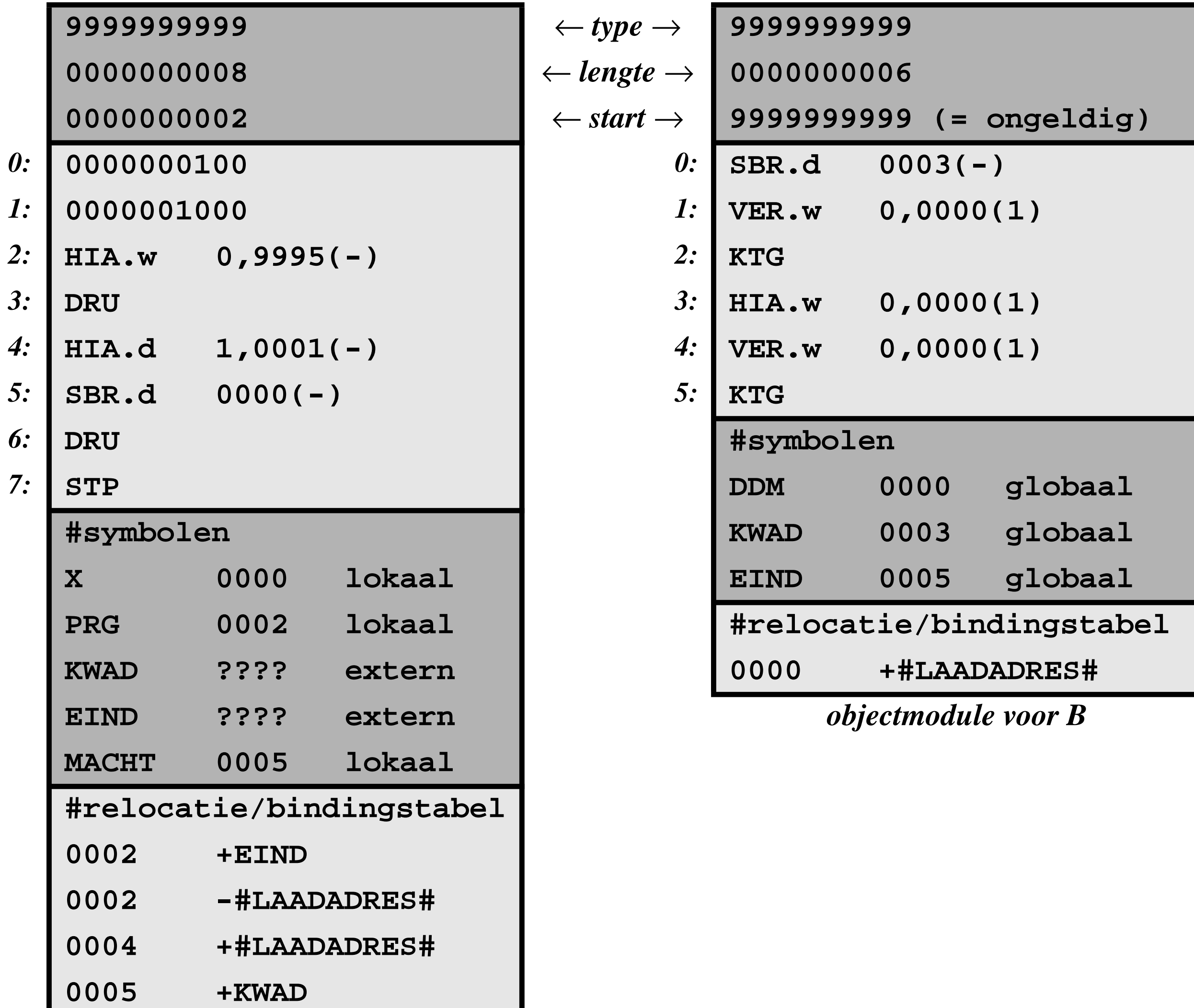
Omdat het in een geschreven cursus en tijdens het college moeilijk is met decimale notaties te werken, zullen we de inhoud van objectmodules en uitvoerbare programma’s op een vriendelijker wijze voorstellen:

1. Alle lijnen die **`#LAADADRES#`** bevatten zijn relocatieopdrachten; de overige zijn bindingsopdrachten.

- ***de mnemotechnische naam wordt behouden,***
- ***de interpretatie (indien van toepassing) wordt expliciet weergegeven,***
- ***de operanden (indien nodig) worden decimaal voorgesteld, en gescheiden door een komma, (dus alle symbolische referenties zijn verdwenen),***
- ***alle getallen worden voorgesteld in zoveel cijfers als zij werkelijk plaats innemen,***
- ***negatieve getallen worden in de 10-complementnotatie voorgesteld,***
- ***indien een indexregister gebruikt wordt, wordt deze tussen haakjes achter de operanden geplaatst, anders wordt een streepje geplaatst,***
- ***de machinebevelen/constanten worden voorafgegaan door hun relatief adres.***

Het is een triviale taak om deze voorstelling om te zetten in decimale vorm.

De vertaler zal voor de modules uit voorbeeld 3-5 (op pag. 67) de volgende twee objectmodules genereren. De verklaring volgt na de figuur.



FIGUUR 3-32. Vertaling van Modules A en B uit voorbeeld 3-5

Objectmodule A bevat een geldige `start`-aanduiding. Het geeft aan dat deze module de code van het hoofdprogramma bevat, en dat de uitvoering bij ***relatief adres***[1] `0002` moet beginnen. Bij de andere modules zal de `start`-aanduiding −1 zijn, wat een ongeldige waarde is. Wanneer modules worden samengevoegd zal de binder dit 'startadres' ook moeten aanpassen.

Merk op dat de vertaler de volledige symbooltabel heeft toegevoegd, met de extra vermelding of het symbolisch adres lokaal, globaal of extern gedefinieerd is. De binder zal alleen de informatie over ***globale etiketten*** gebruiken. (Voor externe symbolen kan geen adres gegeven worden, vandaar de vraagtekens; in feite mag hier om het even wat staan, er zal toch geen rekening mee gehouden worden.)

Bestudeer eerst even hoe de vertaler de inhoud van de objectmodules bepaald heeft. Vooral lijnen `(L4)`, `(L6)` en `(L7)` van ***module A*** zijn interessant (zie voorbeeld 3-5 (op pag. 67)).

`(L4)`: De expressie `EIND-MACHT` evalueert tot `-5` (dit wordt in 10-complement `9995`): voor `EIND` (een ***extern symbolisch adres***) wordt `0000` genomen, aangezien het een extern symbolisch adres is, en `MACHT` heeft de waarde `0005`, voor zover deze module vanaf `0000` geladen wordt. Bijgevolg vind je in de ***relocatie/bindingstabel*** twee lijnen voor deze instructie, één voor de binding, en één voor de relocatie:

```
0002    +EIND
0002    -#LAADADRES#
```

`(L6)`: De expressie `X+1` evalueert tot `0001`: `X` heeft waarde `0000`, ook weer voor zover de module vanaf `0000` geladen wordt. Dus is er een relocatieopdracht nodig.

```
0004    +#LAADADRES#
```

`(L7)`: Tenslotte is `KWAD` opnieuw een ***extern symbolisch adres***, waarvoor de vertaler `0000` heeft genomen. Natuurlijk is ook hier weer een bindingsopdracht nodig:

```
0005    +KWAD
```

3.4.5 De taken van de binder

Enkele taken van de binder komen overeen met die van de relocerende lader. De binder geeft als resultaat een uitvoerbaar programma, dat al dan niet in zijn geheel reloceerbaar kan zijn. De binder zal de volgende acties moeten ondernemen:

1. **plaats voorzien** voor de verschillende modules; immers, ze kunnen niet allemaal vanaf `0000` beginnen. Dit is een eenvoudige taak, aangezien elke objectmodule de lengte van die module bevat.
2. de **machinecode** van al de modules **samenvoegen**,
3. het **startadres** voor de uitvoering bepalen, op basis van de (enige geldige) `start`-aanduiding en de plaats waar die module in het uitvoerbaar bestand komt te staan,
4. de **code** een eerste maal **reloceren** (behalve voor de eerste module); alle overige modules worden immers opgeschoven,

1. Voor elke module is het adres van de eerste instructie `0000`. Alle adressen zijn dus ***relatief t.o.v. de plaats waar de module in het geheugen terecht zal komen.***

5. **de externe referenties in rekening brengen**; (dit is het eigenlijke binden),
6. indien het resultaat zelf nog reloceerbaar moet zijn, dan moet de binder ook een **relocatietabel** opstellen.

Het reloceren en het binden zijn de moeilijkste taken. Ze vereisen dat de binder respectievelijk beschikt over een ***allocatietabel*** en een ***globale symbooltabel***. In de allocatietabel wordt bijgehouden welke plaats voorzien is voor elke module; de globale symbooltabel geeft weer wat de waarde is van elk globaal symbolisch adres. Deze tabellen kunnen slechts opgesteld worden nadat alle modules onderzocht werden.

De binder zal dus — zoals de vertaler — in twee stappen werken:

- in een eerste stap worden de allocatietabel en de globale symbooltabel opgesteld,
- in een tweede stap wordt de code samengevoegd, eventueel gereloceerd en gebonden.

Stap 1: Allocatie- en globale symbooltabel

Deze stap bestaat uit twee secties: de allocatiesectie en de symbolensectie.

a) De allocatiesectie

De binder leest de modules achtereenvolgens in, en bepaalt het **beginadres** voor elke module (d.i. het beginadres van de vorige module plus de lengte van die module). Deze beginadressen worden in de ***allocatietabel*** bijgehouden.

b) De symbolensectie

De machinecode wordt in deze stap overgeslagen. De `#symbolen`-sectie wordt gebruikt om de globale symbooltabel op te stellen:

- ***lokale*** en ***externe*** symbolische adressen worden voorlopig genegeerd,
- ***globale etiketten*** worden samen met hun waarde, die zelf verhoogd wordt met het nieuwe beginadres van de module, in de globale symbooltabel toegevoegd.

Tabel 3-12 toont de inhoud van deze twee tabellen voor de modules ***A*** en ***B***.

TABEL 3-12. De allocatie- en globale symbooltabel.

Allocatietabel	
Module	**Beginadres**
`Module A`	`0000`
`Module B`	`0008`

Globale Symbooltabel	
Etiket	**Adres**
`DDM`	`0008`
`KWAD`	`0011`
`EIND`	`0013`

De binder begint met module *A*. Het beginadres is `0000`. Bijgevolg wordt (`Module A`, `0000`) toegevoegd aan de allocatietabel. De lengte van de machinecode bedraagt `8`. Er zijn geen globale etiketten in de `#symbolen`-sectie opgenomen.

Daarna is module *B* aan de beurt. Het beginadres is `0000+8=0008`. Dus wordt (`Module B`, `0008`) toegevoegd aan de allocatietabel. De lengte van de machinecode bedraagt `6`. Er bevinden zich drie globale etiketten in de `#symbolen`-sectie van de objectmodule:

- `DDM` heeft een relatieve waarde `0000`; vermits het beginadres van module *B* `0008` bedraagt, is de waarde van `DDM` `0000+0008=0008`. Bijgevolg wordt (`DDM`, `0008`) aan de globale symbooltabel toegevoegd.
- Analoog worden (`KWAD`, `0003+0008=0011`) en (`EIND`, `0005+0008=0013`) in de globale symbooltabel geplaatst.

Stap 2: Samenvoegen, reloceren en binden

In de tweede stap wordt de code van alle modules achter elkaar geplaatst, en gereloceerd en gebonden waar nodig. Het reloceren gebeurt m.b.v. de allocatietabel, het binden m.b.v. de globale symbooltabel.

Het resultaat van het binden van de objectmodules A en B is weergegeven in figuur 3-33. De onderlijnde operanden zijn gereloceerd en/of gebonden.

```
       9999999991 (type)
       0000000014 (lengte)
       9999999999 (laadadr)
xxxx:  0000000002 (start)
0000:  0000000100
0001:  0000001000
0002:  HIA.w  0,0008(-)
0003:  DRU
0004:  HIA.d  1,0001(-)
0005:  SBR.d  0011(-)
0006:  DRU
0007:  STP
0008:  SBR.d  0011(-)
0009:  VER.w  0,000(1)
0010:  KTG
0011:  HIA.w   0,0000(1)
0012:  VER.w   0,0000(1)
0013:  KTG
       #symbolen
       X        0000   lokaal
       PRG      0002   lokaal
       MACHT    0005   lokaal
       DDM      0008   lokaal
       KWAD     0011   lokaal
       EIND     0013   lokaal
       #relocatietabel
       0004     +#LAADADRES#
       0005     +#LAADADRES#
       0008     +#LAADADRES#
```

FIGUUR 3-33. Het uitvoerbaar programma resulterend uit objectmodules A en B

De binder kwam tot deze waarden op de volgende wijze:

(xxxx): De `start`-aanduiding bevatte oorspronkelijk de waarde `0002`. Hierbij moet de waarde van `#LAADADRES#` van module *A* bijgeteld worden (d.i. de waarde van `Module A` in de allocatietabel; aangezien module *A* niet gereloceerd werd, blijft het startadres ongewijzigd). Dus: `0002+0000=0002`.

(0002): De oorspronkelijke waarde is `9995`; bij deze waarde moet `EIND` bijgeteld worden, en `#LAADADRES#` van module *A* afgetrokken (alles modulo `10000`). Dus: `9995+0013-0000=0008` (modulo `10000`).

(0004): Hier stond oorspronkelijk `0001`; bij deze waarde moet `#LAADADRES#` van module *A* bijgeteld worden. Dus: `0001+0000=0001`.

(0005): Oorspronkelijk stond hier `0000`; bij deze waarde moet `KWAD` bijgeteld worden. Dus: `0000+0011=0011`.

(0008): Tot slot moet `0003` aangepast worden, vermits module *B* gereloceerd werd (`#LAADADRES#` van module *B* is `0008`; zie de allocatietabel). Dus `0003+0008=0011`.

Men kan gemakkelijk nagaan dat hetzelfde resultaat was bekomen indien de programmeur de broncode van de twee modules had samengebracht in één bestand, en alles tesamen vertaald.

In het uitvoerbaar programma werd tenslotte ook nog een `#symbolen`-sectie opgenomen, om eventueel door een speurprogramma gebruikt te worden. Ze is de samenvoeging van alle lokale en globale symbolische adressen. De waarde die met elk etiket geassocieerd is, is de oorspronkelijke waarde verhoogd met het beginadres van de module waarbinnen het gedefinieerd is. Aangezien een (lokaal) symbolisch adres in verschillende modules gebruikt kan zijn, kan een bepaalde naam meer dan één keer voorkomen in de `#symbolen`-sectie. Globale symbolische adressen zijn echter uniek.

Merk ook op dat het uitvoerbaar programma nog steeds een relocatietabel bevat. Ook hier weer is de regel eenvoudig: elke relocatieopdracht blijft behouden (mits aanpassing van het relatief adres). Elke bindings-opdracht wordt vervangen door een relocatieopdracht. Tenslotte kan deze tabel vereenvoudigd worden door tegengestelde relocatieopdrachten tegen elkaar te laten wegvallen. Links staat de relocatie/bindingsopdracht zoals ze in de objectmodules voorkomen, rechts zoals ze in het uitvoerbaar programma voorkomt. De eerste twee lijnen in de rechter tabel heffen elkaar op.

TABEL 3-13. Relocatie/bindingstabellen in de modules en het uitvoerbaar bestand.

objectmodule A			uitvoerbaar bestand	
0002	+EIND	0002+0000=	0002	+#LAADADRES#
0002	-#LAADADRES#	0002+0000=	0002	-#LAADADRES#
0004	+#LAADADRES#	0004+0000=	0004	+#LAADADRES#
0005	+KWAD	0005+0000=	0005	+#LAADADRES#
objectmodule B				
0000	+#LAADADRES#	0000+0008=	0008	+#LAADADRES#

Opmerkingen

Welke invloed heeft dynamische relocatie op de **binder**? Weinig. Indien er hardware voorzien is voor dynamische relocatie, moet de binder geen relocatietabel toevoegen aan het uitvoerbaar programma.

Merk op dat het afzonderlijk vertalen van de modules de **vertaler** aanzienlijk complexer heeft gemaakt:

- hij moet externe symbolische adressen kunnen verwerken,
- hij moet aan elke objectmodule een relocatie/bindingstabel toevoegen

Dit laatste is nodig omdat de binder een ‘statische relocatie’ uitvoert. Indien we de hardware nog complexer maken, door bijvoorbeeld voor elke objectmodule een apart basisregister te voorzien, is deze statische relocatie overbodig geworden.

Tegenwoordig wordt meer en meer ‘**dynamisch binden**’ toegepast. In plaats van op voorhand één uitvoerbaar programma samen te stellen, wordt het binden uitgesteld tot op het ogenblik waarop de objectmodule nodig is (in ons voorbeeld zou dit juist vóór de sprong naar **`KWAD`** zijn; vóór de **`SBR`**-instructie worden extra instructies ingelast die deze module lokaliseren en eventueel laden in het geheugen). Dit heeft als voordeel dat wanneer een bepaalde module niet gebruikt wordt, ze ook niet in het geheugen geladen wordt. Dit is interessant voor programma’s die veel procedures bevatten die zelden gebruikt worden (bijvoorbeeld routines die alleen nodig zijn wanneer zich een fout voordoet, …). In de cursus ‘Besturingssystemen’ worden nog een aantal andere varianten besproken.

3.4.6 Programmabibliotheken

Wanneer men zich een vertaler aanschaft, krijgt men gewoonlijk een ganse verzameling subroutines meegeleverd in objectmodulevorm. Deze objectmodules worden samen in een ***programmabibliotheek***[1] bewaard. Zo’n bibliotheek is veel handiger dan de (honderden) afzonderlijke objectmodules. De binder zal ook met programmabibliotheken kunnen werken. Wanneer een symbolisch adres nergens gedefinieerd werd, zal de binder proberen dat etiket te vinden in één van de objectmodules van de opgegeven bibliotheken. Indien zo’n objectmodule gevonden wordt, zal ze ook meegebonden worden.

1. Dit is een bestand waarin de objectmodules op een speciale manier zijn samengebracht.

3.4.7 Vertaler-, binder- en laderdirectieven

In deze (en vorige paragrafen) werden heel wat verschillende directieven ingevoerd. We vatten alles nog eens samen in tabel 3-14. Een vertalerdirectief (zoals **STARTPR**) dat aanleiding geeft tot informatie voor de binder/lader wordt op dezelfde lijn genoteerd (zie **start**).

TABEL 3-14. Vertalerdirectieven en informatie voor binder en lader.

Vertalerdirectieven	Informatie voor de Binder	Informatie voor de Lader
	type	type
	lengte	lengte
LAADGR xxxx		laadadres
STARTPR xxxx	start	start
EXTERN s, …		
GLOBAAL s, …		
	#reloc./bind.-tabel	#relocatietabel
	#symbolen	
RESGR n		
EINDPR		

Indien we enkel een absolute lader ter beschikking hebben, zal de binder een niet-reloceerbaar uitvoerbaar programma moeten produceren. In dit geval zouden we de binder kunnen informeren waar het uitvoerbaar programma moet geladen worden.

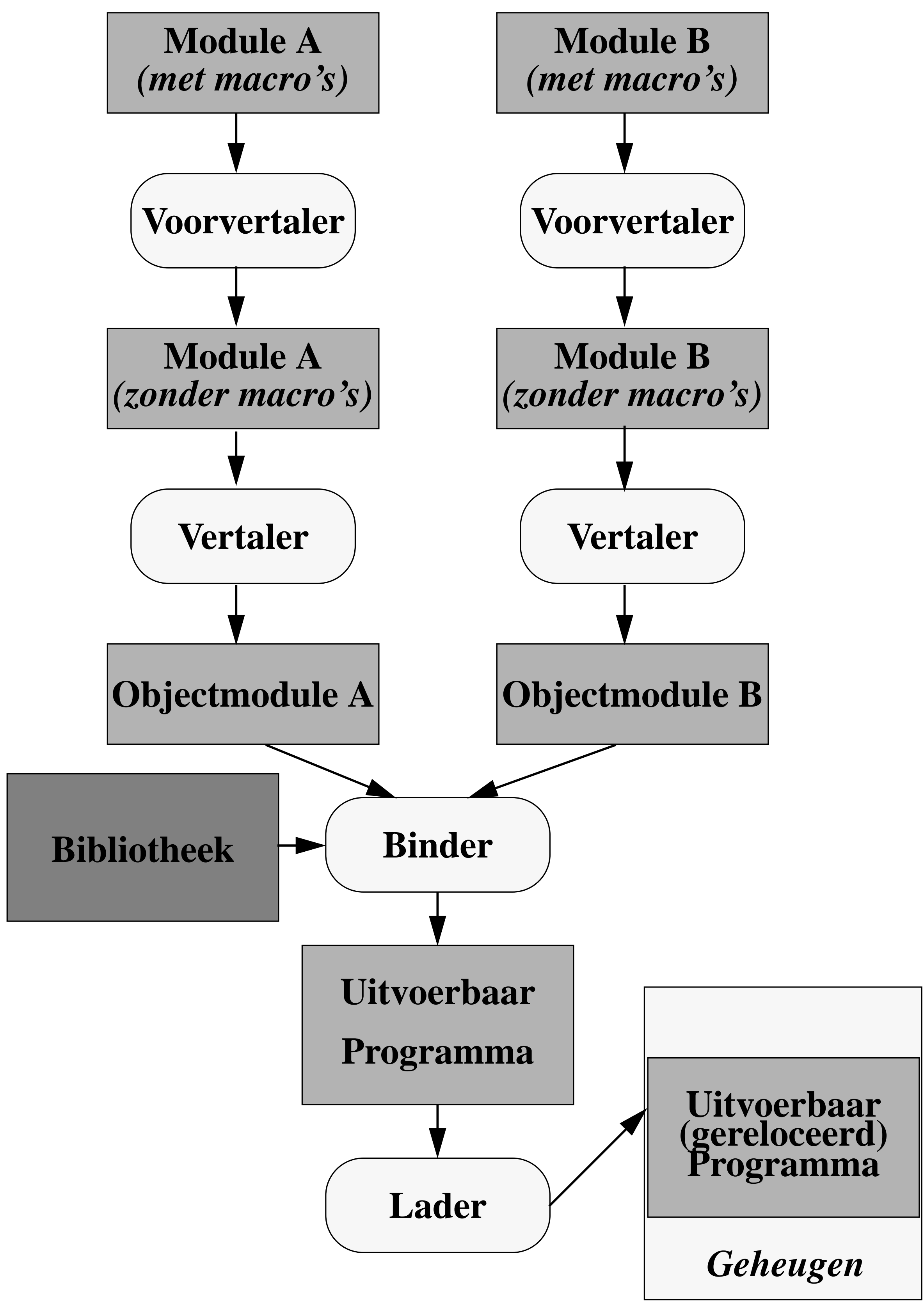
3.4.8 Van bronprogramma tot uitvoering

Een programma wordt dikwijls opgedeeld in verschillende modules (die elk in een afzonderlijk bestand bewaard worden). Deze modules worden afzonderlijk ***vertaald***, daarna ***samengevoegd*** en tenslotte in het geheugen ***geladen*** zodat het programma ***uitgevoerd*** kan worden.

Figuur 3-34 (pag. 76) toont de lange weg die afgelegd moet worden alvorens een programma uitgevoerd kan worden: voorvertalen, vertalen, binden, laden en tenslotte uitvoeren.

Eenmaal het programma correct werkt, kunnen de (voor)vertalings- en bindingsstappen overgeslagen worden; d.w.z. indien ook het uitvoerbaar programma in een bestand bewaard wordt, moet bij elke volgende uitvoering alleen de lader nog in actie komen.

Indien je software aankoopt, zoals een tekstverwerker, een vertaler, … krijg je meestal alleen het uitvoerbaar programma. Dit is voldoende om dit programma te kunnen uitvoeren. Fouten kunnen echter niet meer verbeterd worden, aangezien de broncode niet beschikbaar is. Softwarefirma's brengen regelmatig nieuwe versies van hun programma's op de markt (deze worden ook soms ***upgrades*** genoemd). De nieuwe versies zijn vaak verbeteringen en uitbreidingen van de oudere versies.



FIGUUR 3-34. Van broncode tot geladen uitvoerbaar programma

Opgaven

1. Gegeven de volgende DRAMA-module. Hoe ziet de vertaling eruit? (Gebruik symbolische namen voor de instructies; vergeet de relocatie/bindingstabel niet!)

```
    GLOBAAL  A, B, C
    EXTERN   SUB, ARRAY
    STARTPR  P
A:  100
B:  RESGR    10
C:  30
P:  HIA.w    R1,0
    BIG      R1,ARRAY+4
    SBR      SUB
    OPT      R1,A
    VER.w    R1,C-B
    AFT.w    R1,SUB-ARRAY+3
    HIA      R0,R1
    DRU
    STP
EINDPR
```

2. Waarom wordt een binder voorzien?
3. Wat betekenen van de volgende directieven: **GLOBAAL**, **EXTERN**, **STARTPR**? Wat is de **#symbolen**-sectie, **#relocatie/bindingstabel**-sectie, **start**-aanduiding? Door wie worden ze gegenereerd, en voor wie zijn ze bedoeld? Waarom zijn ze nodig?
4. Wat is een objectmodule? Wat is het verschil met een uitvoerbaar programma?
5. Beschrijf de taken die een binder moet uitvoeren.
6. Wat is een relocatie/bindingstabel? Door wie wordt ze opgesteld? Welke informatie bevat ze?
7. Wat is een allocatietabel? Door wie wordt ze opgesteld? Welke informatie bevat ze?
8. Wat is een globale symbooltabel? Door wie wordt ze opgesteld? Welke informatie bevat ze?
9. Welke invloed heeft ‘het afzonderlijk vertalen van modules’ op de complexiteit van de vertaler? Kan dynamische relocatie hier iets vereenvoudigen?
10. Wat betekent ‘dynamisch binden’?

11. Gegeven de volgende twee (afzonderlijk) vertaalde modules. Geef het resultaat wanneer deze modules met elkaar gebonden worden. Is het uitvoerbaar programma nog reloceerbaar? Wat is de inhoud van de allocatietabel en de globale symbooltabel?

(a) objectmodule Prog1

```
9999999999 (type = OBJ)
0000000007 (lengte)
0000000000 (start)
---------------------------------
HIA.d      0,0005(-)
DRU
HIA.w      1,0000(-)
SBR.d      0000(-)
DRU
STP
00000000000
---------------------------------
#symbolen
ERROR      ????     extern
VAR        ????     extern
HOOFDPROG  0000     globaal
A          0006     lokaal
---------------------------------
#relocatie/bindingstabel
0000       +#LAADADRES#
0002       +VAR
0002       -ERROR
0003       +ERROR
```

(b) objectmodule Prog2

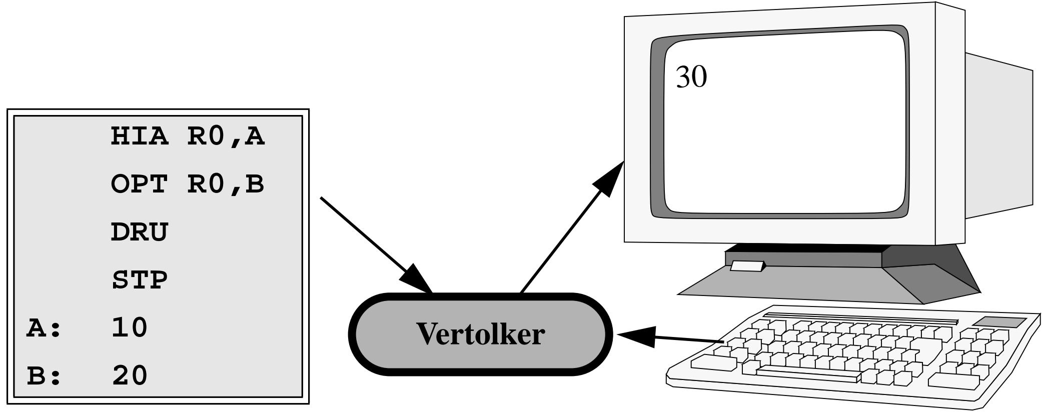
```
9999999999 (type = OBJ)
0000000005 (lengte)
9999999999 (ongeldig)
---------------------------------
HIA.w      0,0001(-)
OPT.i      0,0003(-)
KTG
00000000004
00000000100
---------------------------------
#symbolen
ERROR      0000      globaal
A          0003      lokaal
VAR        0008      globaal
---------------------------------
#relocatie/bindingstabel
0001       +#LAADADRES#
0003       +#LAADADRES#
```

Kan je — uitgaande van de objectmodules — de bronprogramma's reconstrueren?

3.5 De vertolker

Tot hiertoe hebben we één methode gezien om een programma dat geschreven is in een lagere (of hogere) programmeertaal uit te voeren op een computer: elke instructie van het bronprogramma wordt eerst omgezet in een equivalente sequentie van machine-instructies (dit wordt de **vertaling** genoemd), nadien wordt het resultaat van de vertaling (het uitvoerbaar programma) rechtstreeks door de processor uitgevoerd. Er zijn dus twee duidelijk gescheiden fasen: de vertalingsfase en de uitvoeringsfase.

Een andere methode bestaat erin gebruik te maken van een speciaal programma, een **vertolker** (in het Engels *'interpreter'*[1]). Deze vertolker heeft als invoer het oorspronkelijke (niet-vertaalde) bronprogramma en voert het uit door voor elke ingelezen opdracht de overeenkomstige machinecode rechtstreeks uit te voeren. Dit wordt 'vertolking' of 'interpretatie' van het programma genoemd. Deze techniek vereist dus niet dat er eerst een equivalent programma in machinecode wordt gegenereerd.
Samengevat: een ***vertolker*** is een programma dat een ander programma inleest en het vertolkt. Een vertolker combineert dus de vertalingsfase en de uitvoeringsfase van een programma. Zie figuur 3-35.



FIGUUR 3-35. De werking van de vertolker.

Merk op dat het resultaat van de vertolking (het uitschrijven van `30` op het scherm) precies hetzelfde is als wanneer eerst het DRAMA-programma vertaald was geworden en daarna uitgevoerd.

1. De klemtoon valt op de tweede lettergreep, niet op de derde!

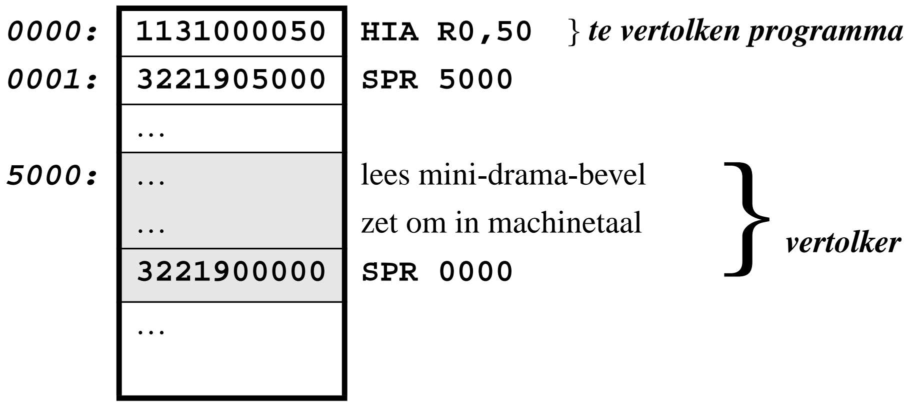
3.5.1 Werking van een vertolker

De eerste (vrij elementaire) vertolkers werkten als volgt: de vertolker leest telkens één broninstructie in, vertaalt deze naar de overeenkomstige sequentie van machine-instructies en voert deze onmiddellijk uit.

Gegeven het volgende MINI-DRAMA-programmafragment:

```
HIA  R0,50
BIG  R0,60
...
```

We veronderstellen dat de vertolker zich in het geheugen bevindt vanaf adres `5000`. (Daar bevinden zich de instructies die nodig zijn voor het inlezen van een MINI-DRAMA-instructie.) Zie ook figuur 3-36: in het donker afgebeelde gedeelte van het geheugen bevindt zich de vertolker. Wanneer de vertolker de eerste opdracht inleest: **`HIA R0,50`**, zal hij de equivalente machinecode genereren, deze ergens in het geheugen plaatsen (bijvoorbeeld in het geheugenregister met adres `0000`), op adres `0001` de machinecode voor **`SPR 5000`** plaatsen en dan een sprong uitvoeren naar adres `0000`. Hierdoor wordt de **`HIA R0,50`** uitgevoerd, en wordt daarna de uitvoering van de vertolker verdergezet (met het inlezen van de volgende MINI-DRAMA-instructie).



FIGUUR 3-36. De geheugeninhoud tijdens de vertolking.

Het zou duidelijk moeten zijn, dat tijdens de ‘vertolking’ eigenlijk twee programma’s door elkaar uitgevoerd worden: telkens wordt een stukje van de vertolker en dan weer een stukje van het te vertolken programma uitgevoerd. Deze uitvoeringen mogen niet met elkaar interfereren. Bijvoorbeeld, indien de vertolker register **`R0`** gebruikt, dan mag tijdens de uitvoering van het te vertolken programma ditzelfde register niet gebruikt worden (ofwel moet de vertolker eerst de inhoud van **`R0`** ergens in het geheugen opslaan alvorens naar de code van de te vertolken opdracht te springen). Hetzelfde geldt voor geheugenregisters waarin variabelen of constanten bewaard worden.

De aandachtige lezer zal ongetwijfeld opgemerkt hebben dat deze werkwijze in veel gevallen niet zal werken. Waarom niet?

Een programma geschreven in een lagere programmeertaal is niet echt geschikt om 'vertolkt' te worden, omdat je, voor de vertolking begint, al moet beschikken over het vertaalde programma. Immers, geheugenregisters die door het programma gebruikt worden, kunnen een initiële waarde hebben. Bovendien is het mogelijk om te 'rekenen' met instructies. In figuur 3-36 (pag. 80), wordt de inhoud van geheugenregister 50 opgehaald; de programmeur had op die plaats misschien de waarde '1000' voorzien, maar hier zal de waarde die toevallig in dat geheugenregister zit opgehaald worden.

Vertolking wordt echter wel gebruikt voor programma's die geschreven zijn in een hogere programmeertaal (bijv. C, Pascal, ...) of in machinetaal (bijv. DRAMA, Java bytecode, ...).

Laten we beginnen met het laatste: een *vertolker voor vertaalde programma's*. Waarom zouden we hiervoor een vertolker gebruiken? De programma's zijn immers reeds vertaald! Een mogelijke reden is dat de programma's voor een ander soort computer vertaald zijn. De computer waarvoor ze vertaald zijn (en waarop ze bijgevolg rechtstreeks kunnen uitgevoerd worden) zullen we de ***doelcomputer*** noemen; de computer waarop we ze willen uitvoeren, de ***gastcomputer***. Als de machinetaal van de doelcomputer en gastcomputer verschillen, dan kunnen we die vertaalde programma's uiteraard niet rechtstreeks uitvoeren. Er zijn twee mogelijke oplossingen voor dit probleem:

- ofwel ontwerpen we een vertaler, die het programma omzet van de ene machinetaal naar de andere. Dit is zeker geen eenvoudige opgave! Het is best mogelijk dat een bepaalde instructie in de ene machinetaal geen tegenhanger heeft in de andere machinetaal; of een bepaald mechanisme bestaat niet in de andere taal (bijv. bestaat er geen auto-increment bij indexatie).
  Een typisch voorbeeld van zo'n vertaalprogramma vind je in een aantal Java-omgevingen: een zogenaamde JIT[1]-compiler zet vertaalde Java-programma's (bytecode) om naar programma's geschreven in de machinetaal van de gastcomputer.
- ofwel ontwerpen we een vertolker, die het gedrag van de doelcomputer nabootst op de gastcomputer. We noemen zo'n vertolker een ***simulator***.
  Een typisch voorbeeld hiervoor is de DRAMA-simulator. De doelcomputer (de DRAMA-machine) bestaat niet. Willen we toch vertaalde programma's op onze PC kunnen uitvoeren, dan hebben we een programma nodig dat op de PC kan uitgevoerd worden en het gedrag van de DRAMA-machine nabootst. In appendix B.3 (op pag. 211) vind je een gedeelte van de C-code voor deze simulator.
  Een ander voorbeeld vind je bij Java-omgevingen. Java-programma's worden vertaald naar bytecode, de machinetaal van de JVM (Java Virtual Machine). Alhoewel er reeds processoren bestaan die rechtstreeks dergelijke bytecode kunnen uitvoeren, zullen de meeste programma's uitgevoerd worden door een vertolker. Sommige programma's, zoals bladerprogramma's (*browsers* in het Engels) bevatten zo'n vertolker zodat ze in staat zijn om Applets uit te voeren.

Vertolkers worden ook vaak gebruikt bij *hogere programmeertalen* (HPT). In deze talen kan je niet 'rekenen' met bevelen, en krijgen variabelen via een toekenning een initiële waarde. Zo'n vertolker zal een gegevensstructuur bijhouden voor elke variabele van het programma, en elke opdracht (in HPT) analyseren en uitvoeren.

1. JIT staat voor Just In Time.

Laten we als voorbeeld C nemen. De opdracht '`b = 3*a;`' zou door de C-compilator (en assembleerprogramma) omgezet zijn tot drie DRAMA-instructies (een **`HIA`**, een **`VER`** en een **`BIG`**). Tijdens de uitvoering van het vertaalde programma , zullen er dus drie instructies uitgevoerd worden (voor die opdracht).
Gebruiken we echter een vertolker, dan zijn er veel meer instructies nodig voor die opdracht:

- de opdracht moet ingelezen worden;
- daarna moet de gelezen opdracht geanalyseerd worden (lexicale en semantische analyse); dit is precies hetzelfde werk als wat de compilator moest doen bij het vertalen;
- tenslotte moet de opdracht uitgevoerd worden: de inhoud van de gegevensstructuur die de variabele '`a`' bevat wordt vermenigvuldigd met drie en het resultaat wordt weggeborgen in de gegevensstructuur die voorzien is voor de variabele '`b`'.

In totaal zullen enkele tientallen of misschien zelfs enkele honderden instructies uitgevoerd worden voor die ene opdracht.

3.5.2 Voordelen

Vertolkers hebben enkele belangrijke voordelen t.o.v. vertalers. Deze voordelen komen in eerste instantie tot uiting bij hogere programmeertalen.

- Een vertolker is gemakkelijker te schrijven dan een compilator.
- Een vertolker heeft geen geheugenruimte nodig voor het volledig vertaalde programma, dat meestal heel wat meer plaats vergt dan het compactere programma in hogere programmeertaal.
- Omdat de vertolker beschikt over de broncode van het uit te voeren programma, kan hij — wanneer een fout optreedt — zinvolle foutboodschappen afdrukken. Wanneer bijvoorbeeld het C-programma de volgende opdracht bevat: `a = b / c`; en wanneer op het ogenblik van de uitvoering van deze opdracht `c` de waarde `0` zou hebben, dan kan de vertolker de volgende boodschap afdrukken:

  ```
  *** fout in lijn 32: a = b / c; *** deling door 0
  ```

  Indien het C-programma echter eerst vertaald was geweest, zou bij de uitvoering van de **`DEL`**-instructie een programmaonderbreking opgetreden zijn. Het besturingsprogramma (zie hoofdstuk 4) zou dan hooguit een foutboodschap zoals:

  ```
  *** fout op adres 670 *** overloop bij deling
  ```

  hebben kunnen afdrukken. De programmeur moet dan zelf op zoek gaan waar de fout zich precies heeft voorgedaan en wat de oorzaak ervan was.
- De vertolker hoeft zelf de machine-instructies van de computer waarop hij uitgevoerd wordt niet te kennen. De ontwerper van een vertolker zal deze laatste in een hogere programmeertaal (bijvoorbeeld C of Pascal) ontwerpen. Op elke machine die een vertaler heeft voor de programmeertaal waarin de vertolker geschreven is, kan de vertolker uitgevoerd worden. Bijvoorbeeld: stel dat de vertolker voor Perl-programma's in C is geschreven, dan kan men Perl-programma's laten vertolken op elke machine waar een C-vertaler beschikbaar is: men moet enkel eenmaal de vertolker vertalen met de C-vertaler; tijdens de uitvoering van de vertolker kunnen dan Perl-programma's geïnterpreteerd worden.

3.5.3 Nadelen van vertolking

De uitvoering van een programma via een vertolker duurt langer dan wanneer het programma uitgevoerd wordt na eerst vertaald te zijn naar machinecode.

- Bij elke uitvoering zal de vertolker een deel van het werk van een vertaler overdoen (namelijk de analyse van de bron-opdracht).

  Daarenboven is het best mogelijk dat de 'interpretatie' van de opdracht meer machine-instructies zal vergen dan wanneer de opdracht vertaald was geweest.

  Dit is niet erg wanneer het programma zich nog in de ontwikkelingsfase bevindt (d.w.z. wanneer het programma zich nog niet in de definitieve vorm bevindt of wanneer het programma nog fouten bevat, …). Na elke uitvoering wordt het programma dan toch aangepast en zou het toch opnieuw moeten vertaald worden. Bovendien kan de vertolker zinvollere foutboodschappen genereren, waardoor fouten gemakkelijker gevonden kunnen worden.
- Bij vertolking wordt elke opdracht eerst geanalyseerd en daarna uitgevoerd. Dit betekent dat opdrachten die voorkomen in een **programmalus** meer dan eens zullen geanalyseerd worden. De vertraging zal dus vooral merkbaar zijn indien er programmalussen voorkomen die verschillende duizenden keren uitgevoerd worden.

  Je mag dus niet zomaar stellen dat de totale uitvoeringstijd van een geïnterpreteerd programma ongeveer zal overeenkomen met de som van de vertalingstijd en de uitvoeringstijd van een vertaald programma. Die tijd kan veel groter zijn (indien er veel lussen in voorkomen; elke opdracht in die lus wordt keer op keer vertaald) of kleiner zijn (indien een groot gedeelte van het programma niet wordt uitgevoerd; een vertaler daarentegen vertaalt altijd een volledig programma).

Voor een programma in ***productiefase*** (d.w.z. wanneer het programma correct is bevonden en verder ongewijzigd zal gebruikt worden), sparen we behoorlijk wat uitvoeringstijd uit door voor eens en altijd het programma te vertalen.

De opgesomde nadelen worden wat afgezwakt omdat moderne vertolkers vaak een **aantal optimalisaties** doorvoeren:

- Daar het ontleden van de bron-opdrachten bij hogere programmeertalen tamelijk complex kan zijn, zal de vertolker meestal het ingelezen programma in een voor de computer meer geschikte vorm omzetten. Dit heeft het bijkomende voordeel dat wanneer een code-fragment meerdere malen wordt uitgevoerd, de ontleding slechts één keer moet gebeuren. Deze interne voorstellingswijze noemt men ook wel 'intermediaire code'. De C-opdracht `a = b + 3` zou bijvoorbeeld intern kunnen voorgesteld worden door een drie-adres-opdracht [+ | var a | var b | const 3] waarbij [var a] verwijst naar een variabelentabel. Die intermediaire code is nog zeer bondig maar wel gemakkelijker te interpreteren.
- Soms worden programmafragmenten die vaak uitgevoerd worden (bijvoorbeeld omdat ze in een lus voorkomen) direct omgezet naar machinecode en in hun geheel uitgevoerd, zodat herinterpretatie niet meer nodig is.

3.5.4 Toepassingen van vertolking

Het principe 'vertolking' kent verschillende toepassingen in de informatica:

- Indien men een **nieuwe programmeertaal** uitvindt, is het eenvoudiger hiervoor een vertolker te schrijven dan een vertaler. Indien daarenboven de vertolker zelf in een conventionele hogere programmeertaal geschreven is, is het erg waarschijnlijk dat deze vertolker op verschillende types van machines kan uitgevoerd worden. Dit in tegenstelling tot een vertaler, die natuurlijk alleen code genereert voor een bepaald type van machine.

  Nadat de nieuwe taal volledig uitgetest is, zal men er een echte compilator voor schrijven, één voor elk computertype waarop men de taal wil gebruiken.
- Hogere programmeertalen **van zeer hoog niveau** kunnen eventueel permanent door een vertolker behandeld worden. Hogere programmeertalen van zeer hoog niveau zijn talen waarbij de opdrachten niet meer specificeren 'hoe' iets moet gebeuren maar wel 'wat' er moet gebeuren. Dergelijke talen worden gekarakteriseerd door het feit dat één bron-instructie ***zeer veel machinebevelen*** genereert, bijvoorbeeld een poly-algoritme; dit is een verzameling van algoritmes, bijvoorbeeld om stelsels differentiaalvergelijkingen op te lossen. Tijdens de uitvoering van zo'n poly-algoritme wordt dan de meest geschikte methode gekozen om het probleem op te lossen (er wordt bijvoorbeeld gestart met het meest eenvoudige algoritme; indien dit geen bevredigende resultaten oplevert, wordt een ander algoritme gekozen, enz.) De vertaaltijd van zo'n 'hoog-niveau opdracht' valt dus in het niet t.o.v. de uitvoeringstijd van die opdracht, zodat het voornaamste nadeel van vertolking bijna volledig wegvalt. Een voorbeeld van zo'n programmeertaal is APL[1].
- Sommige programmeertalen zijn **dynamisch** van aard. Dit wil zeggen dat een programma in die taal geschreven, zichzelf tijdens de uitvoering van het programma kan wijzigen. Een voorbeeld hiervan is de programmeertaal LISP. De taal is gericht op het manipuleren van lijsten; het programma zelf wordt als een lijststructuur voorgesteld, en kan tijdens de uitvoering gemanipuleerd en gewijzigd worden. Voor dergelijke talen zal men bij voorkeur een vertolker ontwikkelen.
- Een derde soort toepassing is een **simulator**. Hier zal de invoer niet bestaan uit een programma geschreven in een of andere hoog-niveautaal, maar een programma in machinecode.
  We hebben reeds als voorbeeld de DRAMA- en de JVM-simulatoren aangehaald, maar er zijn nog andere nuttige toepassingen: wanneer een onderneming overschakelt van een bepaald type computer naar een ander type, kunnen zich 'conversieproblemen' voordoen. Een bedrijf is bijvoorbeeld de PC ontgroeid en beslist een Sun-werkstation aan te kopen. Programma's die men enkel in machinecode heeft (bijvoorbeeld aangekochte software) zullen niet langer uitgevoerd kunnen worden op de nieuwe machine. In plaats van te wachten tot een aangepaste versie beschikbaar is, kan men op de nieuwe machine een 'simulator' voor de oude machine ontwikkelen. Op deze wijze kunnen deze niet-overdraagbare programma's toch gesimuleerd worden op de nieuwe computer.
- Men kan nog een stapje verder gaan en de nieuwe machine 'aanpassen' zodat hij in staat is rechtstreeks vertaalde programma's voor de oude machine uit te voeren. Dit kan men als volgt bekomen: men ontwerpt een simulator voor de oude machine, deze vertaalt men naar

1. APL staat voor **A** **P**rogramming **L**anguage.

micro-code en laadt men in de nieuwe machine[1]. Vanaf dan zal de nieuwe machine zich precies gedragen zoals de oude machine. De oude programma's kunnen rechtstreeks op de nieuwe machine uitgevoerd worden. Meestal zal de nieuwe machine twee micro-programma's bevatten: één voor de nieuwe en één voor de oude computer. De 'vertolker' voor de oude computer wordt in dit geval een '**emulator**' genoemd. Een typisch voorbeeld is een VAX-machine die een PDP-11 emuleert.

Het moeilijkste gedeelte van de emulator is de emulatie van de invoer/uitvoerinstructies van de oude computer. Invoer/uitvoerinstructies worden immers vaak in parallel uitgevoerd met gewone instructies (zie ook hoofdstuk 4). Invoer/uitvoer is bovendien erg tijdsafhankelijk in tegenstelling tot gewone instructies. Indien de oude programma's veronderstellingen maakten over de relatieve snelheid van de invoer/uitvoer t.o.v. de processorsnelheid, is het mogelijk dat deze programma's niet correct zullen geëmuleerd worden.

- Naast vertaling en vertolking, bestaat er nog een derde manier om programma's uit te voeren: men ontwerpt een machine waarvan de machinetaal overeenkomt met de programmeertaal waarin het programma geschreven is. Men maakt bijvoorbeeld een computer waarvan de machine-instructies Pascal-bevelen zijn. Science-fiction? Helemaal niet. Het is voldoende een Pascal-vertolker om te zetten naar 'micro-code' en deze code te laden in de computer. Vanaf dan zal de computer zich gedragen als een 'Pascal-machine' waarop rechtstreeks Pascal-programma's kunnen uitgevoerd worden.

Opgaven

1. Wat is een vertolker? Beschrijf bondig zijn werking. Illustreer met een klein voorbeeldje.

2. Weeg de voor- en nadelen af van een vertolker t.o.v. een gewone compilator.

3. Definieer nauwkeurig het verschil tussen vertolker, simulator en emulator.

4. Geef enkele toepassingsgebieden voor vertolkers.

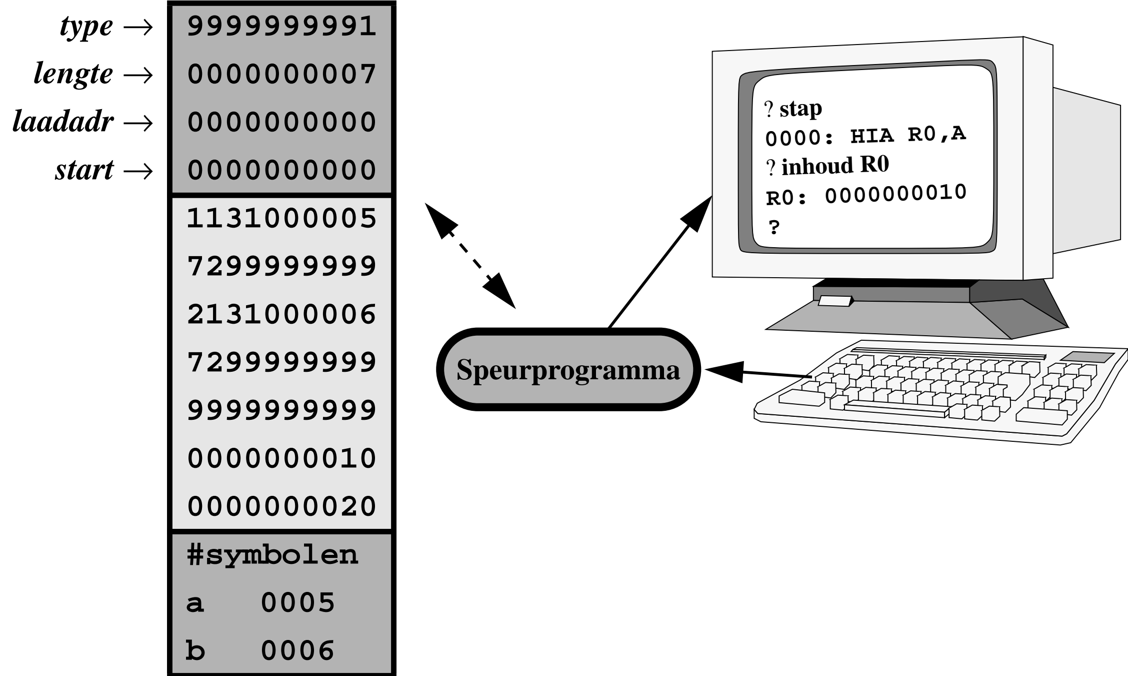
5. Waarom valt vertolking van programma's die geschreven zijn in een lagere programmeertaal nogal nadelig uit?

1. Dit veronderstelt wel dat de nieuwe machine dynamisch micro-programmeerbaar is.

3.6 Het speurprogramma

Wanneer men een programma van een zekere lengte moet schrijven, zal het resultaat bijna altijd fouten bevatten. De vertaler of vertolker kan de grammaticale of syntaxisfouten detecteren. Logische fouten kan de vertaler echter niet opsporen, en deze zullen tijdens het uittesten van het programma moeten geëlimineerd worden. In het Engels noemt men dergelijke fouten '*bugs*[1]', en het lokaliseren en verwijderen van fouten '*debugging*'.

Een speurprogramma (in het Engels *'debugger'*) helpt de programmeur bij het ***opsporen van fouten*** in een programma. Speurprogramma's zijn interactieve[2] programma's, die een uit te testen programma onder controle van de gebruiker uitvoeren. De invoer van het speurprogramma is dus enerzijds een vertaald programma (dat uitgetest moet worden) en anderzijds opdrachten van de gebruiker, die via het klavier van de terminal of het werkstation worden ingegeven. Zie figuur 3-37.



FIGUUR 3-37. Het speurprogramma.

1. De eerste computers waren reusachtige elektromechanische machines. Op een dag waren de resultaten van een programma volledig foutief. De machine werd grondig doorzocht, en toen bleek dat een kleine mot vastzat tussen een van de relays. Het feit dat men regelmatig kleine insekten moest verwijderen die aanleiding gaven tot 'fouten', was er de oorzaak van dat men voortaan sprak over 'bugs' en 'debugging'.

2. Interactieve programma's zijn programma's die invoer van de gebruiker verwachten en het resultaat tonen aan de gebruiker; dus programma's die een interactie met de gebruiker vergen.

Het repertorium van opdrachten dat door het speurprogramma begrepen wordt, kan uitgebreid of eerder beperkt zijn. Meestal zal de gebruiker de volgende bevelen kunnen ingeven:

- '**`toon`**' de code die uitgevoerd zal worden (en dit in een lagere programmeertaal); het speurprogramma kan dus machinecode omzetten naar een lagere programmeertaal (in het Engels '*disassemble*'),
- wat is de '**`inhoud`**' van bepaalde geheugenregisters?
- '**`overschrijf`**' de inhoud van een geheugenregister met een nieuwe waarde,
- druk de inhoud van een '**`accumulator`**' af,
- '**`wijzig`**' de inhoud van een accumulator,
- '**`start`**' of '**`herstart`**' de uitvoering van het programma,
- voer één instructie uit ('**`stap`**'),
- plaats een '**`breekpunt`**' (in het Engels '*break point*'), (zie verder),
- '**`verwijder`**' een breekpunt,
- geef een '**`overzicht`**' van de geplaatste breekpunten,
- ga '**`verder`**' met de uitvoering (na een breekpunt),
- …

Er worden twee programma's afwisselend uitgevoerd, enerzijds het uit te testen programma en anderzijds het speurprogramma. Het speurprogramma interpreteert de code echter niet, het reeds vertaalde en uit te testen programma wordt letterlijk in het geheugen geplaatst (eventueel gereloceerd), en later misschien gewijzigd onder controle van de gebruiker of het speurprogramma.

Laten we even kijken hoe bijvoorbeeld **breekpunten** zouden kunnen geïmplementeerd worden. Wanneer tijdens de uitvoering van het uit te testen programma aan zo'n breekpunt gekomen wordt, moet de uitvoering onderbroken worden, en moet de processor verder gaan met de uitvoering van het speurprogramma, die dan een boodschap op het scherm zal afdrukken, zoals:

```
programma onderbroken bij breekpunt 7
0020: HIA.w R1,23
```

Daarna zal het speurprogramma nieuwe opdrachten van de programmeur inlezen (bijvoorbeeld: het afdrukken van de inhoud van de accumulatoren of van bepaalde geheugenregisters, of het wijzigen ervan, of het verdergaan met de uitvoering).

Een eenvoudige manier om de uitvoering van een programma te onderbreken (en verder te gaan met een ander programma) is de uitvoering van een `SBR`-bevel, waarbij een routine van het speurprogramma wordt opgeroepen. Aangezien het terugkeeradres op de top van de stapel geplaatst wordt, kan het speurprogramma gemakkelijk nagaan waar de onderbreking opgetre-

den is. Het speurprogramma kan dus (op vraag van de gebruiker) een breekpunt plaatsen in de code door de oorspronkelijke instructie te vervangen door **`SBR BREEKPUNT`**[1], waarbij **`BREEKPUNT`** het symbolisch adres is van de procedure die breekpunten behandelt[2]. Natuurlijk moet hij de oorspronkelijke instructie bewaren. Deze moet immers uitgevoerd worden wanneer verder gegaan wordt met de uitvoering vanaf het breekpunt. Bovendien moet de instructie hersteld worden als de gebruiker het breekpunt wil verwijderen. Daarom zal het speurprogramma een breekpuntentabel bijhouden, waarin voor elk geplaatst breekpunt zowel het geheugenadres als de oorspronkelijke instructie bijgehouden worden. Voorbeeld 3-6 illustreert dit: eerst wordt het oorspronkelijke (uit te testen) programma getoond. Daarna zie je ditzelfde programma (geladen vanaf adres `0000`) waarin het speurprogramma twee breekpunten heeft aangebracht (op adres `0005` en `0009`).

Achteraan in het geheugen zie je de (gelineariseerde) breekpuntentabel (die start bij het symbolisch adres **`BRKPUNT_TABEL`**). De inhoud van deze tabel is:

TABEL 3-15. De breekpuntentabel van het speurprogramma.

adres	oorspronkelijke instructie
0005	3111100010
0009	7299999999

Voorbeeld 3-6. *Dit voorbeeld illustreert hoe een speurprogramma een breekpunt kan implementeren. Het oorspronkelijk uit te testen programma wordt eerst gegeven: zowel de broncode (links) als het uitvoerbaar programma (rechts).*

```
| Lees 10 getallen in
| Plaats ze in GETAL
| en bereken gemiddelde
| R1 : index
| R2 : partiele som
      HIA.w   R1,0
      HIA.w   R2,0
LUS:  LEZ
      BIG     R0,GETAL(R1+)
      OPT     R2,R0
      VGL.w   R1,10
```

type →	9999999991
lengte →	0000000022
laadadr →	0000000000
start →	0000000000
	1111100000
	1111200000
	7199999999
	1224010012
	2112200000
	3111100010

1. Aangezien de oorspronkelijke instructie een machine-instructie is, moet hier uiteraard ook de vertaalde versie van **`SBR BREEKPUNT`** gebruikt worden, dus **`412190vwyx`**, waarbij **`vwxy`** het adres is waar de procedure **`BREEKPUNT`** begint in het geheugen.
2. In hoofdstuk 4 zullen we een betere instructie zien om de overgang tussen twee programma's te bewerkstelligen: de onderbrekingsinstructie **`OND`**. Het gebruik van **`SBR`** levert immers een probleem op wanneer het breekpunt geplaatst wordt op een bevel dat de waarde van de conditiecode test (**`VSP`**), of een bevel dat de conditiecode niet wijzigt (bijv. **`SPR`**). Begrijp je waarom?

```
         VSP      KL,LUS
         DEL      R2,R1
         HIA      R0,R2
         DRU
         NWL
         STP
GETAL:   RESGR    10
```

```
3321700002
2412210000
1112020000
7299999999
7399999999
9999999999
0000000000
...
#symbolen:
LUS        0002
GETAL      0012
```

Aangezien het uitvoerbaar programma niet reloceerbaar is, moet het geladen worden vanaf adres `0000`*. Het speurprogramma heeft twee breekpunten geplaatst: één op de* `VGL`*-instructie (adres* `0005`*, en één op de* `DRU`*-instructie (adres* `0009`*).*

Adres	Inhoud
0000:	1111100000
0001:	1111200000
0002:	7199999999
0003:	1224010012
0004:	2112200000
0005:	4121909050
0006:	3321700002
0007:	2412210000
0008:	1112020000
0009:	4121909050
0010:	7399999999
0011:	9999999999
0012:	0000000000
	...
9000:	...
	...
9050:	...
	...
9200:	0000000005
9201:	3111100010
9202:	0000000009
9203:	7299999999

```
                 HIA.w   R1,0
                 HIA.w   R2,0
           LUS:  LEZ
                 BIG     GETAL(R1+)
                 OPT     R2,R0
                 SBR     BREEKPUNT
                 VSP     KL,LUS
                 DEL     R2,R1
                 HIA     R0,R2
                 SBR     BREEKPUNT
                 NWL
                 STP
         GETAL:  RESGR   10
```

```
     SPEURPROG:  ...
                 ...
     BREEKPUNT:  ...
                 ...
BRKPUNT_TABEL:   0005
                 VGL.w   R1,10
                 0009
                 DRU
```

Ook voor programma's geschreven in een hogere programmeertaal (zoals C) worden speurprogramma's voorzien. Dergelijke speurprogramma's worden soms '***symbolische speurprogramma's***' genoemd (in het Engels '*symbolic debuggers*'), omdat ze in staat zijn de verbinding te leggen met het oorspronkelijke (symbolische) bronprogramma. Via een speciale optie zal de programmeur aan de compilator melden dat hij een dergelijk speurprogramma wil gebruiken. De compilator zal dan extra informatie toevoegen aan de objectmodule (o.a. de naam van het bronbestand waarvan deze objectmodule de vertaling is, enz.) en bovendien ervoor zorgen dat de gegenereerde machinecode 'systematischer' is (elk bronbevel komt overeen met een bepaalde sequentie van machinebevelen) zodat het speurprogramma niet in de war geraakt; met andere woorden, er zullen geen optimalisaties doorgevoerd worden in de gegenereerde code (zoals het uitsparen van enkele instructies, …).

Opgaven

1. Wat is een speurprogramma? Waartoe dient het?
2. Geef enkele voorbeelden van opdrachten die men aan een speurprogramma kan geven.
3. Wat zijn breekpunten? Hoe kan men ze eenvoudig implementeren?
4. De methode om breekpunten te plaatsen (namelijk het vervangen van de oorspronkelijke instructie door een SBR-instructie) werkt niet indien een breekpunt geplaatst wordt op een instructie die de conditiecode niet wijzigt! Wat loopt er mis indien je een breekpunt plaatst op de `SPR`-instructie in het volgende programmafragment?

```
            HIA    R1,A
            VGL    R1,B
            SPR    VERDER
        A:  10
        B:  15
   VERDER:  VSP    KL,KLEINER
            STP
  KLEINER:  ...
```

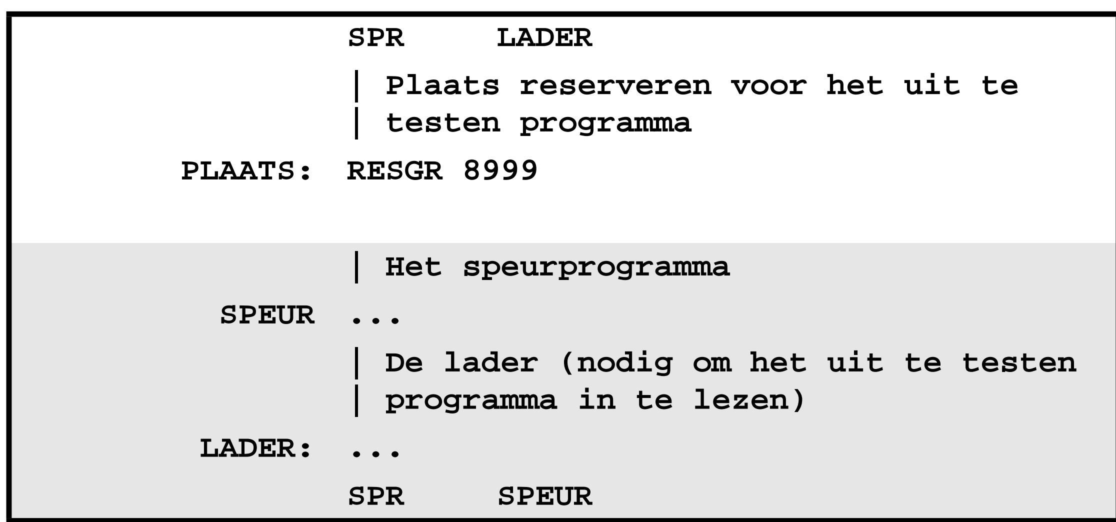
5. Ontwerp zelf een eenvoudig speurprogramma. Er worden `6` bevelen voorzien genummerd van `1` t.e.m. `6`. De volgende tabel geeft de betekenis van deze bevelen.

Bevel	Betekenis
1	start/herstart de uitvoering van het programma (op adres `0000`)
2	plaats een breekpunt (het speurprogramma leest daarna het adres in waarop het breekpunt geplaatst moet worden)
3	laat een breekpunt weg (het speurprogramma leest daarna het nummer van het weg te laten breekpunt in)
4	toon de geplaatste breekpunten (d.w.z. druk de breekpuntentabel af op het scherm)

Bevel	Betekenis
5	ga verder met de uitvoering (na een breekpunt)
6	druk de inhoud van de rekenregisters af

Bij opdracht 5 (verdergaan met de uitvoering) moet je ervoor zorgen dat eerst de instructie die in de breekpuntentabel staat uitgevoerd wordt. Bedenk zelf een manier om dit te doen.

Aangezien het speurprogramma het uit te testen programma in het geheugen moet laden, zal je speurprogramma ook een (absolute) lader moeten bevatten. Wil je dit speurprogramma uitproberen, dan zal je een gelijkaardige '***ingreep***' moeten doen zoals we dat bij de lader hadden gedaan.



Dit geheel wordt vertaald en uitgevoerd op de DRAMA-machine. Door het extra sprongbevel (als eerste instructie), wordt eerst het uit te testen programma in het geheugen geladen en daarna begonnen met de uitvoering van het speurprogramma. Deze moet dan een bevel (een getal van `1` tot `6`) inlezen en de gepaste acties ondernemen.

Tenslotte nog de volgende **hint**: zorg ervoor dat bij elke overgang tussen het uit te testen programma en het speurprogramma (resp. omgekeerd) de rekenregisters weggeborgen worden (resp. hersteld worden). Hoe zit het met de conditiecode? Begrijp je nu waarom je geen breekpunt mag plaatsen op een `VSP`-instructie? Of kan je een methode vinden om dit probleem te omzeilen?
Hint: In het volgende hoofdstuk zullen we een andere wijze bekijken om de overgang tussen verschillende programma's te regelen.

6. Wat is een 'symbolisch speurprogramma'? Waarom moet de compilator weten of een dergelijk speurprogramma gebruikt zal worden?

3.7 Het opstarten van de computer

In deze paragraaf beschrijven we wat er precies gebeurt bij het opstarten van een computer, vanaf het ogenblik dat we de spanningschakelaar opzetten. Het centrale geheugen is nog leeg[1]. Bijgevolg kan de processor geen instructies uitvoeren. De enige wijze waarop we instructies van buiten de computer in de DRAMA-machine kunnen laden, is door het uitvoeren van **LEZ**-opdrachten. Maar om deze te kunnen uitvoeren, moeten ze reeds in het geheugen geladen zijn[2]! Om uit deze vicieuze cirkel te geraken, is er een ander mechanisme nodig om deze initiële **LEZ**-instructies in het geheugen te krijgen.

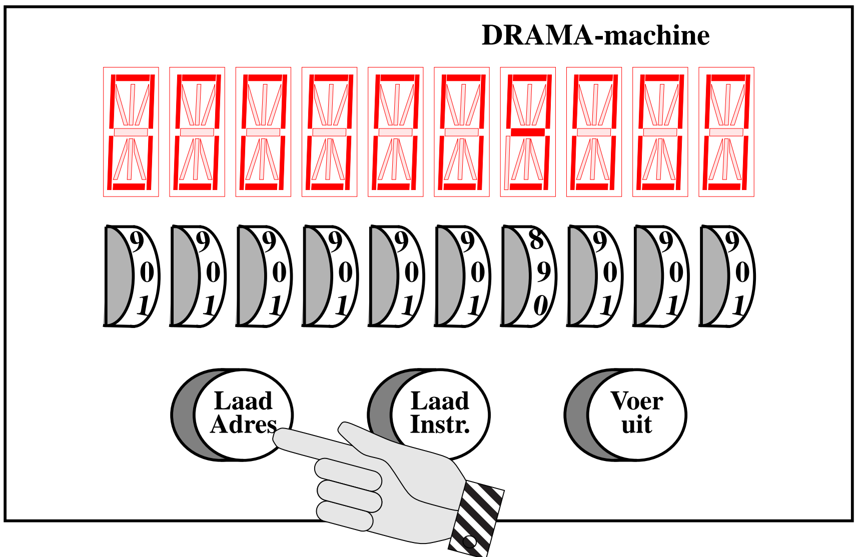
Vroeger hadden computers een ***besturingspaneel***. Hierop stonden heel wat lampjes (later LED's) en drukknoppen of schakelaars. Bij het opzetten van de spanning kwam de computer automatisch in de ***halttoestand***. In deze toestand voert de computer geen instructies uit. De programmeur (die tevens operator was) ging als volgt te werk:

1. m.b.v. de schakelaars stelde hij een bepaald adres in (bijvoorbeeld **9000**); dit adres kon in het geheugenadresregister (**GAR**) geladen worden door op de **LAADADRES**-knop te drukken,
2. vervolgens stelde hij met de schakelaars een machine-instructie in (bijvoorbeeld **7299999999**, d.i. de **LEZ**-instructie); door op de **LAADINSTRUCTIE**-knop te drukken, werd dit getal in het geheugenbufferregister (**GBR**) geplaatst, en aan het geheugen een schrijfopdracht gegeven. Hierdoor werd deze instructie in het geheugenregister met adres **9000** geplaatst. Normaal werd de inhoud van het **GAR** automatisch met éé́n verhoogd, zodat de programmeur onmiddellijk deze stap kon herhalen, tot het volledige programma in het geheugen geladen was.
3. tenslotte moest de programmeur een startadres (voor de uitvoering) instellen (bijvoorbeeld opnieuw **9000**), en op de **VOERUIT**-knop drukken. Het startadres werd in de bevelenteller geladen, en de computer kwam in de ***uitvoeringstoestand*** waarbij de normale bevelencyclus gestart wordt.

Figuur 3-38 (pag. 93) schetst het besturingspaneel van de DRAMA-machine. De programmeur heeft net het adres geladen door eerst het getal **9000** in te stellen (via de laatste vier wieltjes) en daarna de **LAADADRES-**toets in te drukken.

Het spreekt vanzelf dat het geen sinecure is om op een dergelijke manier een programma in het geheugen te laden. De kans dat een instructie of constante verkeerd wordt ingegeven is vrij groot. Bijgevolg is deze methode alleen geschikt voor heel kleine programma's, bijvoorbeeld het programma voor de absolute lader (zie figuur 3-22 (pag. 48)); de programmeur moet dit programma nog wel eerst omzetten naar machinecode — d.w.z. naar een serie decimale getallen — waarbij hij bovendien rekening moet houden met het feit dat hij de lader zelf vanaf **9000** in het geheugen zal brengen via het besturingspaneel). Met deze eenvoudige lader kunnen we een complexere lader in het geheugen brengen, bijvoorbeeld een relocerende lader, die dan op zijn beurt een nog complexer programma kan inlezen, bijvoorbeeld het besturingsprogramma.

1. Dit is onjuist. Elke geheugencel zal een of andere toevallige waarde bevatten.
2. Dit probleem is analoog aan het vraagstuk "***Wat was er eerst, de kip of het ei?***"



FIGUUR 3-38. Het besturingspaneel van de DRAMA-machine

Deze werkwijze, waarbij we starten met een heel eenvoudig programma en hiermee steeds complexere programma's gaan laden, noemt men een ***bootstrap-procedure***[1].

Bij moderne computers wordt een deel van het geheugen als leesgeheugen (ROM) uitgevoerd. Wanneer de spanning wordt opgezet, zal de processor beginnen met de uitvoering van het programma dat in dit ROM-geheugen opgeslagen is:

- dit programma test eerst de werking van de apparatuur,
- vervolgens wordt een klein programmaatje ingelezen dat op een ***vaste plaats*** (bijvoorbeeld de eerste sector) op een secundair geheugen opgeslagen is (in het Engels spreekt men over de '*bootsector*'),
- tenslotte wordt dit geladen programma uitgevoerd; het zal zelf de rest van het besturingsprogramma inlezen.

De voordelen van het gebruik van ROM zijn:

- ROM is een bestendig geheugen, dus de programmeur moet geen programma meer invoeren via het besturingspaneel,
- ROM is beveiligd, (je kan het niet per ongeluk overschrijven),
- ROM is iets vlugger dan RAM.

1. Dit komt van het Engels: "*To pull yourself up by your bootstraps*", wat wil zeggen dat je iets bereikt met heel weinig hulp.

Opgaven

1. Om een nieuw programma in te lezen (en daarna uit te voeren) in het geheugen van de computer is ... een programma nodig in het geheugen van de computer! Hoe wordt dit *"Wat was het eerst: de kip of het ei?"* probleem opgelost?
2. Wat wordt bedoeld met 'het besturingspaneel van een computer'? Hoe zag dit er vroeger uit? Vind je dit nog terug op moderne computers? Beschrijf het besturingspaneel van de DRAMA-machine.
3. Wat gebeurt er bij het opzetten van de spanningschakelaar van een computer?
4. Wat betekent de term 'bootstrap-procedure'?
5. Hoe starten moderne computers op? Is (EEP)ROM-geheugen hierbij echt noodzakelijk?

HOOFDSTUK 4
Besturingssystemen

Besturingssystemen zijn programma's die de werking van de computer ***besturen*** of ***regelen***. Deze programma's worden in het geheugen geladen en uitgevoerd bij het opstarten van de computer, en blijven normaal in het geheugen geladen tot de computer opnieuw wordt afgezet. Dankzij deze programma's is het ***eenvoudiger*** om met een computer te werken: zij nemen heel wat van de laag-niveau taken voor hun rekening, zoals het laden van een programma, het lezen van of schrijven op de harde schijf, enz. Besturingssystemen zorgen er ook voor dat de computer op een ***efficiënte*** wijze gebruikt wordt.

In dit hoofdstuk zullen we eerst het mechanisme bestuderen dat nodig is om dergelijke programma's effectief te kunnen uitvoeren, namelijk de ***programma-onderbreking***. Daarna behandelen we tamelijk uitvoerig hoe de ***randapparaten*** bestuurd worden. Tenslotte bekijken we in vogelvlucht de ***taken*** die een besturingssysteem voor zijn rekening neemt (of kan nemen). Achtereenvolgens behandelen we multiprogrammatie (of het *"gemeenschappelijk gebruik van de processor"*), geheugenbeheer (of het *"gemeenschappelijk gebruik van het geheugen"*), bestandenbeheer en informatiebeheer. De nadruk ligt op ***wat*** het besturingssysteem doet; de gebruikte algoritmes (het ***hoe***) komen aan bod in de cursus "***Besturingssystemen***".

BELANGRIJKE NOTA:
De DRAMA-machine zoals zij in het vervolg beschreven wordt, komt niet steeds overeen met de huidige simulator. Het zal dus niet altijd mogelijk zijn de voorbeelden letterlijk uit te testen op de simulator. In een latere versie van de simulator zal dit wel mogelijk zijn.

4.1 Inleiding

Besturingssystemen of besturingsprogramma's ('*operating systems*' in het Engels) hebben twee verschillende ***doelstellingen:***

1. de computer gemakkelijker toegankelijk of **gebruiksvriendelijker** maken,
2. het gebruik van de computer **efficiënter** maken.

Besturingssystemen zijn geen 'sine qua non' om met een computer te kunnen werken. In principe kan men zich een computer aanschaffen en hiermee proberen rechtstreeks (in machinecode omgezette) programma's uit te voeren. Op de DRAMA-machine worden programma's uitgevoerd zonder de aanwezigheid van een echt besturingsprogramma. In het vorige hoofdstuk hebben we echter reeds gezien dat bepaalde elementaire taken (zoals het laden van een programma) heel moeilijk zijn indien er niet reeds een programma (de lader) aanwezig is in het geheugen. Op de DRAMA-machine wordt dit probleem stilzwijgend omzeild. Een soort ***deus ex machina*** zorgt ervoor dat de programma's steeds vanaf 0000 in het geheugen geladen worden. Laders zijn op zich reeds eenvoudige besturingsprogramma's. Maar een modern besturingssysteem neemt veel meer taken op zich. Een gebruiker bewaart zijn programma's op hulpgeheugens (harde of soepele schijven, optische schijven, magneetbanden, ...). Het lezen of schrijven op deze hulpgeheugens gebeurt via zeer elementaire bewerkingen zoals 'positioneer lees/schrijf-kop op bepaalde cilinder', 'lees/schrijf een bepaalde sector'. Daarenboven verschillen deze instructies volgens het merk en type van hulpgeheugen. Stel je maar eens voor dat een gebruiker expliciet zou moeten bijhouden waar (d.w.z. in welke sectoren) zijn programma's op schijf bewaard worden! Werken met een dergelijke computer is zeker niet **gebruiksvriendelijk**. Een gebruiker is trouwens niet echt geïnteresseerd waar precies op schijf een programma wordt bewaard. Hij heeft daarentegen nood aan hoger-niveau bewerkingen zoals 'voer programma met naam XYZ uit', of 'verander het programma'. We zullen verder zien dat een besturingssysteem dergelijk hoog-niveau bewerkingen kan implementeren, en aldus de vervelende details van een computer en van zijn randapparatuur kan verborgen houden.

Een tweede bestaansreden voor besturingsprogramma's, die vooral van toepassing is op grotere computers, betreft het **efficiënt gebruik** van de computer. De eerste computers waren zeer duur, zodat de eigenaars wilden dat ze zo goed mogelijk gebruikt zouden worden, d.w.z. zo efficiënt mogelijk. Dit was dan ook de aanvankelijke drijfveer voor het ontwikkelen van besturingsprogramma's. Elke minuut die kan bespaard worden, is welkom. Bovendien worden deze computers vaak door verschillende programmeurs gebruikt, zodat ook de beschikbare ruimte op de hulpgeheugens zo billijk mogelijk moet verdeeld worden. Tenslotte willen deze gebruikers 'gelijktijdig' kunnen werken op de computer, waardoor de beschikbare 'computertijd' op een eerlijke manier verdeeld moet worden over deze gebruikers. Voor al deze taken is het besturingssysteem verantwoordelijk. In het vervolg zullen we de benamingen besturingssysteem en besturingsprogramma door elkaar gebruiken. We bedoelen hetzelfde.

Sommige besturingsprogramma's zijn voor een **bepaald type hardware** ontwikkeld (zoals DOS of OS/2 voor Intel-gebaseerde PC's, VMS voor DEC-machines, MVS voor IBM-mainframes, ...), terwijl andere dan weer op heel wat verschillende hardware 'platforms' beschikbaar zijn. Een voorbeeld hiervan is UNIX (of een van de vele -IX varianten) die aanvankelijk in de academische wereld erg populair was, maar vandaag de 'werkstation'-markt lijkt te veroveren.

Er bestaan grote verschillen tussen al deze besturingssystemen. Dit is ook niet verwonderlijk daar gebruikers en beheerders van computers verschillende wensen hebben. Een PC-gebruiker heeft andere verzuchtingen dan een beheerder van een groot computercentrum in een bedrijf. Bovendien is de hardware erg verschillend: de eerste DOS-versie moest kunnen uitgevoerd worden op een PC die slechts enkele honderden kilobytes RAM geheugen had. Het spreekt vanzelf dat op een dergelijke computer het aantal diensten dat een besturingsprogramma kan aanbieden eerder beperkt is. Bovendien zullen deze diensten niet allemaal met de meest efficiënte algoritmes geïmplementeerd zijn.

Alhoewel er een zeer grote diversiteit bestaat, kunnen we toch gemeenschappelijke **kenmerken** herkennen. In de rest van dit hoofdstuk zullen we deze kenmerken bestuderen, en via kleine voorbeeldjes illustreren hoe een besturingsprogramma werkt. De echte algoritmes die in een besturingsprogramma gebruikt worden, zullen in de cursus ***"Besturingssystemen"*** bestudeerd worden.

Samengevat, een besturingssysteem ***regelt*** of ***controleert*** de werking van een computer. Het biedt hoog-niveau bewerkingen aan, en verbergt aldus de laag-niveau details van een computersysteem. Hierdoor wordt de computer ***gebruiksvriendelijker***. Het besturingsprogramma zorgt er ook voor dat de computer op een ***efficiënte wijze*** wordt gebruikt.

In de rest van dit hoofdstuk zullen we eerst wat aandacht besteden aan de 'hardware', want die moet aangepast worden om een modern besturingsprogramma te ondersteunen. De DRAMA-machine wordt uitgebreid met een programma-onderbrekingsmechanisme, en ook de tot nu toe eenvoudige in- en uitvoer wordt realistischer gemaakt.

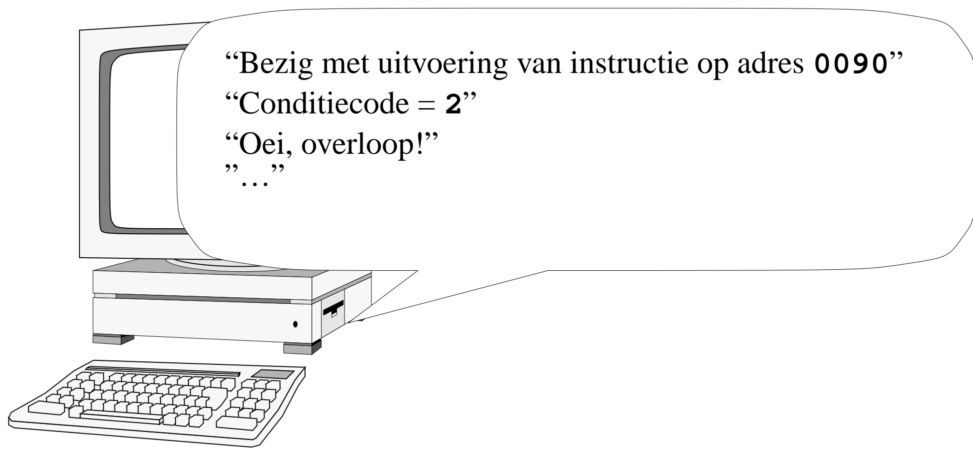
Daarna zullen we een heel belangrijk begrip, 'multiprogrammatie', introduceren. Verschillende programma's worden min-of-meer gelijktijdig uitgevoerd (d.w.z. elk om de beurt gedurende een korte tijd). Het besturingsprogramma zorgt ervoor dat dit op een 'ordentelijke' wijze gebeurt.

Tenslotte zullen we een overzicht geven van de verschillende 'types' besturingsprogramma's en eindigen we met een bondige beschrijving van de taken van een besturingssysteem.

Besturingsprogramma's of besturingssystemen zullen we in het vervolg met de afkorting BP ('*OS*' in het Engels) aanduiden.

4.2 Het programma-toestandswoord

In het vervolg van de tekst zullen we regelmatig verwijzen naar het ***programma-toestandswoord*** (PTW) (‘*program status word*’ of ‘PSW’ in het Engels). Het is een speciaal register dat deel uitmaakt van de processor. Zoals de benaming suggereert bevat het informatie over de toestand van het programma dat uitgevoerd wordt. In werkelijkheid bevat het ook informatie over de toestand van de ‘machine’ zelf. (Zie figuur 4-1.)



FIGUUR 4-1. De toestand van de DRAMA machine.

Op de DRAMA-machine bestaat het PTW uit twintig decimale cijfers (genummerd vanaf `0` tot en met `19`). We kunnen verschillende sub-velden onderscheiden, sommige hiervan zijn reeds vroeger vermeld (zie ook figuur 4-2):

- het **conditiecode**-veld (aangeduid door `CC`), dat de conditiecode bevat (`0`, `1` of `2`);
- de **bevelenteller** (`BT`), die vier cijferposities inneemt, en het adres bevat van de volgende uit te voeren instructie;
- een aantal **indicatoren** elk bestaande uit één decimaal cijfer, zoals:
  - de ***overloopindicator***, (`OVI`),
  - de ***stapeloverloop-indicator***, (`SOI`),

  indicatoren kunnen slechts twee waarden hebben: `0` of `1`, waarbij de `1` de uitzonderingstoestand is;
- de andere velden worden later besproken.

ONV	H/U	S/P	CC	OVI	SOI	◄──	────	BT	──►
0	1	2	3	4	5	6	7	8	9

G	GPF	WEK	DRK	IN	UIT	SCH	OVL	SPL	MFT
10	11	12	13	14	15	16	17	18	19

FIGUUR 4-2. Het programmatoestandswoord van de DRAMA **machine**

Om de individuele velden of een groep van velden van het PTW aan te duiden zullen we indices gebruiken. Bijvoorbeeld: PTW_5 duidt het veld met nummer 5 aan (d.i. de stapeloverloopindicator of SOI), $PTW_{6..9}$ duidt de bevelenteller (BT) aan, en $PTW_{0..9}$ is het eerste deel van het PTW.

Sommige velden van het PTW krijgen hun waarde ‘impliciet’, d.w.z. als neveneffect van de uitvoering van een instructie. Enkele voorbeelden:

- de meeste instructies (zoals **HIA**, **BIG**, **OPT**, …) zullen ervoor zorgen dat het conditiecodeveld (**CC**) een nieuwe waarde krijgt;
- wanneer het resultaat van een rekenkundige bewerking op twee getallen (bv. optelling) niet meer correct kan voorgesteld worden (met 10 decimale cijfers in de 10-complement voorstellingswijze) zal de overloopindicator (**OVI**) op ‘**1**’ geplaatst worden; in het andere geval wordt hij op ‘**0**’ gezet;
- de bevelenteller wordt automatisch opgehoogd tijdens het ophalen van een bevel, en krijgt een nieuwe waarde wanneer een sprongbevel wordt uitgevoerd.

Er bestaan ***geen*** instructies die de waarde van de individuele velden ter beschikking stellen van de programmeur. De waarde van een aantal velden kan echter wel ***indirect*** getest worden m.b.v. de **VSP**-instructie (als aan de voorwaarde voldaan is, wordt de sprong uitgevoerd; anders niet):

- de voorwaarden **NUL**, **POS**, **NEG**, **NNUL**, **NPOS**, **NNEG**, en hun alternatieve namen (**GEL**, **GR**, **KL**, …) testen de waarde van de conditiecode **CC**,
- de voorwaarden **OVL** (***‘overloop’***) en **GOVL** (***‘geen overloop’***) testen de waarde van de overloopindicator (**OVI**),
- de voorwaarden **SO** (***‘stapeloverloop’***) en **GSO** (***‘geen stapel-overloop’***) testen de waarde van de stapeloverloop-indicator (**SOI**).

Voorbeeld 4-1 illustreert het gebruik van deze voorwaarden in een DRAMA-programma.

Voorbeeld 4-1. *Het volgende programma leest in een lus telkens twee gehele getallen in, berekent de absolute waarde van hun verschil, en plaatst deze waarde op de stapel. De lus wordt beëindigd als het eerste ingelezen getal gelijk is aan nul. Aangezien het bereik van de ingelezen waarden niet gekend is, kan overloop optreden. Het is ook mogelijk dat zich een stapeloverloop voordoet. Merk op dat overloop ook kan optreden wanneer we de absolute waarde nemen van een negatief getal. (Wanneer?)*

```
            | Plaats |X-Y| op de stapel
            | waarbij X en Y eerst ingelezen worden
    LUS:    LEZ
            VSP     NUL,KLAAR
            HIA     R1,R0
            LEZ
            AFT     R1,R0
            VSP     OVL,OVERLOOP    | Overloop opgetreden?
```

```
            VSP    NNEG,POSITIEF  | Resultaat niet negatief?
            VER.w  R1,-1          | Maak positief
            VSP    OVL,OVERLOOP   | Overloop opgetreden?
POSITIEF:   BST    R1
            VSP    SO,STAPELOVL   | Stapeloverloop?
            SPR    LUS
KLAAR:      …
OVERLOOP:   …      | bij overloop …
            …
STAPELOVL:  …      | bij stapeloverloop …
            …
```

Merk op dat dit een nogal onhandige manier van werken is. De drie gekleurde instructies zijn enkel nodig omdat we niets weten over het bereik van de in te lezen **X**- en **Y**-waarden, noch over het aantal in te lezen getallen (dus stapeloverloop zou wel eens kunnen voorkomen).

Opgaven

1. Leg de volgende begrippen uit:

- programma-toestandswoord
- conditiecode
- indicatoren

2. Hoe en wanneer wordt de conditiecode geplaatst? Hoe kan je nagaan welke waarde hij heeft?

3. Welke indicatoren zijn er op de DRAMA-machine voorzien? Hoe kan je nagaan of de indicator opstaat? Hoe worden deze indicatoren weer afgezet?

4. Ontwerp een programma dat in een lus telkens twee getallen inleest, hun product berekent en op een nieuwe lijn afdrukt. Het programma moet stoppen indien twee keer `0` ingelezen wordt. Indien er overloop optreedt, wordt als resultaat twee keer (op dezelfde lijn) nul afgedrukt. Test je programma uit met de volgende invoer:

10	500000
1000	100000
500000	500000
100	-20000
500000	-500000
1000	-2000
500000	0
10000	0
0	0

4.3 Programma-onderbrekingen

Programma-onderbrekingen zijn erg belangrijk zowel om efficiëntie-redenen als om de correcte werking van programma's en computer te garanderen.

4.3.1 Inleiding

In deze paragraaf worden heel bondig een tweetal situaties geschetst die de nood aan een onderbrekingsmechanisme aantonen.

Overloop

Veronderstel dat een programma heel veel berekeningen moet uitvoeren. Indien het bereik van de getallen waarmee gerekend wordt niet op voorhand gekend is, is het mogelijk dat op een bepaald ogenblik ***'overloop'*** optreedt (d.w.z. dat het resultaat te groot is om nog correct voorgesteld te kunnen worden). Natuurlijk is het niet zinvol om dan nog de berekeningen verder te zetten, daar het resultaat foutief zal zijn.
Het programma zou deze toestand kunnen detecteren door na te gaan of de overloop-indicator opstaat (via de **VSP**-instructie):

```
           HIA  R1,A
           OPT  R1,B
           VSP  OVL,OVERLOOP
           …
OVERLOOP:  …
```

Om zeker te spelen zou het programma na **elke** rekenkundige bewerking deze test moeten uitvoeren. Immers, de overloop-indicator wordt bij de volgende rekenkundige bewerking terug op '**0**' gezet indien deze opdracht zonder overloop kan uitgevoerd worden. Deze testen betekenen een serieuze vertraging in de uitvoering van het programma. Het programma zou veel sneller uitgevoerd worden indien we de **VSP** achterwege konden laten, op voorwaarde dat de uitvoering ***automatisch onderbroken*** wordt ***door de apparatuur*** wanneer overloop optreedt. Zie figuur 4-3.

Bovendien zou het programma minder geheugen in beslag nemen. Een onderbrekingsmechanisme zou dus zowel op uitvoeringstijd als op geheugen besparen. Merk op dat de elektronica voor het detecteren van overloop reeds aanwezig is op de DRAMA-machine. Immers, de overloop-indicator wordt dan opgezet. Wat we nu nog extra willen, is een mechanisme dat de uitvoering van een programma onderbreekt wanneer overloop zich voordoet.

FIGUUR 4-3. Automatische onderbreking bij een uitzonderingstoestand.

Oneindige lussen

Op een computersysteem moeten vaak meerdere programma's uitgevoerd worden. Naast het gebruikersprogramma is er minstens ook een besturingsprogramma (BP) aanwezig. Een BP wordt bij het opstarten van de computer in het geheugen geladen (zie ook par. 3.7), en zal normaal in het geheugen blijven zolang de computer niet opnieuw wordt afgezet. Eén van de taken van een BP kennen we reeds: het laden van programma's in het geheugen. Andere taken omvatten het verzorgen van in- en uitvoer, het beheren van bestanden, enz. Deze taken kunnen slechts uitgevoerd worden indien de processor de instructies van die taak van het BP uitvoert.

Op elk ogenblik kan de CVE slechts bevelen van één programma uitvoeren. Het afwisselen tussen de verschillende programma's zou kunnen gebeuren met behulp van **`SPR`**-instructies. Dit is niet echt werkbaar omdat het BP vaak op willekeurige of 'asynchrone' (d.w.z. niet op voorhand gekende) ogenblikken moet uitgevoerd worden. Later zullen we hiervoor een aantal voorbeelden geven.

De apparatuur moet dus zodanig zijn dat dergelijke 'programma-wisselingen' op gelijk welk ogenblik kunnen doorgevoerd worden. Bij de eerste computergeneraties bestond deze mogelijkheid niet. De apparatuur moest hiervoor speciaal aangepast worden. Ook de DRAMA-computer werd hiervoor uitgebreid.

We zeggen dat een ***programma controle heeft over de processor*** indien de CVE bezig is met de uitvoering van instructies van dit programma. Zonder speciale voorzieningen kan een programma dat op dit ogenblik controle heeft over de CVE, deze CVE monopoliseren, d.w.z. dat de processor steeds instructies van dit programma blijft uitvoeren. Een eenvoudig voorbeeld is een **oneindige lus**, d.w.z. een lus die nooit eindigt (zie figuur 4-4). Dit is een regelmatig voorkomende programmeer-fout. Een lus wordt uitgevoerd zolang aan een bepaalde voorwaarde (wel of niet) voldaan is, terwijl in de lus zelf de voorwaarde nooit gewijzigd wordt. In de figuur staat links een foutief C-programma (zolang de inhoud van '**`a`**' kleiner is dan de inhoud van '**`b`**' wordt bij '**`a`**' de inhoud van '**`c`**' opgeteld) en rechts het naar DRAMA omgezette programma;

aangezien de inhoud van ‘**c**’ nul is zal de inhoud van ‘**a**’ ook niet wijzigen en blijft de voorwaarde in de **while**-lus altijd waar. De processor zal bijgevolg de instructies van deze lus een oneindig aantal keer blijven uitvoeren. Zonder een ‘programma-onderbrekingsmechanisme’ kunnen we hier alleen maar uit geraken door de computer af te zetten.

C programma (oneindige lus)	Lagere Programmeertaal (oneindige lus)
...	...
c = 0;	*HIA.w R3,0* | c = 0
a = 0;	HIA.w R1,0 | a = 0
b = 17;	HIA.w R2,17 | b = 17
while (a < b)	LUS: VGL R1,R2
	VSP GRG,EINDE
a = a + c;	OPT R1,R3 | a = a + c
	SPR LUS
	EINDE: ...

FIGUUR 4-4. Een oneindige lus in C en DRAMA.

In de rest van deze paragraaf zullen we eerst in het kort de bevelencyclus herhalen en ze daarna aanpassen voor programma-onderbrekingen.

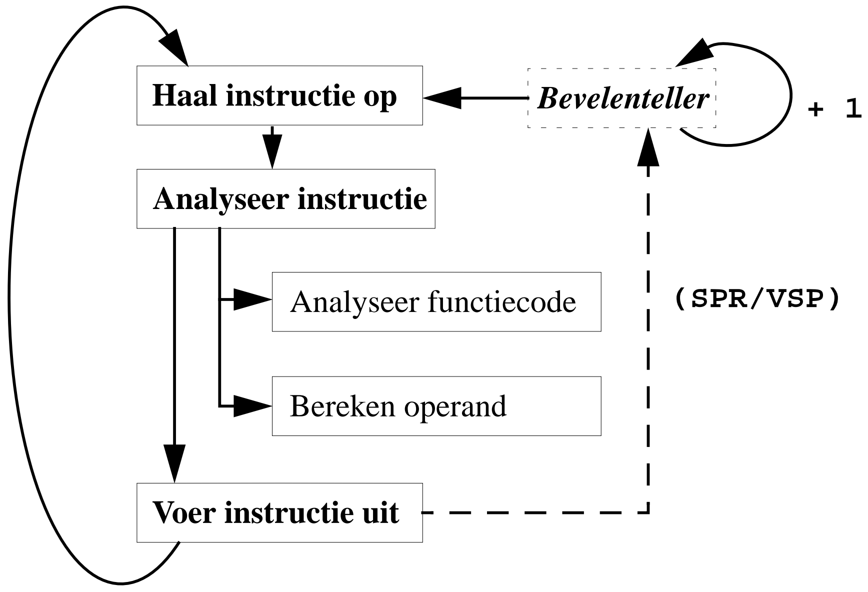
4.3.2 De bevelencyclus

Tijdens de normale werking van een computer, worden voortdurend instructies uitgevoerd. Deze uitvoering kan opgesplitst worden in een aantal deelstappen, die gesuperviseerd worden door het besturingsorgaan van de processor (zie ook figuur 4-5 (pag. 104)):

1. de volgende instructie wordt ***opgehaald*** uit het geheugenregister waarvan het adres gevonden wordt in de bevelenteller (BT, eigenlijk $PTW_{6..9}$);

   tijdens het ophalen wordt de bevelenteller met één verhoogd zodat hij reeds de juiste waarde heeft voor de volgende bevelencyclus,
2. de instructie wordt ***geanalyseerd***;
   - de functiecode wordt onderzocht,
   - de operanden worden berekend (en eventueel uit het geheugen opgehaald),
3. de instructie wordt ***uitgevoerd***

   (eventueel wordt een waarde in het geheugen weggeborgen).



FIGUUR 4-5. Blokschema van de bevelencyclus.

Aangezien de bevelenteller in de eerste stap verhoogd wordt, zullen normaal tijdens de opeenvolgende bevelencycli de opeenvolgende instructies van een programma uitgevoerd worden. Men spreekt soms over een computer met ***volgorde*** of ***sequentiële besturing***.

Er zijn slechts twee manieren om deze ***sequentiële opeenvolging te doorbreken***:

- de eerste manier is het gebruik van sprongbevelen (zie stippellijn in figuur 4-5);
- de tweede manier is het mechanisme van programma-onderbrekingen (zie verder).

Wanneer een **sprongbevel** in stap 3 wordt uitgevoerd zal de berekende operand in de bevelenteller worden geplaatst. Hierdoor gaat de vorige inhoud (namelijk het adres van de instructie die volgt op het sprongbevel) verloren, en wordt de sequentiële opeenvolging doorbroken. In figuur 4-6 (pag. 105) wordt dit geïllustreerd. Bovenaan wordt de inhoud van het geheugen afgebeeld, onderaan worden de opeenvolgende stappen van de bevelcycli en de inhouden van de beveltteller (BT) en bevelenregister (BR) getoond. De bevelen in geheugenregisters met adressen **`0100`**, **`0101`**, **`0103`**, …worden achtereenvolgens uitgevoerd. (**`0102`** wordt overgeslagen door de uitvoering van het **`SPR`**-bevel, zie het handje (☞) in de figuur.)

Bij voorwaardelijke sprongbevelen (**`VSP`**) zal de bevelenteller alleen overschreven worden indien aan de voorwaarde die in de opdracht gespecificeerd is, voldaan is. In het andere geval heeft de uitvoering geen effect, en zal tijdens de volgende bevelencyclus de instructie die volgt op de **`VSP`**-opdracht opgehaald, geanalyseerd en uitgevoerd worden.

	Symbolische inhoud	
	...	
	BIG	R3,RES
	SPR	VERDER
RES:	RESGR	1
VERDER:	HIA.w	R2,0
	...	

	Geheugen
	...
0100:	1221300102
0101:	3221900103
0102:	0000000000
0103:	1111200000
	...

fase	**bevelenteller**	**bevelenregister**
	...	
	0100	
ophalen		1221300102
	0101	
analyse		
uitvoeren		
ophalen		3221900103
	0102	
analyse		
uitvoeren	☞ 0103	
ophalen		1111200000
	0104	
	...	

FIGUUR 4-6. De uitvoering van een sprongbevel.

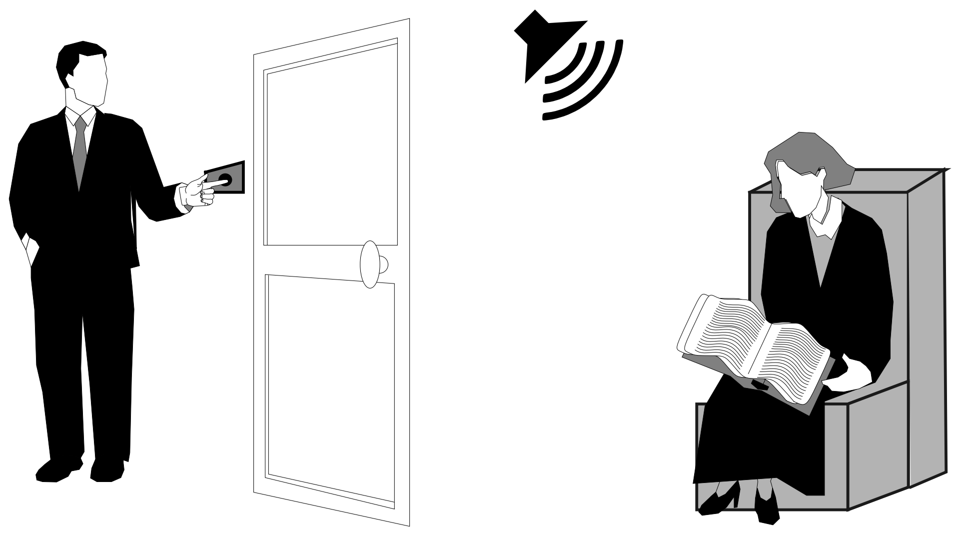
4.3.3 Het programma-onderbrekingsmechanisme

Indien het besturingsprogramma op een willekeurig ogenblik de controle over de processor wil krijgen, of indien zich een fout voordoet (bijvoorbeeld 'overloop'), moet er een mechanisme bestaan dat de normale 'volgorde besturing' kan doorbreken. Dit mechanisme wordt de ***programma-onderbreking*** genoemd, omdat het in staat is de uitvoering van een programma te onderbreken en de uitvoering van een ander programma (i.c. een routine van het BP) te beginnen of verder te zetten.

Ook wij mensen worden in ons dagelijks bestaan met onderbrekingen geconfronteerd. Veronderstel dat je een boek aan het lezen bent (dit is een sequentiële bezigheid; gewoonlijk lees je een boek zin na zin, bladzijde na bladzijde, van voor naar achter). Op een bepaald ogenblik kan

de bel van de voordeur of van de telefoon rinkelen. Het gerinkel geeft aan dat iemand anders je aandacht nodig heeft, en dat je best je huidige bezigheden onderbreekt. Normaal gebruik je een ***bladwijzer*** om aan te geven waar je reeds in het boek gekomen bent en je begeeft je naar de voordeur respectievelijk de telefoon. Nadat je klaar bent met persoon die je aandacht vroeg, neem je terug je boek, en ga je verder lezen vanaf de plaats waar je reeds gekomen was (d.w.z. waar de bladwijzer zich bevindt).

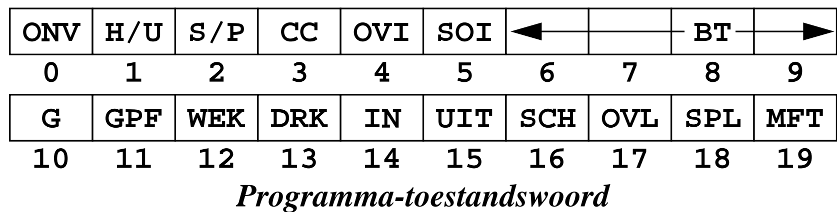
Soms wil je helemaal niet gestoord worden in je huidige bezigheden. In dit geval kan je de bel laten rinkelen en het gerinkel negeren ("Laat maar bellen, ik ben bezig!").

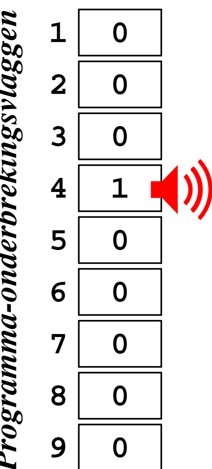


FIGUUR 4-7. Een onderbrekingsmechanisme (Analogie).

Ga maar eens na wat je zou moeten doen indien er geen bel aanwezig is. Indien je bezoek 'verwacht' zou je op regelmatige tijdstippen naar de deur moeten lopen om te kijken of je bezoeker nog niet gearriveerd is (je wil hem/haar toch niet te lang laten wachten). Maar hoe kan je buurman je verwittigen dat het dak van je huis in brand staat?

Een gelijkaardig onderbrekingsmechanisme is op elk modern computersysteem voorzien. De bellen worden voorgesteld door **programma-onderbrekingsvlaggen** (PO-vlaggen). Op de DRAMA-machine zijn er negen, genummerd van `1` tot en met `9`; dus kan de machine negen verschillende soorten onderbrekingen onderscheiden. Deze PO-vlaggen zijn één decimaal cijfer lang en maken deel uit van de CVE. (Zie ook figuur 4-8.) Een `0` betekent dat er geen onderbreking van die soort wordt aangevraagd (de bel gaat niet); een `1` betekent het tegengestelde (de bel gaat). (Op een reële computer volstaat natuurlijk één bit per vlag). De PO-vlaggen kunnen, zoals we later zullen zien, geplaatst worden op vraag van de hardware (die iets abnormaals opmerkt), een randapparaat (dat de aandacht vraagt) of op vraag van het programma dat uitgevoerd wordt (en hulp nodig heeft van het besturingsprogramma). In figuur 4-8 zie je dat er op dit ogenblik een onderbreking aangevraagd is: namelijk PO_4.





Processor

FIGUUR 4-8. Het programma-toestandswoord en de onderbrekingsvlaggen.

Zoals voor ons het gerinkel van de telefoon soms **ongelegen** komt, kan ook een aanvraag tot een PO op een ongelegen moment komen voor de processor: hij is bijvoorbeeld bezig met het invullen van een belangrijke tabel of heeft een dringender taak te vervullen. Daarom zal een PO niet altijd kunnen optreden. Dit wordt echter in een volgende paragraaf besproken. Wanneer een onderbreking wordt aangevraagd, en de processor wil hierop ingaan, dan zal de CVE stoppen met de uitvoering van het huidige programma en verder gaan met de uitvoering van een onderdeel van het besturingsprogramma.

Aanvragen tot een programma-onderbreking kunnen op elk ogenblik gesteld worden, ook tijdens het ophalen van een instructie, tijdens de analyse of tijdens de uitvoering. Om alles toch **ordelijk** te laten lopen, zal de processor alleen kunnen ingaan op een PO-aanvraag juist nà de uitvoering van een instructie[1].

De **bevelencyclus** wordt dus met een vierde stap uitgebreid: het besturingsorgaan zal nagaan of zich een aanvraag voor een PO heeft voorgedaan (m.a.w. of een PO-vlag opstaat).

De volledige bevelencyclus wordt dus:

1. haal de volgende instructie op,
   en verhoog de bevelenteller;
2. analyseer de instructie:
   - analyseer de functiecode en
   - bereken de operand(en);
3. voer de instructie uit;

1. Op reële computers is dit ook zo. Er wordt alleen een uitzondering gemaakt voor instructies waarvan de uitvoering zeer veel tijd vergt, zoals het kopiëren van een groot gedeelte van het geheugen naar een andere plaats in dit geheugen.

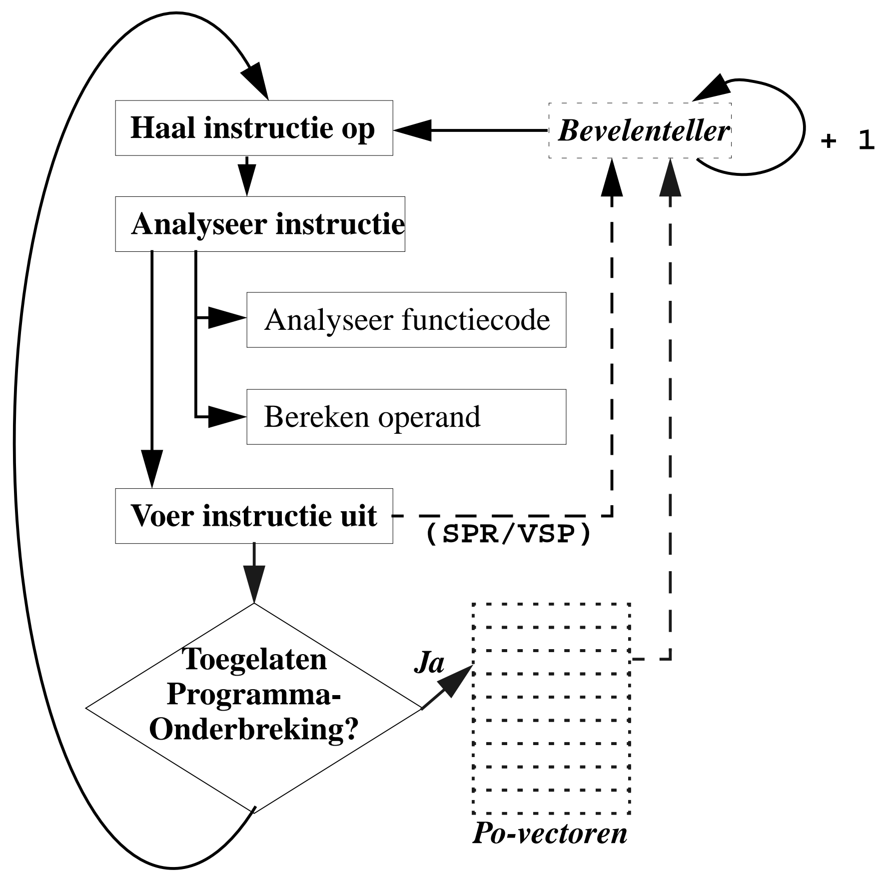
4. indien een PO-vlag opstaat en de CVE is bereid hierop in te gaan (m.a.w. de PO is toegelaten):

a. onthoud waar de processor gekomen is
d.w.z. bewaar de inhoud van de **BT**, … ***(vgl.: gebruik een bladwijzer)***

b. zet de PO-vlag af ***(vgl.: neem de hoorn op)***

c. begin de uitvoering van een ander programma
d.w.z. geef de BT een nieuwe waarde ***(vgl.: spreek met de oproeper)***

Indien de processor wil ingaan op een aanvraag tot PO, hoe bepaalt hij dan **welk onderdeel** van het BP hij zal beginnen uitvoeren (m.a.w. welk nieuw adres zal er in de BT geplaatst worden)? Op oudere machines was dit een vaste waarde (bijvoorbeeld **9000**). Op modernere (zoals DRAMA) worden deze adressen bijgehouden in een tabel: de **programma-onderbrekingsvectoren** (PO-vectoren), die dikwijls in het geheugen worden bijgehouden. Op de DRAMA-machine bevinden de PO-vectoren zich in geheugenregisters met adressen **9991** tot en met **9999**. Elke vector bevat het adres van het programmaonderdeel dat moet uitgevoerd worden indien de processor ingaat op een PO-aanvraag van die soort. **9991** bevat het adres voor PO_1, **9992** voor PO_2, enz. (Merk op dat alleen de laatste vier cijfers van deze geheugenregisters gebruikt zullen worden.) Het algoritme wordt ook schematisch in figuur 4-9 voorgesteld. De stippellijnen duiden aan dat via die weg de bevelenteller een andere waarde kan krijgen (dus door de uitvoering van een sprongbevel of door het optreden van een programma-onderbreking).



FIGUUR 4-9. Blokschema van de bevelencyclus met een programmaonderbrekingsmechanisme

Alvorens een nieuwe waarde in de BT wordt geplaatst, wordt de **oude waarde** weggeborgen op een veilige plaats, zodat de processor later verder kan gaan met de uitvoering van het onderbroken programma.

Op oudere machines werd hiervoor een vast geheugenregister gebruikt (bijvoorbeeld **9990**). Dit levert echter problemen op indien tijdens de afhandeling van een PO een andere PO optreedt: het bewaarde adres wordt dan terug overschreven en is onherroepelijk verloren gegaan.

Tegenwoordig gebruikt men een flexibeler methode, die ook bij DRAMA toegepast wordt: de vorige waarde van de bevelenteller wordt op de top van de stapel geplaatst. Bovendien zal niet alleen de waarde van de BT maar de volledige eerste 10 cijfers van het programma-toestandswoord ($PTW_{0..9}$) op de stapel worden gezet. We zullen verder zien dat het inderdaad noodzakelijk is om ook de oorspronkelijke waarde van de conditiecode en de indicatoren, … te bewaren. Merk op dat deze aanpak niet geheel zonder **risico** is: er kan stapeloverloop optreden (d.w.z. de op de stapel geplaatste waarde ($PTW_{0..9}$) overschrijft een gegeven of een instructie van het programma). We komen hierop terug in de opgaven na deze paragraaf.

In figuur 4-10 wordt het vorige voorbeeld uit figuur 4-6 (pag. 105) nog eens hernomen. Alleen komt er nu tijdens het ophalen van de **SPR**-instructie een aanvraag tot PO_4 voor (zie het grote gekleurde handje ( ) in de figuur).

Symbolische inhoud

```
              BIG      R3,RES
              SPR      VERDER
      RES:    RESGR    1
   VERDER:    HIA.w    R2,0
              ...
```

```
| Het besturingssysteem
              ...
PO_ROUTINE4:  BIG      R0,BEWAAR
              ...
   BEWAAR:    RESGR    10

              ...
```

Geheugen

Adres	Inhoud
	...
0100:	1221300102
0101:	3221900103
0102:	0000000000
0103:	1111200000
	...
9050:	1221009990
	...
	...
	...
9990:	...
	...
9994:	0000009050
	...

FIGUUR 4-10. Het effect van een programma-onderbreking op de bevelenteller.

fase	bevelenteller	bevelenregister
	... `0100`	
ophalen	`0101`	`1221300102`
analyse		
uitvoeren		
PO?		
ophalen	`0102`	`3221900103`
analyse		
uitvoeren	`0103`	
PO !!!	`9050`	
ophalen	`9051`	`1221009990`
	...	

bewaar **`xxxxxx0103`** op stapel
zoek de nieuwe waarde voor de BT op adres **`9994`** = **`9050`**

Uitvoering van **`PO_ROUTINE4`**

De uitvoering van het **`SPR`**-bevel heeft als resultaat dat de BT een nieuwe waarde krijgt (namelijk **`0103`**). Nog voor deze nieuwe waarde gebruikt kan worden, ondergaat de processor een PO (zie de licht gekleurde vakjes in de figuur): de waarde **`xxxxxx0103`**[1] wordt op de stapel geplaatst, in geheugenregister **`9994`** wordt de nieuwe waarde van de BT opgezocht: **`9050`**, die tenslotte in de BT wordt geplaatst (zie het kleine witte handje ( )). Hierdoor begint de processor de uitvoering van het programma **`PO_ROUTINE4`**. Merk op dat de processor steeds in staat zal zijn om (na het afwerken van de programma-onderbreking) de uitvoering van het onderbroken programma verder te zetten vermits de oude waarde van de BT op de stapel werd bewaard.

4.3.4 Soorten onderbrekingen

Er zijn tal van oorzaken die aanleiding kunnen geven tot een programma-onderbreking. We zullen hier een korte opsomming geven; later worden sommige meer uitgebreid besproken. De programma-onderbrekingen kunnen onderverdeeld worden in vijf verschillende klassen:

1. De **`x`**-en staan voor de cijferwaarden van $\text{PTW}_{0..5}$.

1. **externe oorzaken**: deze zijn bijvoorbeeld afkomstig van een externe klok of van allerlei wekkers die aflopen; het kan ook veroorzaakt worden door het indrukken van de **`reset`**-toets[1].
2. **randapparatuur**: de besturingseenheden (bestuurders) van randapparaten (in het Engels '*controllers*') of de kanaalbestuurders (in het Engels '*channels*') kunnen de aandacht vragen van de processor[2]; ***randapparatuur*** is een vrij algemene term en omvat o.a. ***invoerapparaten***, zoals toetsenborden, ***uitvoerapparaten***, zoals schermen of drukkers, ***hulpgeheugens***, zoals schijven- of magneetbandeenheden, alsook ***netwerkinterfaces***, …
3. **machinefout**: de meest ernstige "machinefout" is het uitvallen van de elektrische spanning: het besturingsprogramma zal proberen te redden wat er nog te redden valt (bijvoorbeeld de inhoud van het geheugen wegschrijven op schijf; dit natuurlijk in de veronderstelling dat een of andere batterij tijdelijk de nodige spanning levert).
   Elke computer bevat ook een controlesysteem dat fouten bij het gegevenstransport opspoort. Dit controlesysteem zal zoveel mogelijk de opgespoorde fouten verbeteren. Wanneer dit onmogelijk is zal de apparatuur het probleem signaleren via een programma-onderbrekingsaanvraag.
4. **programmafout**: tijdens het uitvoeren van een programma kan 'overloop' optreden of kan 'een deling door nul' uitgevoerd worden; het is ook mogelijk dat de processor een niet bestaande instructie moet uitvoeren (dit kan zich voordoen indien het programma een sprong uitvoert naar een geheugenregister waarin gegevens bewaard worden); de uit te voeren instructie kan verwijzen naar een ongeldig (of onbestaand) geheugenadres; of de metabits[3] of het datatype-veld van de operanden kloppen niet; …

   Sommige van deze fouten worden niet alleen door een programma-onderbreking, maar ook via een indicator aangegeven (zie bijvoorbeeld **`OVI`** en **`SOI`** in het PTW);
5. **geprogrammeerde PO**: soms wil het programma onderbroken worden op een bepaalde plaats (in principe zou dit m.b.v. een **`SBR`**-instructie kunnen gerealiseerd worden; we zullen echter zien dat er redenen zijn om dit via een speciale instructie te doen, die een programma-onderbreking tot gevolg heeft (cfr. paragraaf 4.3.7 (op pag. 121))).

In het Engels worden onderbrekingen, die veroorzaakt worden door het programma zelf (d.w.z. programma-fouten of een geprogrammeerde onderbreking) '*traps*' of '*exceptions*' genoemd; de overige worden aangeduid met de algemene term '*interrupts*'. Het verschil tussen beide ligt in het feit dat de eerste categorie ***'synchroon' met de uitvoering van het programma*** optreedt, terwijl de tweede soort zich ***'asynchroon'*** aandient.

1. Wanneer het systeem helemaal vast lijkt te zitten, en er geen respons meer komt, zal de gebruiker soms geen andere keuze hebben dan het systeem te herstarten. In plaats van de spanning af en terug op te zetten, zal men bij voorkeur de **`reset`**-toets indrukken.
2. Bijvoorbeeld, wanneer de gebruiker een toets indrukt op het toetsenbord, zal de bestuurder van het toetsenbord proberen de processor hierop attent te maken d.m.v. een PO-aanvraag.
3. Sommige computers voegen extra bits toe aan de gegevens die opgeslagen of verwerkt worden. Deze bits duiden aan wat het type is van deze gegevens: een natuurlijk of geheel getal, een rij symbolen, een bewegende komma getal, ... Hierdoor kunnen een aantal fouten vermeden worden, zoals het rekenen met een rij van symbolen, het optellen van een bewegende kumma getal met een geheel getal, enz. De meeste computers gebruiken echter geen metabits.

Tabel 4-1 geeft voor elk onderbrekingsnummer de prioriteit, het adres van de overeenkomstige PO-vector, de oorzaak voor de PO-aanvraag, het bijbehorende ***masker*** en het type (synchroon of asynchroon) weer. De prioriteit en het masker worden in de volgende paragraaf besproken.

TABEL 4-1. Programma-onderbrekingen in DRAMA.

Nummer PO = Prioriteit	PO-vector	Oorzaak	Masker	Type
9	**9999**	machinefout ***of*** ongeldige instructie ***of*** ongeoorloofde instructie	**MFT**	asynch. synch. synch.
8	**9998**	programmafout (stapeloverloop en -onderloop)	**SPL**	synch
7	**9997**	programmafout (overloop)	**OVL**	synch.
6	**9996**	schijfbestuurder	**SCH**	asynch.
5	**9995**	schermbestuurder	**UIT**	asynch.
4	**9994**	toetsenbordbestuurder	**IN**	asynch.
3	**9993**	drukkerbestuurder	**DRK**	asynch.
2	**9992**	wekker	**WEK**	asynch.
1	**9991**	supervisie-oproep (via **OND**-bevel)	—	synch.

Merk op dat voor overloop (resp. stapeloverloop) zowel een indicator (**OVI** resp. **SOI**) als een onderbrekingsvlag (PO_7 resp. PO_8) voorzien is. De indicatoren zijn eerder bedoeld om getest te worden tijdens de uitvoering van een programma, bijvoorbeeld wanneer het onderbrekingsmechanisme uitgeschakeld is (zie verder). Bovendien geven de indicatoren slechts heel kortstondig een bepaalde uitzonderingstoestand aan: ze krijgen immers een nieuwe waarde na elke rekenkundige en transport-bewerking.

4.3.5 Verbieden van programma-onderbrekingen

Soms is het nodig bepaalde onderbrekingen te verbieden. De onderbreking komt ongelegen, of er zijn eerst dringender taken uit te voeren.

'Verbieden' is een verzamelwoord met twee betekenissen:

- opschorten of uitstellen,
- negeren.

In DRAMA kozen we, behalve in één uitzonderingsgeval, steeds voor 'uitstellen'.

Op de DRAMA-machine voorziet het onderbrekingsmechanisme twee verschillende methodes om dit te regelen:

a. maskers en

b. prioriteiten.

Maskers

Jijzelf kan de telefoon of de bel laten rinkelen, of nog drastischer, je kan de hoorn van de haak leggen, zodat niemand je nog kan opbellen.

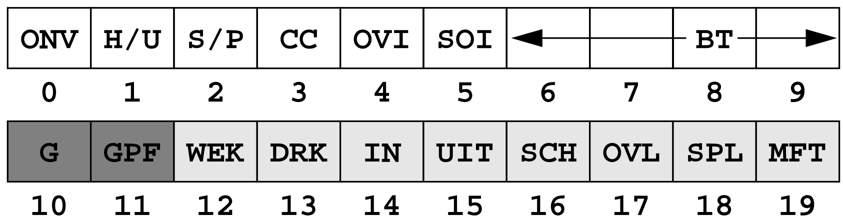
FIGUUR 4-11. Het verbieden of uitstellen van onderbrekingen.

Indien je niet wenst gestoord te worden in je bezigheden (je leest een spannend boek en wil absoluut de ontknoping kennen), kan je twee dingen doen:

- ofwel maak je het onmogelijk dat men je onderbreekt (je legt bijvoorbeeld de hoorn van de haak: niemand zal nog in staat zijn je op te bellen);

- ofwel schort je de onderbreking tijdelijk op (je zet bijvoorbeeld een hoofdtelefoon op en beluistert een CD: de bel kan nog rinkelen, maar aangezien je ze tijdelijk niet hoort, ga je er niet op in).

Iets gelijkaardigs is ook nodig voor de processor. Wanneer hij bezig is met het wijzigen van een belangrijke tabel of wanneer hij een dringende taak moet vervullen, wordt hij beter niet onderbroken tot deze klus geklaard is. Op de DRAMA-machine voorzien we voor elk soort PO[1] een PO-***masker*** ('*mask*' in het Engels). Deze maskers behoren tot het programma-toestandswoord (PTW); op de DRAMA-machine nemen zij de tweede helft van dit woord in beslag. We zouden dit het ***maskerwoord*** kunnen noemen. Elk masker neemt één decimale cijferpositie in, die alleen de cijfers **0** en **1** kan bevatten[2] (zie figuur 4-12: het maskerwoord komt overeen met $\text{PTW}_{10..19}$).



FIGUUR 4-12. Het maskerwoord in het DRAMA-PTW.

Indien de waarde '**0**' is, is de processor bereid om in te gaan op de overeenkomstige PO-aanvraag; een PO van dit type is dus '**toegelaten**' ('*enabled*' in het Engels) Wanneer echter het masker de waarde '**1**' heeft, wordt elke aanvraag voor een PO van dit type '**geweigerd**' ('*disabled*' in het Engels).

Merk op dat wanneer een bepaald masker opstaat, het nog steeds mogelijk is dat de overeenkomstige PO-vlag geplaatst wordt. Alleen zal de processor voorlopig niet meer ingaan op deze aanvraag tot een programma-onderbreking. De programma-onderbreking wordt dus eerder **uitgesteld**, maar de aanvraag blijft **hangende** ('*pending*' in het Engels). ***(De telefoon rinkelt wel, maar je neemt voorlopig niet op, omdat je het gerinkel niet hoort!)*** Zodra het masker opnieuw wordt afgezet en er zijn geen dringender taken (cfr. verder), zal de programma-onderbreking optreden.

Als alternatief zou een opstaand masker kunnen verhinderen dat de overeenkomstige PO-vlag geplaatst wordt. Men zegt dan dat de aanvraag tot PO **'verboden'** (of '**genegeerd**') wordt ('*ignored*' in het Engels). ***(Je legt de hoorn van de haak!)***

Op de DRAMA-machine hebben we voor de eerste variante gekozen: de gemaskeerde PO's blijven dus hangende, behalve in één geval: indien **GPF** opstaat worden bepaalde PO's genegeerd (zie verder).

1. Er is één uitzondering, voor programma-onderbreking PO_1 is geen masker beschikbaar. Later zal duidelijk worden waarom.
2. Op een reële (binaire) computer volstaat hiervoor één bit.

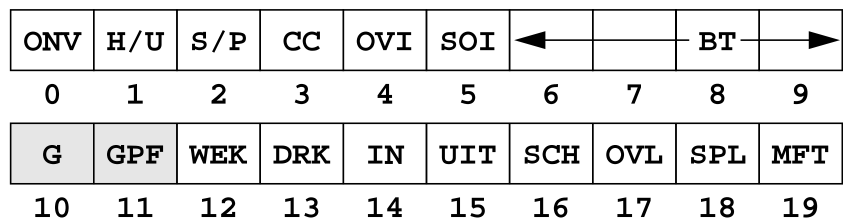
Op de DRAMA-machine zijn vier bevelen beschikbaar voor het plaatsen, verwijderen of testen van maskers en voor het testen van een PO-vlag. Zie hiervoor tabel 4-2.

TABEL 4-2. Instructies voor het plaatsen/testen van maskers en testen van vlaggen.

DRAMA-instructie	Verklaring
`MKH x`	zet het masker voor PO$_x$ op één (d.w.z. de processor zal voorlopig onderbrekingen van deze soort uitstellen). ‘`x`’ wordt symbolisch aangeduid door de naam die in de kolom ‘Masker’ van tabel 4-1 (pag. 112) staat. `MKH` is de afkorting van “**m**as**k**er **h**oog”.
`MKL x`	zet het masker voor PO$_x$ op nul. `MKL` betekent “**m**as**k**er **l**aag”.
`TSM x`	test de waarde van het masker voor ‘`x`’. De conditiecode wordt geplaatst: nul indien het masker afstaat en één (d.i. positief) indien het masker opstaat. `TSM` staat voor “**t**e**s**t **m**asker”.
`TSO x`	test de waarde van de onderbrekingsvlag voor PO$_x$. Ook hier wordt de conditiecode geplaatst. Merk op dat zelfs wanneer onderbrekingen niet toegestaan zijn, een programma toch kan nagaan of zo’n aanvraag tot onderbreking ‘hangende’ is (m.a.w. of zo’n aanvraag gebeurd is). `TSO` betekent “**t**e**s**t **o**nderbrekingsvlag”.

Globale Maskers

Er zijn twee speciale maskers voorzien, (zie ook figuur 4-13); ze worden ook wel **globale maskers** genoemd, want hun invloed is groter dan bij de specifieke maskers.



FIGUUR 4-13. Globale maskers in het DRAMA-maskerwoord.

1. De eerste cijferpositie van het maskerwoord, namelijk masker `G`, bevat het ‘**g**lobale’ masker. Indien dit cijfer `1` is wordt ***geen enkele*** programma-onderbreking toegestaan. Het is dus heel eenvoudig om ineens alle PO’s uit te stellen: namelijk via `MKH G`.
2. De tweede cijferpositie (`GPF`) is het ‘**g**lobaal **p**rogramma-**f**out’ masker, dat onderbrekingen ten gevolge van twee soorten programmafouten (overloop en stapeloverloop) kan verhinderen. Wanneer dit masker opstaat worden aanvragen voor PO$_7$ (`OVL`) en PO$_8$ (`SPL`) **gene-**

geerd; m.a.w. indien **GPF** opstaat, kunnen de onderbrekingsvlaggen voor overloop en stapeloverloop **niet meer** geplaatst worden[1]. Dit masker is nuttig indien men bijvoorbeeld alle berekeningen **modulo** 10.000.000.000 wil uitvoeren en indien men het stapelregister **R9** voor andere doeleinden dan voor het bijhouden van het adres van de top van de stapel wenst te gebruiken. De overloop- en stapeloverloop-indicatoren (**OVI** en **SOI**) zullen echter wel nog geplaatst worden.

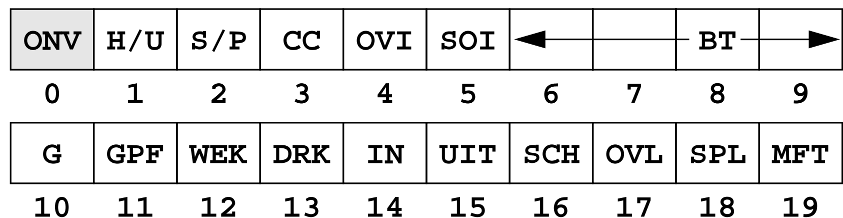
Prioriteiten

Zoals bij je thuis de voordeurbel en de telefoon tegelijkertijd kunnen rinkelen, zo kunnen verschillende programma-onderbrekingen tegelijkertijd aangevraagd worden.

FIGUUR 4-14. Prioriteiten bij onderbrekingen.

Jij zal moeten kiezen welke van de twee onderbrekingen het eerst afgehandeld zal worden. Zo zal de processor ook een keuze moeten maken. Hiervoor wordt meestal een **prioriteitsschema** gebruikt. De soorten onderbrekingen worden volgens hun belangrijkheid gerangschikt. De processor zal eerst ingaan op de belangrijkste aanvraag. Wanneer de processor ingaat op een PO, wil hij niet meer gestoord worden door onderbrekingen die minder of even belangrijk zijn. In het PTW wordt in de eerste cijfer-positie (PTW_0) het huidige **onderbrekingsniveau** (**ONV**) bijgehouden; zie ook figuur 4-15. De processor zal alleen ingaan op PO-aanvragen die een hogere prioriteit hebben dan de huidige waarde van **ONV** (voor zover ze niet 'gemaskeerd' zijn). Als de processor ook daadwerkelijk ingaat op een PO-aanvraag, krijgt **ONV** een nieuwe waarde: de prioriteit van deze PO. (Merk op dat de vorige waarde van het onderbrekingsniveau reeds weggeborgen werd op de stapel; het maakt namelijk deel uit van het eerste gedeelte van het PTW.)

1. Mochten deze onderbrekingsvlaggen vóór het opzetten van het **GPF**-masker reeds opstaan, dan worden ze door het opzetten van dit masker terug afgezet.



FIGUUR 4-15. Het huidige onderbrekingsniveau.

De prioriteiten worden in tabel 4-1 (pag. 112) weergegeven. Er zit een zekere logica in de toekenning van deze prioriteiten: een machinefout of een ongeldige instructie is een zeer ernstige aangelegenheid, dus moet het besturingsprogramma hier zo snel mogelijk op reageren. Programmafouten (PO_8 en PO_7) komen op de volgende plaats, vermits de resultaten van de berekeningen onbetrouwbaar geworden zijn. Bij de randapparatuur krijgen ‘snelle’ randapparaten zoals de schijf (PO_6) een hogere prioriteit dan ‘trage’ randapparaten zoals het scherm (PO_5), het toetsenbord (PO_4) of de drukker (PO_3).

Programma-onderbrekingen en de bevelencyclus

De **vierde stap van de bevelencyclus** wordt voor de DRAMA-machine als volgt verfijnd:

a. indien `G` opstaat, begin de volgende bevelencyclus (d.w.z. negeer alles wat hierna volgt; er zijn immers geen programma-onderbrekingen toegestaan),

b. onderzoek de PO-vlaggen die een ***hogere*** prioriteit hebben dan het huidige onderbrekingsniveau (`ONV`),

(merk op dat indien `ONV` = `9`, geen enkele PO nog kan optreden)

c. negeer deze die een waarde ‘`0`’ hebben (er is geen aanvraag) of wiens overeenkomstig PO-masker ‘`1`’ is (deze worden expliciet uitgesteld)

(merk op dat wanneer `GPF` opstaat, de onderbrekingsvlaggen voor PO_7 en PO_8 niet kunnen opstaan),

d. indien er geen overblijven, hoeft de processor niets extra te doen, anders:

e. kies deze met het hoogste nummer; veronderstel dat dit PO_k is

i. plaats het eerste deel van het PTW, namelijk $PTW_{0..9}$ op de stapel

ii. `ONV` $\leftarrow k$ // of verhoog het onderbrekingsniveau

iii. PO-vlag[k] $\leftarrow$ `0` // zet de PO-vlag af

iv. `BT` $\leftarrow$ PO-vector[k] // wat eigenlijk overeenkomt met `SPR.i 9990+k`

4.3.6 Programma-onderbrekingsroutines

Indien de telefoon rinkelt, en je wilt daarop ingaan (d.w.z. je hebt geen watjes in je oren gestoken; in computertaal: het masker is '`0`') zal je normaal de volgende serie acties uitvoeren: de hoorn opnemen, je kenbaar maken, luisteren wat de opbeller je te zeggen heeft, daarop antwoorden, en tenslotte de hoorn terug neerleggen.

Voor elke programma-onderbreking zal ook een programma geschreven worden dat de nodige acties uitvoert. Wanneer de processor ingaat op een aanvraag tot PO, zal hij beginnen met de uitvoering van dit programma dat we ook de ***PO-behandelingsroutine*** of verkort de ***PO-routine*** noemen (in het Engels spreekt men over een '*interrupt service routine*' of een '*interrupt handler*'). Het adres van de eerste instructie van deze routine wordt bewaard in de overeenkomstige PO-vector.

Hoe ziet een typische PO-routine eruit? In de eerste plaats moeten we —na het uitvoeren van deze routine— in staat zijn de uitvoering van het vorige programma (d.i. het programma dat de processor aan het uitvoeren was op het ogenblik dat de PO optrad) verder te zetten. De oude waarde van de BT werd reeds op de stapel bewaard. Is dit echter voldoende om na de PO het oorspronkelijke programma verder te zetten? Veronderstel dat de processor bezig was met de uitvoering van het volgende programma-fragment:

```
HIA.w  R2,50
HIA.w  R1,10
OPT    R1,R2
...
```

Stel dat een PO (bijvoorbeeld vanwege het toetsenbord) optreedt juist nà de uitvoering van `HIA.w R1,10`. Op dat ogenblik is de inhoud van register **`R1`** gelijk aan '`10`', en die van register **`R2`** '`50`'. Wanneer we na de uitvoering van de PO-routine dit programma verder zetten, zal de volgende instructie die uitgevoerd zal worden `OPT R1,R2` zijn. Het resultaat zou moeten zijn dat **`R1`** nu '`60`' bevat, en dit zal maar correct zijn indien **`R1`** juist voor de uitvoering van de optelling de waarde '`10`' heeft en **`R2`** '`50`' bevat. Maar **`R1`** of **`R2`** zouden kunnen gewijzigd zijn tijdens de uitvoering de PO-routine! Deze laatste routine zal dus de nodige maatregelen moeten nemen zodat geen essentiële gegevens verloren gaan: de inhouden van alle accumulatoren zullen helemaal in het begin van de routine bewaard worden en helemaal op het einde hersteld worden tot hun oorspronkelijke waarde[1]. Een typische PO-routine wordt in figuur 4-16 (pag. 119) getoond. Merk op dat we ook de hardware hadden kunnen aanpassen, zodat bij elke programma-onderbreking de inhouden van de accumulatoren automatisch door de apparatuur op de stapel worden gezet, samen met het PTW. Echter, niet elke PO-routine heeft alle rekenregisters nodig; bovendien zou door het gebruik van de stapel 'stapeloverloop' kunnen optreden. Op een reële computer zal men vaak een aparte stapel voorzien voor het besturingsprogramma.

1. Het volstaat uiteraard alleen die accumulatoren te bewaren (en later te herstellen) die door de PO-routine gebruikt worden.

De KTO-instructie

De **`KTO`**-instructie (**k**eer **t**erug na **o**nderbreking) beëindigt een programma-onderbrekingsroutine. Het is een nieuwe functiecode, die zeer veel gelijkenis vertoont met het **`KTG`**-bevel (namelijk het herstellen van de bevelenteller), doch iets meer doet. Tijdens de uitvoering van het **`KTO`**-bevel wordt ***(door de apparatuur)*** het top-element[1] van de stapel gehaald en in het eerste gedeelte van het PTW geplaatst. Aangezien het eerste gedeelte van het PTW o.a. de bevelenteller (BT) bevat, kan het effect van de uitvoering van een **`KTO`**-instructie als volgt beschreven worden:

- de BT krijgt zijn "oude" waarde terug (namelijk het adres van de instructie die uitgevoerd had moeten worden indien geen PO was opgetreden),
- het onderbrekingsniveau (**`ONV`**) krijgt terug zijn vorige waarde,
- de conditiecode (**`CC`**) en de overloop-indicatoren (**`OVI`** en **`SOI`**) krijgen opnieuw de waarde die ze hadden op het ogenblik dat de PO optrad.

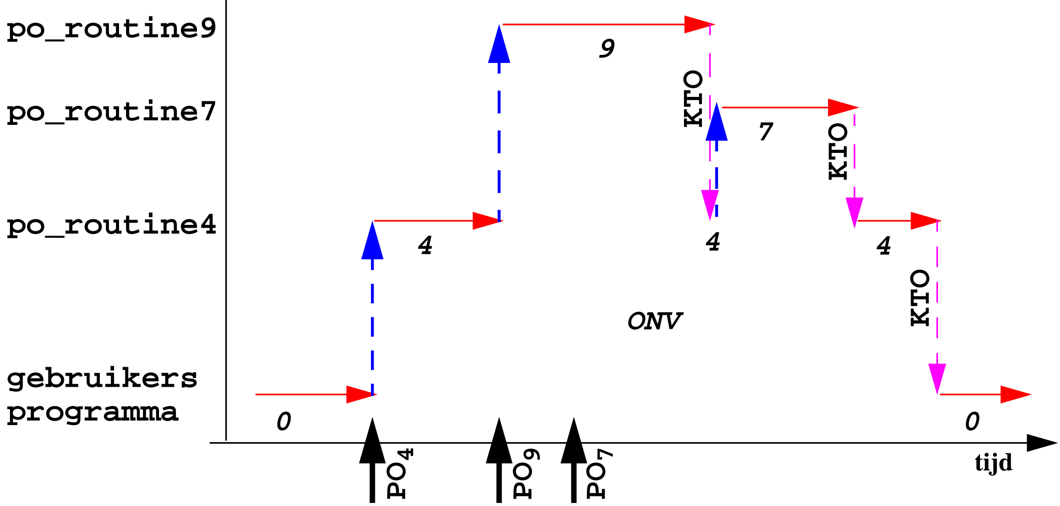
```
| Behandelingsroutine voor POn
                | bewaar de inhouden van de accumulatoren
PO_ROUTINEn:    BIG      R0,BEWAAR+0
                BIG      R1,BEWAAR+1
                ...
                BIG      R9,BEWAAR+9
                | hier begint de eigenlijke behandeling

                ...

                | herstel de inhouden van de accumulatoren
                HIA      R0,BEWAAR+0
                HIA      R1,BEWAAR+1
                ...
                HIA      R9,BEWAAR+9
                | keer terug
                KTO
     BEWAAR:    RESGR    10
```

FIGUUR 4-16. Een typische PO-behandelingsroutine.

1. Indien de PO-routine de oorspronkelijke inhoud van de stapel niet overschreven heeft —wat ze niet zou mogen doen— vind je op de top van de stapel $PTW_{0..9}$, namelijk de oude waarde van het PTW die op het ogenblik van de onderbreking op de stapel werd gezet. De uitvoering van **`HIA R9,BEWAAR+9`**, juist vóór de **`KTO`**, zorgt ervoor dat het stapelregister (**`R9`**) opnieuw de correcte waarde heeft.

Merk op dat indien ***tijdens de behandeling van een programma-onderbreking*** de PO-vlag voor deze PO opnieuw geplaatst wordt, het onderbrekingsniveau verhindert dat ernaar geluisterd wordt. Na de uitvoering van de **KTO**-instructie kan dus opnieuw een programma-onderbreking van deze soort optreden (tenzij ze vóór de uitvoering van de **KTO**-instructie gemaskeerd werd).

Figuur 4-17 schetst het verloop van de uitvoering van programma's gedurende een zeker tijdsinterval. De horizontale as duidt de tijd aan. Op de verticale as wordt aangeduid welk programma-onderdeel de processor aan het uitvoeren is. De dikke pijlen geven aan wanneer een PO-vlag opgezet wordt. Onder elke horizontale lijn staat het onderbrekingsniveau dat dan van toepassing is. De gebroken lijnen zijn de overgangen die ofwel door een PO veroorzaakt wordt (pijl naar boven), ofwel door de **KTO**-instructie (pijl naar beneden). In het begin is de processor bezig met de uitvoering van een gebruikersprogramma. We veronderstellen dat het onderbrekingsniveau (**ONV**) '**0**' is, en dat geen enkel masker opstaat. De figuur toont aan dat de uitvoering van een PO-routine kan onderbroken worden voor de uitvoering van de PO-routine voor een meer dringende PO (hogere prioriteit). Bijvoorbeeld, **po_routine4** wordt tweemaal onderbroken: eerst om PO_9 af te handelen en daarna om PO_7 af te handelen. Merk ook op dat **po_routine9** eerst volledig afgewerkt moet zijn alvorens met de uitvoering van **po_routine7** begonnen kan worden. Als **po_routine9** afgewerkt is, zal de uitvoering van de **KTO**-instructie het onderbrekingsniveau opnieuw op '4' herstellen; na de uitvoering van de **KTO** zal echter onmiddellijk een nieuwe PO optreden (PO_7); er wordt niet eerst een instructie van **po_routine4** uitgevoerd!.



FIGUUR 4-17. De uitvoering van programma's gedurende een zeker tijdsinterval.

4.3.7 Geprogrammeerde programma-onderbreking

Eén PO-vlag (PO_1) wordt niet na een fout of door besturingseenheden van randapparaten geplaatst, maar via een speciale instructie: de **onderbrekingsinstructie** of `OND`.

`OND αβγδ` zal de PO_1-vlag plaatsen. Dit betekent dat na de uitvoering van deze instructie normaal een PO optreedt (indien het globale masker afstaat en het onderbrekingsniveau dit toelaat[1]). In feite heeft deze instructie geen argument nodig; om efficiëntieredenen hebben de ontwerpers van de DRAMA-machine toegelaten dat de laatste vier cijfers van deze instructie om het even welke waarde mogen hebben. (Het assembleerprogramma zorgt ervoor dat deze instructie vertaald wordt tot **619999**αβγδ.) De waarde αβγδ wordt echter ***niet*** gebruikt tijdens de uitvoering van de instructie. Zoals we verder zullen zien (in voorbeeld 4-9 (op pag. 168)) kan de behandelingsroutine voor PO_1 dit argument zelf ophalen uit het geheugen en —afhankelijk van de waarde— een bepaalde routine oproepen.

Deze instructie is erg belangrijk in de context van besturingssystemen. Ze zal toelaten dat een programma op een ‘ordelijke’ wijze hulp vraagt aan het besturingsprogramma bij moeilijke taken (zoals het laden van een programma, het lezen of schrijven van een bestand, …). Het argument αβγδ zal gebruikt worden om een onderscheid te maken tussen de verschillende diensten die het BP aanbiedt. We zullen het daarom het **‘dienstnummer’** noemen. Bijvoorbeeld bij **0001** vraagt het programma aan het BP om een getal in te lezen via het toetsenbord, bij **0002** zal het BP een getal afdrukken op het scherm, enz.

Initialisatie DRAMA-machine

Op de DRAMA-machine is er geen besturingsprogramma aanwezig. Dus hebben de DRAMA-ontwerpers beslist dat onmiddellijk na het opstarten van de DRAMA-machine alle onderbrekingen onderdrukt worden (d.w.z. alle maskers in het PTW hebben de waarde ‘**1**’, en het onderbrekingsniveau is ‘**0**’). Zie ook figuur 4-18.

0	H/U	S/P	CC	OVI	SOI	◄──	── BT ──		──►
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
10	11	12	13	14	15	16	17	18	19

FIGUUR 4-18. Initialisatie van het PTW bij de drama-machine.

1. Gewoonlijk zal deze instructie alleen gebruikt worden indien zowel het onderbrekingsniveau als het globale masker `0` zijn.

Indien je dus programma-onderbrekingen wil toelaten, zal je eerst de maskers (m.b.v. **MKL**) op nul moeten zetten:

```
MKL   WEK
MKL   DRK
MKL   IN
MKL   UIT
MKL   SCH
MKL   OVL
MKL   SPL
MKL   MFT
MKL   GPF
MKL   G
```

Bovendien zal de 'gewone programmeur' dit programmaatje niet kunnen uitvoeren. Hiervoor heb je speciale privileges nodig! (Zie verder, paragrafen 4.5.2 en 4.5.3.)

De PO-routines zijn minimaal (d.w.z. bestaan uit één instructie, namelijk het **KTO**-bevel). De PO-vectoren bevatten alle het adres **9990**, terwijl op adres **9990** zich de **KTO**-instructie bevindt. Zie ook figuur 4-19.

Adres	Geheugeninhoud	Betekenis	Oorzaak
	...		
9990:	6299999999	KTO	
9991:	0000009990	PO-vector 1	OND
9992:	0000009990	PO-vector 2	WEK
9993:	0000009990	PO-vector 3	DRK
9994:	0000009990	PO-vector 4	IN
9995:	0000009990	PO-vector 5	UIT
9996:	0000009990	PO-vector 6	SCH
9997:	0000009990	PO-vector 7	OVL
9998:	0000009990	PO-vector 8	SPL
9999:	0000009990	PO-vector 9	MFT

FIGUUR 4-19. Initialisatie van de DRAMA machine (vervolg).

De meeste programma-onderbrekingsvectoren op de DRAMA-machine (maar ook op een werkelijke computer) hebben te maken met ***invoer- en uitvoerapparaten***. In de volgende paragraaf zullen we daarom dieper ingaan hoe invoer en uitvoer gerealiseerd worden.

Opgaven

1. Wat zijn programma-onderbrekingen? Waarom zijn ze nodig op een modern computersysteem? Geef minstens twee redenen waarom een programma-onderbrekingsmechanisme nuttig zou kunnen zijn.
2. Leg de bevelencyclus uit aan de hand van een voorbeeldje. Wat gebeurt er bij de uitvoering van een sprongbevel?
3. Wat is een oneindige lus? Is dit echt een probleem voor de processor? Hoe los je dit probleem op?
4. Leg de volgende begrippen uit:
   - programma-onderbrekingsvlag
   - programma-onderbrekingsvector
   - masker
   - prioriteit
5. Welke soorten programma-onderbrekingen bestaan er? Geef voor elke klasse enkele voorbeelden.
6. Wat verstaat men onder het 'uitstellen of verbieden van een programma-onderbreking'? Welke methodes kan je hiervoor gebruiken?
7. Wat is het exacte verschil tussen 'het uitstellen van een programma-onderbreking' en 'het negeren van een aanvraag tot een programma-onderbreking'?
8. Wat gebeurt er wanneer er verschillende aanvragen tot een programma-onderbreking gelijktijdig gesteld worden? Hoe kan het CVE deze aanvragen op een ordelijke wijze afhandelen?
9. Wat bedoelt men met 'een aanvraag tot een programma-onderbreking blijft hangende'?
10. Hoe worden de prioriteiten toegekend aan de verschillende programma-onderbrekingen? Waarom zijn prioriteiten noodzakelijk?
11. Wat is een programma-onderbrekingsroutine? Hoe ziet zo'n typische routine eruit?
12. Beschrijf de volledige bevelcyclus op de DRAMA-machine. Geef duidelijk aan wat er precies gebeurt indien een programma-onderbreking optreedt.
13. Waarom wordt bij een programma-onderbreking het volledige eerste gedeelte van het PTW op de stapel geplaatst? Is het niet voldoende om alleen de bevelenteller weg te bergen?
14. Beschrijf het effect van de **KTO**-instructie.
15. Veronderstel dat de uitvoering van elke PO-routine precies 20 msec duurt. Alleen PO_4 is gemaskeerd. Initieel is het onderbrekingsniveau (**ONV**) gelijk aan nul. Geef via een tijdsdiagram aan wanneer elke PO-routine uitgevoerd wordt, indien:
    - op tijdstip **0** de processor bezig is met de uitvoering van **PROG**,
    - 10 msec later de PO_3-vlag geplaatst wordt,
    - 3 msec later de PO_5-vlag geplaatst wordt,
    - 5 msec later de PO_4-vlag geplaatst wordt,
    - 6 msec later de PO_9-vlag geplaatst wordt,
    - 2 msec later de PO_7-vlag geplaatst wordt.
16. Wat is een geprogrammeerde programma-onderbreking? Waartoe dient ze?
17. Beschrijf het verschil tussen de **KTO** en **KTG** instructies?

18. Geef minstens 3 verschillende manieren om op de DRAMA-machine **alle** programma-onderbrekingen te verbieden.

19. Is er een verschil tussen het opzetten van het **GPF**-masker en het opzetten van zowel het **OVL**- als het **SPL**-masker?

20. Er bestaat geen instructie om het onderbrekingsniveau (**ONV**) een nieuwe waarde te geven. Bedenk een manier om het gelijk aan '**7**' te maken.

21. Waarom zijn voor de uitzonderingstoestanden 'overloop' en 'stapeloverloop' zowel een indicator als een onderbrekingsvlag aanwezig?

22. Waarom lieten de DRAMA-ontwerpers de inhouden van de rekenregisters (**R0...R9**) niet automatisch door de apparatuur op de stapel plaatsen, en bij de uitvoering van de **KTO**-instructie automatisch van de stapel halen?

23. Wat gebeurt er als er stapeloverloop optreedt tijdens het wegbergen van $PTW_{0..9}$ op de stapel? Riskeer je in een oneindige sequentie van programma-onderbrekingen terecht te komen?

4.4 Invoer en uitvoer

Een van de belangrijkste taken van het besturingsprogramma (BP) betreft het 'besturen' van de randapparaten ('*devices*' in het Engels). Dit zijn de in- en uitvoerapparatuur (inclusief de hulpgeheugens). Het BP geeft daartoe de nodige opdrachten (in het Engels '*commands*') aan deze randapparaten, behandelt hun programma-onderbrekingen en probeert fouten die zich kunnen voordoen op te lossen.

Randapparaten vertonen grote verschillen in de wijze waarop opdrachten moeten doorgestuurd worden. Ook de opdrachten zelf zijn erg afhankelijk van het soort randapparaat. De gebruiker van het computersysteem (het gebruikersprogramma) is slechts geïnteresseerd in hoog-niveau opdrachten (zoals het lezen of schrijven van een bepaalde hoeveelheid informatie op het apparaat). Het besturingsprogramma zal deze hoog-niveau opdrachten omzetten naar de specifieke opdrachten voor het randapparaat.

Tot hiertoe gebeurde invoer en uitvoer in DRAMA via de **`LEZ`**- en **`DRU`**-opdrachten. Alhoewel ze op het eerste zicht eerder "primitief" lijken, worden er toch complexe operaties uitgevoerd. Bijvoorbeeld, het **`LEZ`**-bevel moet de verschillende toetsindrukken die via een bepaalde code (bv. ASCII) worden voorgesteld, ontvangen van het toetsenbord tot de **`return`**-toets wordt ingedrukt. Blanko's vóór en nà de cijfer-toetsindrukken worden genegeerd. Indien letter-toetsen worden ingedrukt, zal een fout gegenereerd worden. Dan moeten deze afzonderlijke ASCII-waarden omgezet worden tot een decimaal getal van 10 cijfers. Indien de eerste beduidende toetsindruk een '–' is, moet het bekomen getal omgezet worden naar zijn 10-complement voorstelling. Dit is zeker geen eenvoudige taak!

Op het einde van deze paragraaf zullen we aangeven hoe we deze 'hoog-niveau' operaties kunnen blijven aanbieden aan de programmeur.

4.4.1 Randapparaten

Randapparaten bestaan meestal uit een **mechanisch** en een **elektronisch** gedeelte (zie hoofdstuk 2). Het elektronische gedeelte wordt de besturingseenheid of bestuurder genoemd (in het Engels '*controller*' of soms '*adapter*'). Bij microcomputers worden deze bestuurders vaak onder de vorm van 'insteekkaarten' ontworpen. Dikwijls kan één bestuurder meerdere (twee, vier, acht, …) identieke apparaten besturen.

In wat volgt geven we een korte beschrijving van een 'typisch' randapparaat. In appendix D (op pag. 218 e.v.) worden vier randapparaten in detail besproken: het toetsenbord (louter invoer), het scherm en de drukker (enkel uitvoer) en de harde schijf (zowel in- als uitvoer).

De processor communiceert met een randapparaat via de bestuurder. Deze laatste heeft een aantal **speciale registers** (ook soms ***poorten*** genoemd) waarin de processor opdrachten en gegevens kan schrijven of waaruit hij gegevens en informatie (over het apparaat) kan lezen.

Het lezen en schrijven in deze registers kan op twee manieren gebeuren:

1. met behulp van **speciale in- en uitvoer instructies**, (zoals `INV` en `UTV`),

`INV Ri,Px:`	kopiëert de inhoud van poort `Px` naar register `Ri`, en zet de conditiecode.
`UTV Rj,Py:`	schrijft de inhoud van register `Rj` in poort `Py`, en zet de conditiecode.

2. door een **'inpassing'** van deze poorten **in het centrale geheugen** van de computer (in het Engels duidt men dit aan met de term '*memory mapped I/O*'). Hier wordt een deel van het gewone adresbereik gebruikt om deze registers aan te duiden. Bijvoorbeeld, op de DRAMA-machine hadden we de geheugenadressen `9900` tot en met `9909` kunnen gebruiken om de poorten `P0` tot en met `P9` aan te duiden; deze adressen komen **niet** overeen met een geheugenregister. Het lezen en schrijven in de poorten kan eenvoudig gebeuren m.b.v. de `HIA` en `BIG` instructies. Bijvoorbeeld:

```
HIA R3,9901
```

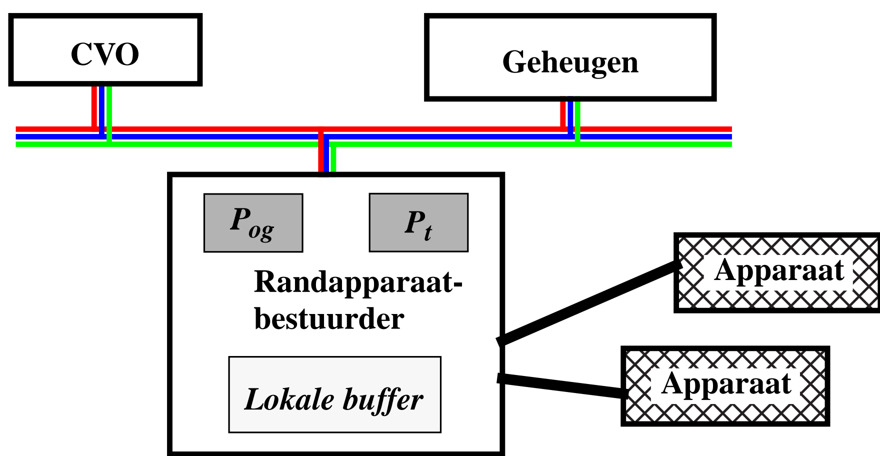
zal de inhoud van poort `P1` in register `R3` kopiëren. Zie ook figuur 4-20.

0000:	
	…
9899:	
9900:	P_0
9901:	P_1
	…
9909:	P_9
9910:	
	…
9999:	

FIGUUR 4-20. Inpassing van poorten in het adresbereik.

Het nadeel van deze methode is dat het beschikbaar centraal geheugen kleiner wordt (slechts `9990` geheugenregisters i.p.v. `10000`). Daarom hebben we in DRAMA gekozen voor de speciale instructies.

Op de DRAMA-machine heeft elke bestuurder slechts twee poorten: een opdracht- en gegevenspoort (verkort P_{og}) en een toestandspoort (verkort P_t). Zie ook figuur 4-21.



FIGUUR 4-21. Een typische bestuurder van een randapparaat.

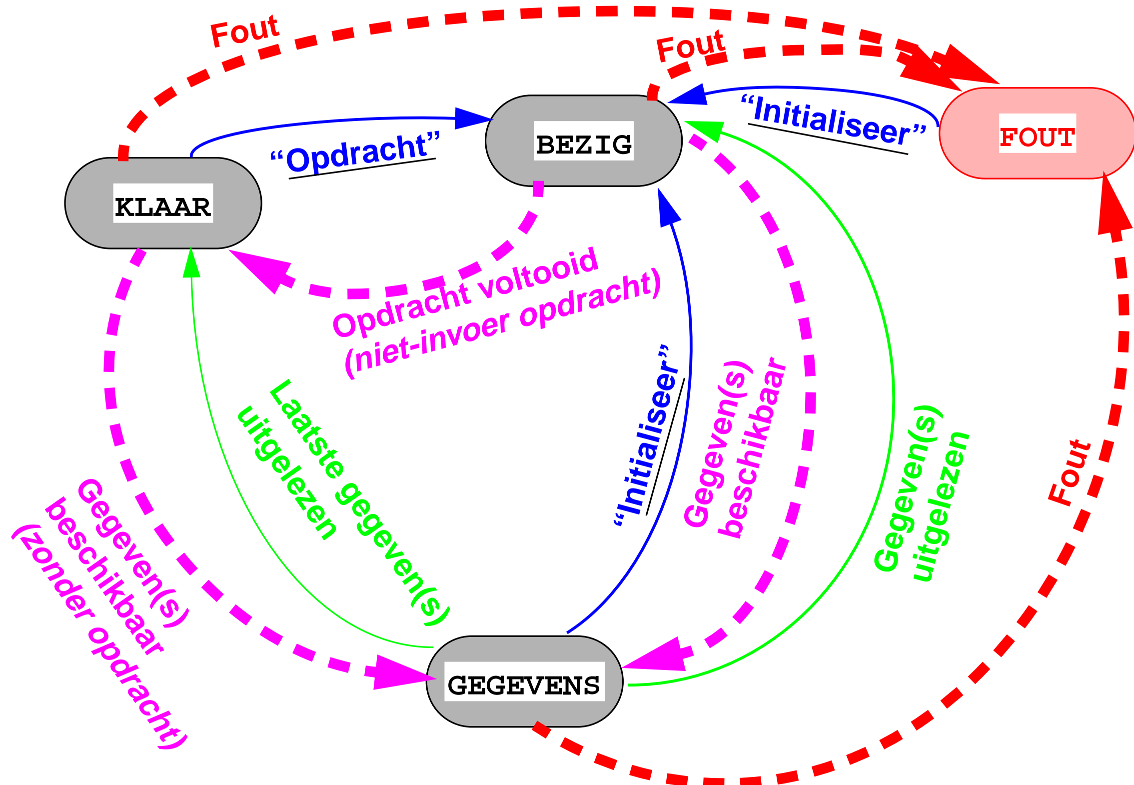
Uit de toestandspoort (P_t) kan alleen gelezen worden (m.b.v. **`INV`**). De bestuurder zal in deze poort de toestand van het randapparaat aangeven. De waarden die in deze poort kunnen staan zijn erg afhankelijk van het soort (en merk) van randapparaat. Om de zaken iets te vereenvoudigen zullen voor elk randapparaat vier (dezelfde) toestanden voorzien worden.

TABEL 4-3. De toestanden van een randapparaat bestuurder.

Waarde	Naam	Verklaring
`0000000000`	**`KLAAR`**	De bestuurder is klaar voor een nieuwe opdracht.
`0000000001`	**`BEZIG`**	Het toestel is bezig met de uitvoering van een opdracht.
`0000000002`	**`GEGEVENS`**	Er zijn gegevens, komende van het randapparaat, beschikbaar in de gegevenspoort.
`9999999999`	**`FOUT`**	Er heeft zich een fout voorgedaan.

Naast deze vier toestanden, kunnen er nog extra voorzien worden voor een bepaald randapparaat. Figuur 4-22 geeft het toestandsdiagram voor een bestuurder weer. Sommige randapparaten hebben geen **`GEGEVENS`**-toestand, namelijk de uitvoer-apparaten (zoals scherm en drukker), tenzij het apparaat toch gegevens kan leveren (bijv. de huidige positie van de cursor op een scherm of de beschikbare fonts bij een drukker, ...). De pijlen tonen de mogelijke overgangen tussen de toestanden. Naast elke pijl staat de oorzaak voor de overgang. Bijvoorbeeld: wanneer de bestuurder '**`KLAAR`**' is om een bevel te ontvangen, en de processor plaatst een opdracht in de opdrachtpoort, dan zal de bestuurder aangeven dat hij '**`BEZIG`**' is met de uitvoering van de opdracht. (De onderlijnde woorden zijn expliciete opdrachten van de processor voor de bestuurder.)

Een 'niet-invoer opdracht' is bijvoorbeeld een 'initialisatie-opdracht' of een 'uitvoer opdracht' ("druk een symbool af" of "schrijf iets weg").



FIGUUR 4-22. Het toestandsdiagram voor een bestuurder.

Wanneer zich een fout voordoet komt het apparaat in de **FOUT**-toestand (ongeacht de toestand waarin het zich bevond). Fouten kunnen zich voordoen wanneer:

- een onbestaande opdracht gegeven wordt aan de bestuurder,
- een opdracht doorgegeven wordt terwijl het apparaat hiervoor niet **KLAAR** is (vb. nog bezig is met de uitvoering van een vorige opdracht, of reeds in de **FOUT**-toestand is),
- het apparaat nieuwe gegevens ter beschikking wil stellen, terwijl de vorige nog niet uitgelezen zijn (dit is een vorm van overloop),
- er zich een fout voordoet in het apparaat, bijvoorbeeld: het papier zit klem in de drukker, er is geen papier meer, of er heeft zich een elektrische storing voorgedaan,
- ...

Alleen de herinitialisatie (dit is een speciale opdracht) van het apparaat kan het apparaat uit de **FOUT**-toestand (via **BEZIG**) naar de **KLAAR**-toestand brengen.

Zes overgangen worden met een dikke stippellijn getekend. Wanneer de bestuurder deze overgang maakt, zal hij een programma-onderbreking aanvragen (d.i. de PO-vlag die met de bestuurder geassocieerd is op ‘**1**’ plaatsen). Deze overgangen zijn weldoordacht gekozen; bij deze overgangen is een tussenkomst van de processor gewenst. In tabel 4-4 wordt voor elk van deze overgangen aangegeven welke acties van de processor (eigenlijk van het besturingsprogramma dat de processor dan uitvoert) verwacht worden.

TABEL 4-4. Toestandsovergangen met een programma-onderbrekingsaanvraag.

Begin-toestand	Eind-toestand	Gewenste acties vanwege de processor
`???`	`FOUT`	De processor moet de fout herstellen (herinitialisatie van het randapparaat)
`BEZIG`	`KLAAR`	De bestuurder is klaar met de uitvoering van een opdracht. De processor mag een nieuwe opdracht geven.
`BEZIG`	`GEGEVENS`	De (via een opdracht) gevraagde gegevens zijn beschikbaar. De processor kan ze uitlezen.
`KLAAR`	`GEGEVENS`	De bestuurder heeft (ongevraagde) gegevens beschikbaar. De processor kan ze uitlezen.

Gegevens die van het randapparaat afkomstig zijn, worden door de processor m.b.v. het **`INV`**-bevel uit de gegevenspoort (P_{og}) ingelezen.

Opdrachten worden met behulp van het **`UTV`**-bevel in de opdrachtpoort (P_{og}) geschreven. De opdrachten zijn erg afhankelijk van het soort randapparaat. Op de DRAMA-machine zal in de meeste gevallen het eerste cijfer aangeven welke actie van het randapparaat verwacht wordt. Appendix D (pag. 218 e.v.) geeft een overzicht van de bevelen die voor de verschillende DRAMA-randapparaten beschikbaar zijn.

4.4.2 Intermezzo: I/O en PO-routines in C.

In wat volgt zullen we de volgende twee C-functies gebruiken om een waarde uit een poort te lezen of een waarde in een poort te schrijven:

- **`int getPort(int poortnummer)`**
- **`void putPort(int poortnummer, int waarde)`**

We veronderstellen ook dat een aantal constanten gedefinieerd zijn:

```
#define KLAAR   0
#define BEZIG   1
#define GEG     2
#define FOUT    -1
```

Bijvoorbeeld, indien we een opdracht willen wegschrijven in poort 1, als het apparaat er klaar voor is (in poort 0 staat dan een nul), zullen we de volgende C-notatie gebruiken:

```
int opdracht = ...;           // De opdracht
if (getPort(0) == KLAAR)      // Lees de toestand
  putPort(1, opdracht);       // Geef opdracht
```

Daarenboven zullen we een PO-routine in C schrijven als:

```
interrupt po_routine()
{

   ...

}
```

Merk op dat dit een ***uitbreiding*** van C is: de C-taal kent het sleutelwoord **interrupt** niet. We zullen dus onze C-compilator moeten uitbreiden. Een PO-routine is vergelijkbaar met een gewone procedure. De vier verschilpunten zijn:

- in het begin van de routine worden alle gebruikte accumulatoren bewaard (op een vaste plaats),
- op het einde van de routine worden de inhouden van deze accumulatoren hersteld (ze krijgen terug hun oorspronkelijke waarde),
- de routine keert terug met een **KTO** instructie i.p.v. met een **KTG** instructie,
- er wordt voldoende plaats gereserveerd voor het bewaren van de accumulatoren (deze plaats krijgt een symbolisch adres dat afgeleid is van de naam van de PO-routine).

De belangrijkste reden voor de uitbreiding van de C-taal is dat deze aanpassingen zelf niet in C te schrijven zijn. We veronderstellen dus dat onze aangepaste C-compilator, deze instructies inlast bij elke routine van het type '**interrupt**'. De vertaalde versie van deze routine ziet er bijgevolg uit als:

```
PO_ROUTINE:
            BIG R0,BEW_PO_ROUTINE          } Ingelast
            BIG R1,BEW_PO_ROUTINE+1
            ...

            | vertaalde code voor ...


            HIA R0,BEW_PO_ROUTINE          } Ingelast
            HIA R1,BEW_PO_ROUTINE+1
            ...

            KTO                            } i.p.v. KTG

BEW_PO_ROUTINE:                            (Gereserveerd)
            RESGR 10 | of minder afh. van # gebruikte acc
```

4.4.3 In- en uitvoer organisatievormen

In deze paragraaf zullen we beschrijven op welke manier de bestuurder van een randapparaat opdrachten krijgt van de processor. Dit kan op verschillende wijzen geïmplementeerd worden:

a. geprogrammeerd (volledig gestuurd door de processor),

b. m.b.v. programma-onderbrekingen,

c. m.b.v. directe geheugentoegang,

d. m.b.v. speciale invoer/uitvoer processoren (ook wel kanaalbestuurders of zelfs kanalen [in het Engels, '*channels*'] genoemd),

e. m.b.v. satelliet-computers.

(a) Geprogrammeerde in- en uitvoer

Dit is de oudste methode die nu nog alleen gebruikt wordt op zeer eenvoudige microprocessoren, of indien de bestuurder van het randapparaat niet in staat is programma-onderbrekingen aan te vragen. We zullen deze methode alleen uitleggen voor 'pure' invoer- of uitvoerapparaten (dus geen hulpgeheugens).

Typisch aan deze methode is dat alle gegevenstransport, teken per teken gebeurt onder **controle van de processor** (door het uitvoeren van `INV-` en `UTV`-bevelen). Alleen de processor beslist wanneer een nieuwe opdracht wordt gegeven of wanneer gegevens worden uitgelezen. Toch is er een zekere synchronisatie nodig tussen de processor en de bestuurder van het randapparaat:

- de bestuurder is niet bereid om op gelijk welk ogenblik een nieuwe opdracht te ontvangen, hij moet er eerst 'klaar' voor zijn;
- zo ook, kunnen slechts gegevens uitgelezen worden (uit de gegevenspoort) indien ze beschikbaar zijn, d.w.z. indien de bestuurder zich in de 'gegevens'-toestand bevindt.

De processor zal dus eerst moeten nagaan of de bestuurder zich in de 'juiste' toestand bevindt. Dit kan hij doen door de waarde uit de toestandspoort in te lezen en te vergelijken met de 'gewenste' waarde. Zolang de toestand hiervan verschilt, kan de processor niets anders doen dan opnieuw nagaan of de toestand nog niet gewijzigd is tot de 'gewenste' toestand. Het spreekt voor zich dat een dergelijke manier van werken niet erg efficiënt is: zolang de bestuurder van het randapparaat zich niet in de 'gewenste' toestand bevindt, doet de processor geen 'nuttig' werk.

Je kan dit vergelijken met een auto die met een draaiende motor voor een gesloten overweg staat. De motor van de auto is de 'processor'. Zolang de overweg gesloten is, verricht de motor 'arbeid' (hij blijft draaien), maar die arbeid wordt niet nuttig aangewend.

De twee kenmerken van 'geprogrammeerde' invoer en uitvoer zijn:

- **alle gegevenstransport gebeurt via de processor**; dus, elk teken dat de bestuurder nodig heeft wordt door het CVO d.m.v. de **`UTV`**-instructie in de opdracht/gegevenspoort geplaatst; en elk teken dat de bestuurder ter beschikking houdt van het CVO, wordt d.m.v. het **`INV`**-bevel uit de gegevenspoort ingelezen.
- de processor **synchroniseert** zichzelf met de bestuurder van het randapparaat. Alvorens een opdracht (m.b.v. **`UTV`** in de opdrachtpoort) te geven aan de bestuurder, moet de processor eerst nagaan of de bestuurder 'klaar' is om een opdracht te ontvangen. Evenzo, alvorens een teken kan ingelezen worden (m.b.v. **`INV`** uit de gegevenspoort), moet de processor nagaan of de bestuurder 'gegevens' ter beschikking heeft. Dit nakijken gebeurt door voordurend de toestand van de bestuurder te testen (d.w.z. de waarde van de toestandspoort in te lezen m.b.v. **`INV`** en te vergelijken met de gewenste waarde m.b.v. **`VGL`**). De processor is eigenlijk aan het wachten (tot de bestuurder zich in de gewenste toestand bevindt), maar dit wachten gebeurt op een 'actieve' wijze (door het testen van een waarde in een lus). We zullen deze vorm van wachten aanduiden met de term '**actief wachten**' (in het Engels spreekt men over '*busy waiting*').

Sommige randapparaten zijn niet rechtstreeks verbonden met de computer, maar via een communicatieverbinding. Dit heeft tot gevolg dat de processor nu niet meer rechtstreeks aan de toestands- en opdracht/gegevenspoort van de bestuurder van dat randapparaat kan. (Echter wel aan de poorten van de bestuurder van de communicatieverbinding; deze poorten worden gebruikt om boodschappen te versturen of te ontvangen.) In dit (eerder uitzonderlijke) geval moet de processor 'vragen' aan de bestuurder van een invoerapparaat "of er gegevens beschikbaar zijn", of aan de bestuurder van een uitvoerapparaat "of hij klaar is om een nieuwe opdracht te ontvangen". Dit gebeurt door via de communicatieverbinding een 'vraag'-boodschap te sturen naar de bestuurder van het apparaat. In het Engels wordt dit '*polling*' genoemd, wat we vrij kunnen vertalen als 'polsen' of 'bevragen'. De bestuurder zal dan ofwel 'ja' of 'neen' antwoorden, en eventueel de beschikbare gegevens meesturen. In de rest van deze paragraaf zullen we deze mogelijkheid niet langer beschouwen.

Laten we het voorgaande even toepassen op een eenvoudig uitvoerapparaat: het scherm van een terminal. Veronderstel dat de toestands- en opdracht-poort van het scherm respectievelijk P_2 en P_3 zijn. De bestuurder kent alleen heel eenvoudige bewerkingen: initialisatie, het schrijven van één letter op het scherm, en het verplaatsen van de huidige positie. Een volledige beschrijving van het scherm vind je in paragraaf D.1 (op pag. 220); in de volgende tabel wordt een fragment van de beschikbare bevelen getoond.

TABEL 4-5. De opdrachten voor de schermbestuurder.

Opdracht	**Betekenis**
...	
`1000000ccc`	Druk een letter af op het scherm `ccc`: de ASCII-voorstelling van het af te beelden symbool
...	

Voorbeeld 4-2 toont een eenvoudig programma dat een 100-tal letters (`Voorbeeld:` ...) afbeeldt op het scherm. Voor elke letter moet een opdracht '`1000000ccc`' gegeven worden aan de bestuurder, waarbij '`ccc`' de ASCII-voorstelling van de letter is. Bijvoorbeeld, voor de letter '`V`' is dit '`1000000086`', voor de letter '`o`' wordt dit '`1000000111`', enz. De lijst met de ASCII-waarden vind je in tabel A-1 (pag. 200).

Bestudeer nauwgezet de werking van het programma en zoek waar de processor 'actief wacht'. Kan je aantonen dat er tijdens dit actief wachten nogal wat tijd kan verloren gaan? Indien het scherm traag werkt in vergelijking met de processor, kan dit verlies hoog oplopen.

Het actief wachten in de C-code gebeurt in de schuin gedrukte while-opdracht. In de DRAMA-code wacht de processor actief in de lijnen *(L10)* tot *(L12)*.

Voorbeeld 4-2. ***Schrijven op het scherm met actief wachten.***
Het volgende programma drukt 100 letters af op het scherm. We veronderstellen dat de af te drukken letters ("`Voorbeeld` ...") zich reeds in een geheugenzone bevinden die start bij symbolisch adres `ZONE`, één letter per geheugenregister. Elke letter wordt voorgesteld via zijn ASCII-code. Telkens de bestuurder klaar is om een opdracht te ontvangen, zal de processor de volgende schrijf-opdracht doorgeven via poort `P3`. We geven eerst het C-programma:

```
#define PtS   2    | toestandspoort
#define PogS  3    | opdracht/gegevenspoort

int aantal = 100; | aantal uit te schrijven tekens
int zone[100] = { (int) 'V', (int) 'o', ...};
int dru_opd = 1000000000;
int index = 0;    | index in zone

while (index < aantal)
{
    int opdracht = dru_opd + zone[index++];
    while (getPort(PtS) != KLAAR) { }   /* Actief wachten! */
    putPort (PogS, opdracht);
}
```

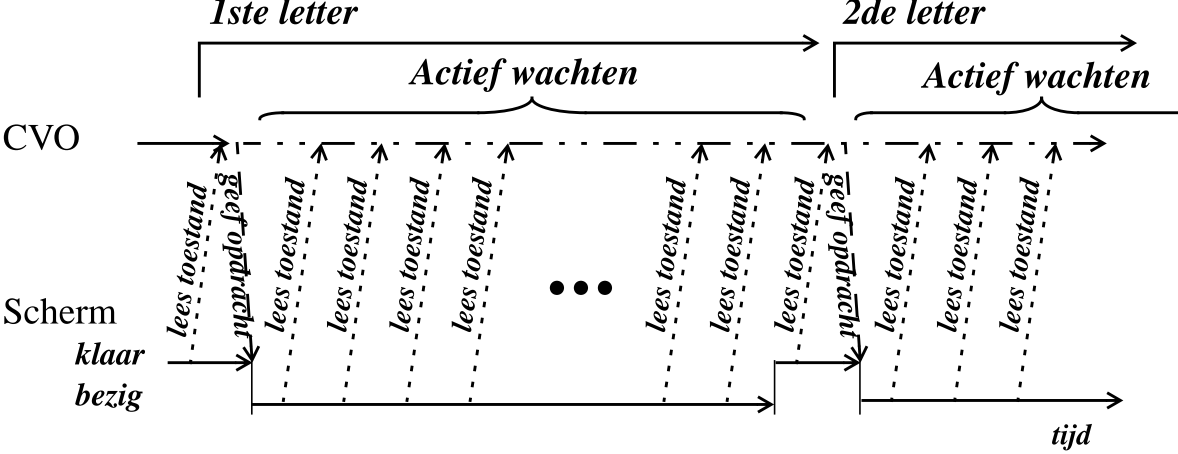
en hier volgt dan de DRAMA*-code:*

```
L1:  MEVA     PtS,2
L2:  MEVA     PogS,3
L3:  MEVA     KLAAR,0
L4:           HIA.w   R6,0           | Index
L5:  WHILE:   VGL     R6,AANTAL      |
L6:           VSP     GRG,ENDWH      | Alles uitgeschreven?
```

```
L7:             | Stel volgende opdracht samen: 1000000ccc
L8:             HIA     R5,DRU_OP
L9:             OPT     R5,ZONE(R6+)
L10:  WACHT:    INV     R1,<PtS>        |
L11:            VGL.w   R1,<KLAAR>      | KLAAR?
L12:            VSP     NGEL,WACHT      |
L13:            UTV     R5,<PogS>       | Schrijf V, o, …
L14:            SPR     WHILE
L15:  ENDWH:    ...
L16:  DRU_OP:   1000000000              | 1000000ccc = beeld ccc af
L17:  AANTAL:   100
L18:  ZONE:     0000000086              | Letter V
L19:            0000000111              | Letter o
                …
```

Tijdsdiagram

Figuur 4-23 toont het tijdsdiagram. Je kan duidelijk zien dat de processor het grootste deel van zijn tijd actief aan het wachten is op de schermbestuurder. Tijdens dit wachten wordt er geen nuttig werk verricht.



FIGUUR 4-23. Tijdsdiagram bij uitvoer met actief wachten.

Performantie-berekening

Veronderstel dat het scherm in staat is 1000 letters per seconde op het scherm af te beelden. De DRAMA-computer heeft gemiddeld 1 microseconde nodig per instructie (1 MIPS machine). Indien we een 100-tal letters op het scherm willen afdrukken zal het scherm hiervoor een 100-tal milliseconden nodig hebben. In die tijd zal de DRAMA-machine 100.000 instructies uitvoeren, waarvan er slechts een 600-tal echt nuttig zijn (namelijk deze die in het voorbeeld op een donkere achtergrond afgedrukt zijn). De overige 99.400 instructies worden gewijd aan het uit-

lezen en het testen van de waarde van de toestandspoort en de voorwaardelijk sprong op het einde van de lus! Dus slechts 0,6% van de beschikbare computertijd wordt nuttig gebruikt. Hoe sneller de processor is, hoe inefficiënter hij gebruikt wordt: op een 10 MIPS processor zou 0,06% van de tijd nuttig gebruikt worden; op een 100 MIPS processor slechts 0,006%!

Invoer van het toetsenbord

Een gelijkaardig programma kan opgesteld worden voor het inlezen van een lijn die ingetypt wordt via het toetsenbord. We definiëren een lijn als een opeenvolging van tekens, afgesloten door het nieuwe-lijn symbool (d.i. de **`enter`**- of **`return`**-toets). Het toetsenbord heeft poort P_0 als toestandspoort en P_1 als gegevens/opdrachtpoort. Zie appendix D.2 (op pag. 222) voor een volledige beschrijving van dit invoerapparaat.

Er is wel een klein verschil met het programma uit voorbeeld 4-2: de processor hoeft hier **geen** opdracht te geven aan de bestuurder van het toetsenbord; de bestuurder zal zelf de ASCII-waarde van de aangeslagen toets in de gegevenspoort ter beschikking stellen. Van de processor wordt verwacht dat hij tijdig elk teken inleest (via **`INV`**).

Ook hier weer is synchronisatie tussen processor en bestuurder nodig: de processor mag alleen een teken inlezen, indien er een beschikbaar is. Dit wordt aangegeven door de **`GEGEVENS`**-toestand in de toestandspoort. Door continu deze toestand te testen, weet de processor wanneer een nieuw teken beschikbaar is. Het rendement van de processor is tijdens de uitvoering van dit programma heel laag. Het actief wachten neemt hier nog meer tijd in beslag. De ‘werksnelheid’ van het toetsenbord wordt immers bepaald door de snelheid waarmee de gebruiker de tekens intypt. Zie voorbeeld 4-3.

Voorbeeld 4-3. ***Invoer van het toetsenbord met actief wachten.***
Het volgende programma leest een aantal toetsaanslagen in (maximaal 100 of tot de return toets wordt ingelezen) en plaatst de ingelezen symbolen in een buffer op adres **`ZONE`***: één symbool per geheugenregister. Elk symbool wordt voorgesteld via zijn* ASCII*-code. Telkens de bestuurder een nieuw symbool ter beschikking heeft, zal de processor het inlezen uit poort* **`P1`***. We geven eerst het C-programma:*

```
#define PtT       0   /* toestandspoort */
#define PogT      1   /* opdracht/gegevenspoort */
#define RETURN    10  /* ascii code voor return-toets */
#define N         100 /* max aantal in te lezen symbolen */

int zone[N];
int index = -1;
int symbool = 0;

while ((index < N) && (symbool != RETURN))
{
    while (getPort(PtT) != GEG) { } /* sym. beschikbaar? */
    symbool = getPort (PogT);
    zone[++index] = symbool;
}
```

en hier volgt dan de DRAMA*-code:*

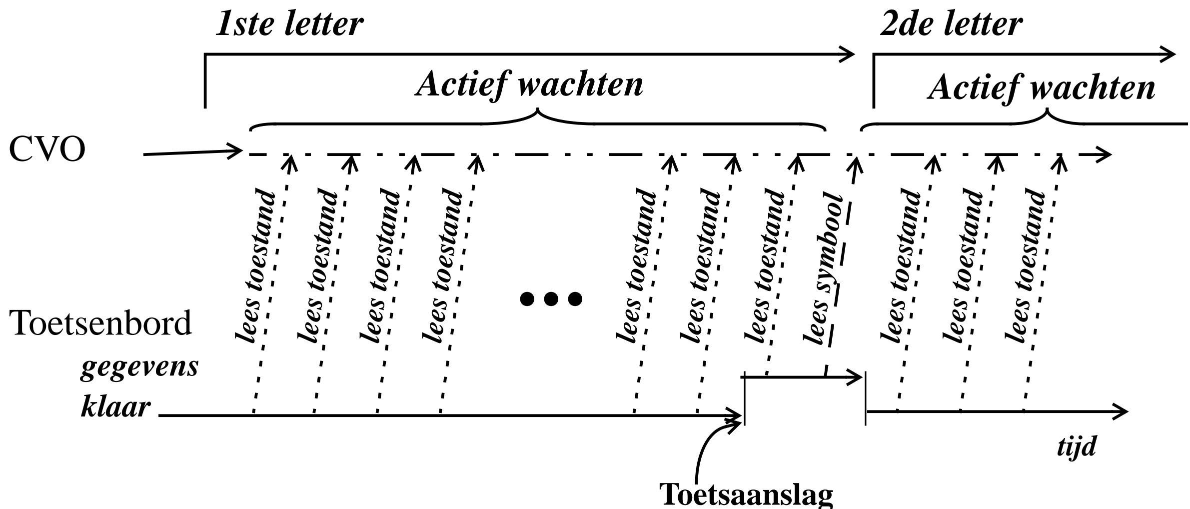
```
L1:  MEVA      PtT,0
L2:  MEVA      PogT,1
L3:  MEVA      GEG,2
L4:  MEVA      RETURN,10
L5:  MEVA      N,100
L6:            HIA.w   R6,-1            | index = -1
L7:            HIA.w   R2,0             | symbool = 0
L8:  WHILE:    VGL     R6,<N>
L9:            VSP     GRG,ENDWH        | al N symbolen ingelezen?
L10:           VGL     R2,<RETURN>      |
L11:           VSP     GEL,ENDWH        | volledige lijn gelezen?
L12:  WACHT:   INV     R1,<PtT>         |
L13:           VGL.w   R1,<GEG>         | GEGEVENS?
L14:           VSP     NGEL,WACHT       |
L15:           INV     R2,<PogT>        | Lees symbool
L16:           BIG     R2,ZONE(+R6)     | zone[++index] = symbool
L17:           SPR     WHILE
L18:  ENDWH:   ...
L16:   ZONE:   RESGR <N>
```

Het actief wachten gebeurt in lijnen *(L12)* tot *(L14)*.

Performantie-berekening

Een zeer snelle typist(e) die perfect weet wat hij/zij zal intypen kan in het beste geval een 5-tal tekens per seconde intypen. Dit betekent dat er gemiddeld 200 ms nodig zijn om één teken in te typen. Voor het inlezen zelf en het organiseren van een lus (om een volledig lijn in te lezen) zijn slechts een zevental instructies per teken nodig (zie instructies met een donkere achtergrond in het voorbeeld), wat op een 1 MIPS machine een zevental micro-seconden in beslag neemt. Bijgevolg zullen bijna 199,993 ms gespendeerd worden aan het actief wachten! Het rendement van de processor bedraagt dus in het beste geval 0,0035%!

Figuur 4-3 toont het tijdsdiagram voor deze invoerbewerking. De toestand van het toetsenbord wisselt tussen KLAAR (geen symbool beschikbaar) en GEGEVENS (nieuw symbool beschikbaar). Indien een symbool niet tijdig wordt ingelezen en een nieuwe toets is aangeslagen, zal de toestand in FOUT veranderen. Bij dit programma zou dit niet mogen voorkomen.



FIGUUR 4-24. Tijdsdiagram bij invoer met actief wachten.

(b) In- en uitvoer m.b.v. programma-onderbrekingen

Een oplossing voor het hierboven geschetste rendementsprobleem bestaat erin dat de bestuurder van het randapparaat de processor ***op de hoogte brengt*** wanneer hij klaar is met een opdracht of wanneer er gegevens beschikbaar zijn. De processor zelf hoeft dus niet voortdurend de toestand van de bestuurder te testen. Het mechanisme dat we hiervoor zullen gebruiken is de programma-onderbreking.

Bekijken we deze methode even vanuit het standpunt van de bestuurder van het apparaat: hij vraagt een PO aan telkens een tussenkomst van de processor gewenst is, met name:

- de bestuurder is klaar met de vorige opdracht, en zou dus graag ***een nieuwe opdracht*** krijgen van de processor;
- er heeft zich een fout voorgedaan, dus de processor moet zo snel mogelijk de bestuurder ***herinitialiseren***;
- er zijn gegevens beschikbaar, bijgevolg wordt de processor aangepord om deze zo snel mogelijk uit de gegevenspoort te ***lezen***.

De processor bepaalt dus nog altijd '**wat**' er gebeurt, maar de bestuurder geeft aan '**wanneer**' iets mag gebeuren. Synchronisatie door 'actief wachten' wordt dus overbodig.

Zoals we verder zullen zien wordt deze methode alleen gebruikt wanneer het ***gegevensdebiet*** (van/naar het apparaat) ***eerder laag*** is. De efficiëntie is reeds beter dan bij geprogrammeerde invoer/uitvoer, maar het afhandelen van programma-onderbrekingen vergt wel wat extra tijd (in het Engels noemt men dit '*overhead*').

Laten we deze methode toepassen op hetzelfde uitvoerapparaat: het scherm. De processor zal aan de bestuurder van het scherm een opdracht geven om een teken af te beelden (m.b.v. **UTV**) en daarna iets anders doen (bijvoorbeeld instructies van een gebruikersprogramma **PROG** uitvoeren; we komen hierop uitgebreid terug in paragraaf 4.6 (op pag. 175)). Als het scherm klaar is met de opdracht zal de bestuurder een PO-vlag plaatsen (namelijk deze die geassocieerd[1] is met dit randapparaat; in casu de PO_4-vlag). Pas wanneer de processor ingaat op deze PO[2] zal in de PO-routine (**PO_ROUT4**) de toestand van het scherm nagekeken worden ("Is alles OK?") en zal de opdracht voor de volgende letter gegeven worden.

Voorbeeld 4-4. ***Schrijven op het scherm met programma-onderbrekingen.***
*De eerste letter van de in de geheugenzone (**ZONE**) aanwezige tekst wordt via de vorige methode uitgeschreven. Zodra de opdracht aan de bestuurder gegeven is, zal de processor verder gaan met de uitvoering van het hoofdprogramma. Alle opdrachten voor het uitschrijven van de volgende letters worden in de PO-routine **PO_ROUT4** gegeven. Merk op dat wanneer de PO zich voordoet, de bestuurder ofwel 'klaar' is voor de volgende opdracht, ofwel zich in de 'fout' toestand bevindt (de processor zal zelf moeten nagaan welke van de twee mogelijke toestanden van toepassing is).*

We geven eerst het C-programma:

```
#define PtS   2   /* toestandspoort */
#define PogS  3   /* opdracht/gegevenspoort */

int aantal = 100; /* aantal uit te schrijven tekens */
int zone[100] = { (int) 'V', (int) 'o', ...};
int dru_opd = 1000000000;
int index = 0;    /* index in zone */

main ()
{
    /* eerste letter: zie vorige methode */
    int opdracht = dru_opd + zone[index++];
    while (getPort(PtS) != KLAAR) { }
    putPort (PogS, opdracht);

    bereken();
}

void bereken() {
    /* hier gebeuren berekeningen */
    ...
}
```

1. Deze associatie kan vaak met schakelaartjes ingesteld worden op de kaart waarop de bestuurder gemonteerd is; via een meer recente techniek kan ze ook software-matig worden ingesteld.

2. De PO-vlaggen worden op einde van elke bevelencyclus getest; de maskers en het huidig onderbrekingsniveau bepalen of de processor al dan niet op een PO kan ingaan. (Cfr. paragraaf 4.3 (op pag. 101).

```
interrupt po_rout4()
{
    if (getPort(PtS) == KLAAR)
    {
        if (index < aantal) {
            int opdracht = dru_opd + zone[index++];|
            putPort(PogS, opdracht);
        }
    } else {
        /* er is een FOUT opgetreden */
        ...
    }
}
```

en hier volgt dan de DRAMA-*code:*

```
MEVA       PtS,2
MEVA       PogS,3
MEVA       KLAAR,0
    MAIN:  HIA      R6,INDEX
           | Stel opdracht voor 1ste letter samen: 1000000ccc
           HIA      R5,DRU_OP
           OPT      R5,ZONE(R6+)
           BIG      R6,INDEX
   WACHT:  INV      R1,<PtS>        |
           VGL.w    R1,<KLAAR>      | KLAAR?
           VSP      NGEL,WACHT      |
           UTV      R5,<PogS>       | Schrijf eerste letter
           SBR      BEREKEN
           STP
 BEREKEN:  ...
           ...
           KTG

PO_ROUT4:  BIG      R0,BEW_PO4+0    | Bewaar inhoud gebruikte reg.
           BIG      R5,BEW_PO4+1    |
           BIG      R6,BEW_PO4+2    |
           INV      R0,<PtS>
           VGL.w    R0,<KLAAR>      | Toestand = KLAAR?
           VSP      NGEL,FOUT       | --> er is een fout gebeurd
           HIA      R6,INDEX
           VGL      R6,AANTAL       | Nog letters?
           VSP      GEL,KLAAR       | --> geen letters meer
```

```
           | Geef volgende opdracht
           HIA     R5,DRU_OP       |
           OPT     R5,ZONE(R6+)    |
           UTV     R5,<PogS>       | Schrijf letter ...
           BIG     R6,INDEX        | Pas index aan
   KLAAR:  HIA     R0,BEW_PO4+0    |
           HIA     R5,BEW_PO4+1    | Herstel inhoud registers
           HIA     R6,BEW_PO4+2    |
           KTO
    FOUT:  …
 BEW_PO4:  RESGR   3
 DRU_OPD:  1000000000              | 1000000ccc = Beeld ‘ccc’ af
   INDEX:  0
  AANTAL:  100
    ZONE:  0000000086              | Letter V
           0000000111              | Letter o
           …
```

Performantie-berekening

Door gebruik te maken van het programma-onderbrekingsmechanisme zijn er extra instructies nodig: o.a. deze voor het bewaren en herstellen van de inhouden van de gebruikte registers; ook het optreden van een programma-onderbreking en de terugkeer uit die onderbreking kost wat extra tijd. (Zie de instructies met een lichtgrijze achtergrond). Dit aantal is echter beperkt: per letter hebben we een 17 instructies nodig plus wat tijd om de PO te laten optreden (ongeveer de duur van 2 instructies om de PTW op de stapel te zetten en een nieuwe waarde in de BT te plaatsen), wat dus op een 1 MIPS machine in totaal ongeveer overeenkomt met 19 micro-seconden). De overige 981 micro-seconden kan de processor aanwenden om instructies van het hoofdprogramma (namelijk, de procedure **`bereken()`**) uit te voeren. Dus in de tijd nodig om de overige 99 letters uit te schrijven kan de processor ongeveer $99 \times 981 = 97.119$ bevelen van **`bereken()`** uitvoeren! De efficiëntie[1] is dus gestegen tot $\frac{(99 \times 6) + 97119}{99000} = 0.987$ of 98,7%.

Indien de snelheid van de processor toeneemt, zal de efficiëntie nog verbeteren: op een 10 MIPS machine bedraagt de efficiëntie al 99,8%. Met andere woorden, de extra overhead wordt minder belangrijk naarmate de processor sneller wordt.

1. Aangezien we niet weten of er voor de eerste letter actief wachten nodig zal zijn, laten we die buiten beschouwing. Als initieel het scherm klaar is voor een opdracht, dan zal het geven van de opdracht voor de eerste letter ook niet veel tijd vergen. De berekende efficiëntie zal in dat geval nog iets verbeteren, omdat voor de eerste letter minder overhead veroorzaakt wordt. Is het scherm echter nog bezig met het afdrukken van een letter, dan is het mogelijk dat de processor actief zal moeten wachten, wat de efficiëntie een klein beetje zal doen dalen.

Uit het voorgaande blijkt dus dat in- en uitvoer m.b.v. programma-onderbrekingen heel wat efficiënter is dan geprogrammeerde in- en uitvoer, omdat de CVE tijdens de wachtperiodes iets zinvol kan uitvoeren. Nochtans kan de tijd die nodig is voor het afhandelen van de onderbrekingsroutine (de 19 micro-seconden in het aangehaalde voorbeeld) in sommige omstandigheden een te grote rem betekenen. Snelle randapparaten, zoals schijven of cassette-eenheden, voeren de gegeven opdrachten zo snel uit, dat de processor geen tijd meer heeft om iets anders te doen. Bijvoorbeeld, een schijf met een debiet van 2 megabyte per seconde, zou in een halve micro-seconde een nieuw teken ter beschikking hebben voor de processor! Voor dergelijke randapparaten moeten we dus een andere methode uitwerken.

In- en uitvoer m.b.v. programma-onderbrekingen zal daarom alleen maar gebruikt worden wanneer het gegevensdebiet, van of naar het randapparaat, eerder laag is: bijvoorbeeld bij toetsenborden, schermen, drukkers, …

(c) Directe geheugentoegang

Directe geheugentoegang (DGT) (in het Engels '*direct memory access*' of DMA), is een optimisatie van het vorige schema. In plaats van de gevraagde gegevens één voor één te ontvangen van (of door te geven aan) de bestuurder van het randapparaat, zal de bestuurder deze gegevens zelf rechtstreeks in het geheugen plaatsen (of eruit ophalen). De processor hoeft slechts een opdracht te geven aan de bestuurder; van dan af zal laatstgenoemde die opdracht zelf verder afwerken ***zonder tussenkomst van de processor***. Pas op het einde van de volledige opdracht zal de bestuurder een programma-onderbreking aanvragen. Bij het vorige schema wordt een programma-onderbreking aangevraagd telkens een nieuw gegeven beschikbaar of verwerkt is. Er zullen dus beduidend minder programma-onderbrekingen optreden. Ook zal de processor niet meer moeten tussenkomen bij het gegevenstransport (van de bestuurder naar het geheugen en omgekeerd). Op deze wijze heeft de processor meer tijd beschikbaar voor het uitvoeren van allerhande programma's.

Laten we dit mechanisme toepassen op een schijvenbestuurder: stel dat er 1000 tekens van de schijf moeten gelezen worden. De processor zal aan de bestuurder de opdracht geven om die 1000 tekens in te lezen ***en ze zelf in het geheugen te plaatsen***. Pas wanneer die 1000 tekens in het geheugen zijn gezet, zal de bestuurder de processor hiervan d.m.v. een PO-aanvraag op de hoogte brengen.

Veronderstel dat we de DRAMA-machine met een harde schijveneenheid uitrusten:

- de schijf bestaat uit 300 cilinders,
- elke cilinder heeft 40 sporen (er zijn dus 40 lees/schrijfkoppen);
  in totaal heeft deze schijveneenheid dus 300 * 40 = 12000 sporen,
- elk spoor bevat 50 sectoren,
- elke sector bestaat uit 100 getallen van 10 cijfers (die in één keer gelezen of geschreven worden), de bestuurder heeft een interne buffer die groot genoeg is om één sector te bevatten,
- de toestandspoort van de schijfbestuurder is **`P6`**, terwijl **`P7`** de opdracht/gegevenspoort is.

Indien we alfanumerische informatie willen voorstellen hebben we (op DRAMA[1]) voor elk teken 3 cijferposities nodig, zodat elke sector 300 tekens kan bevatten. Reken zelf na dat de schijf een capaciteit heeft van 180 miljoen tekens[2]. Indien we dus 1000 tekens willen inlezen, zullen we 4 opeenvolgende sectoren moeten lezen. Van de laatste sector hebben we slechts een gedeelte nodig, maar een sector is de kleinste eenheid die in één keer moet gelezen worden.

De DRAMA-schijfbestuurder kent zes soorten bevelen: initialisatie, formatteren van de schijf, het positioneren van de kam met de lees-/schrijfkoppen, het lezen of schrijven van een aantal sectoren, het doorgeven van een geheugenadres (waar de ingelezen gegevens moeten geplaatst worden, of waar de uit te schrijven gegevens kunnen gevonden worden). Voor een uitgebreide beschrijving van de DRAMA-schijfbestuurder: zie paragraaf D.3 (op pag. 224).

TABEL 4-6. Enkele opdrachten voor de schijfbestuurder.

Opdracht	Betekenis
…	…
`100ttt0ccc`	Positioneer de kam op cilinder `ccc` en activeer lees/schrijfkop `ttt`; de volledige opdracht bepaalt het te gebruiken spoor.
`200lll0sss`	Lees `lll` opeenvolgende sectoren vanaf sector `sss.`
`300lll0sss`	Schrijf `lll` opeenvolgende sectoren vanaf sector `sss.`
`500000gggg`	‘`gggg`’ is het adres van de DGT-geheugenzone.
…	…

De leesoperatie moet in twee stappen uitgevoerd worden (zie ook hoofdstuk 2):

- Eerst moet de kam waarop de lees/schrijfkoppen gemonteerd zijn verplaatst worden naar de correcte positie en moet één van de koppen geactiveerd worden. Alle sporen die zich voor een bepaalde stand van de kam onder een lees/schrijfkop bevinden behoren tot eenzelfde cilinder, dus kan men ook zeggen dat de kam naar de juiste cilinder moet verplaatst worden[3].
  Het verplaatsen van de kop kan heel wat tijd vergen (gemiddeld 10-15 ms).
- Daarna kan de eigenlijke leesopdracht gegeven worden. Hiervoor zijn twee bevelen nodig:
  - Eerst moet de schijfbestuurder op de hoogte gebracht worden van waar hij in het geheugen de gegevens moet zetten.
  - Daarna kan de echte transport-opdracht gegeven worden.

De code hiervoor vind je in voorbeeld 4-5 (op pag. 143). In dit voorbeeld wordt actief wachten vermeden. Na het geven van de positioneer-opdracht aan de schijfbestuurder zal de processor iets anders doen. Als de schijfbestuurder de kam boven het juiste spoor heeft gebracht, vraagt hij een programma-onderbreking aan. De PO-routine zal dan het tweede deel van de lees-operatie opstarten, namelijk de eigenlijke leesopdracht. De schijfbestuurder zal ook hier heel wat tijd

1. Op een reële (binaire) computer volstaat één byte.
2. Het eerste cijfer van elk woord wordt niet gebruikt, of zou een pariteitscijfer kunnen zijn.
3. Laten we voor de eenvoud veronderstellen dat de vier opeenvolgende sectoren in hetzelfde spoor gelegen zijn, zodat we met één leesopdracht deze vier sectoren kunnen inlezen.

voor nodig hebben, hij zal echter de data rechtstreeks in het geheugen plaatsen. Ondertussen kan de processor weer iets anders gaan doen. Als de gegevens volledig in het geheugen staan, zal de schijfbestuurder een tweede keer een programma-onderbreking aanvragen. Nu moet er niets gebeuren. De PO-routine keert na controle van de toestand van de bestuurder, gewoon terug. Bestudeer de code aandachtig. Je vindt wat extra uitleg bij de DRAMA-code.

Voorbeeld 4-5. ***Lezen van magneetschijf m.b.v. directe geheugen toegang.***
In het volgende programma worden 4 opeenvolgende sectoren ingelezen van schijf. De in te lezen sectoren bevinden zich op cilinder 37, spoor 13, en beginnen op sector 4. Het inlezen gebeurt in twee stappen: (1) eerst wordt de kam boven de juiste cilinder gepositioneerd en wordt de juiste lees/schrijfkop geselecteerd en (2) daarna wordt het gegevenstransport gestart. De schijfbestuurder zal zelf de ingelezen gegevens in het geheugen plaatsen (op een adres dat door het programma wordt bepaald). De toestandspoort van de schijvenbestuurder is `P6` *en de gegevens/opdrachtpoort* `P7`*.*
We geven eerst het C-programma:

```
#define PtMS         6  /* Toestandspoort */
#define PogMS        7  /* Opdracht/gegevenspoort */
#define WERKLOOS     0  /* Bestuurder is werkloos */
#define VERPLAATSEN  1  /* De kam wordt verplaatst */
#define TRANSPORT    2  /* Gegevens worden getransporteerd */
#define LEZEN        1  /* Gevraagde actie is lezen */
#define SCHRIJVEN    2  /* Gevraagde actie is schrijven */

int buffer [4 * 100];
main() {
    ms_operatie (LEZEN, 37, 13, 7, 4, buffer);
    bereken(); /* doe berekeningen */
    while (! ms_klaar()) { } /* actief wachten indien nodig */
    wijzig_gegevens(); /* wijzig ingelezen gegevens */
    ms_operatie (SCHRIJVEN, 56, 4, 2, 4, buffer);
    bereken2(); /* doe berekeningen */
}
void bereken() { ... }
void wijzig_gegevens() { ...}
void bereken2() { ...}

int pos_opd =     1000000000;  /* positioneringsopdracht */
int dgt_opd =     5000000000;  /* doorgeven van dgt adres */
int lees_opd =    2000000000;  /* leesopdracht */
int schrijf_opd = 3000000000;  /* schrijfopdracht */

struct opdracht {
    int actie;        /* LEZEN of SCHRIJVEN */
    int cilinder, spoor, sector;
    int aantal;
    int dgt_adres;
} huidig;                    /* uit te voeren i/o opdracht */
int modus = WERKLOOS;        /* schijf bezig met ... */
```

```
void ms_operatie (int actie,
                  int cilinder, int spoor, int sector,
                  int aantal,
                  int * dgt_adres)
{
    if (modus != WERKLOOS) return; /*andere aanvraag bezig*/

    huidig.actie = actie;
    huidig.cilinder = cilinder;
    huidig.spoor = spoor;
    huidig.sector = sector;
    huidig.aantal = aantal;
    huidig.dgt_adres = (int) dgt_adres;

    positioneer_kam();
}

int ms_klaar()
{
    /* gaat na of de laatste opdracht afgewerkt is */
    return modus == WERKLOOS;
}

void positioneer_kam()
{
    int opdracht = pos_opd + huidig.spoor * 10000 +
                    huidig.cilinder;
    modus = VERPLAATSEN;
    if (getPort(PtMS) != KLAAR) { /* FOUT */ ... }
    putPort(PogMS, opdracht);
}

void start_transport() /* wordt opgeroepen vanuit po_rout6 */
{
    /* (a) eerst de DGT opdracht */
    int opdracht = dgt_opd + huidig.adres;
    putPort (PogMS, opdracht);

    /* (b) daarna de lees of schrijf opdracht */
    opdracht = huidig.aantal * 10000 + huidig.sector;
    if (huidig.actie == LEZEN) opdracht += lees_opd;
    else opdracht += schrijf_opd;
    modus = TRANSPORT;
    if (getPort(PtMS) != KLAAR) { /* FOUT */ ...}
    putPort (PogMS, opdracht);
}
```

```
interrupt po_rout6()
{
    int vorige_modus = modus; /* onthoud vorige modus */
    modus = WERKLOOS;
    if (getPort(PtMS) == KLAAR) {
        if (vorige_modus == VERPLAATSEN)
            start_transport();
        /* else: opdracht voltooid! */
    } else {
        /* Er is een FOUT gebeurd */
        ...
    }
}
```

Hier volgt de DRAMA-code voor de constanten en gegevensstructuren:,

```
         MEVA  PtMS,6
         MEVA  PogMS,7
         MEVA  WERKLOOS,0
         MEVA  VERPLAATSEN,1
         MEVA  TRANSPORT,2
         MEVA  LEZEN,1
         MEVA  SCHRIJVEN,2
     TIENDZD:  10000
     POS_OPD:  1000000000          | 'Positioneer' opdracht
     DGT_OPD:  5000000000          | 'DGT' opdracht
    LEES_OPD:  2000000000          | 'Lees' opdracht
 SCHRIJF_OPD:  3000000000          | 'Schrijf' opdracht
      HUIDIG:  RESGR 6
       MODUS:  <WERKLOOS>
```

De DRAMA-code voor positioneer_kam. Eerst wordt de opdracht samengesteld, de huidige modus wordt gewijzigd (wordt nu VERPLAATSEN), de toestand van de bestuurder wordt gecontroleerd (zou KLAAR moeten zijn; indien niet, is er een fout gebeurd), en tenslotte wordt de opdracht in de opdrachtpoort geschreven. De modus hebben we nodig omdat de po-routine moet kunnen nagaan of de transport-operatie al dan niet reeds uitgevoerd is.

```
     POS_KAM:  BST    R0
               BST    R1
               HIA    R1,HUIDIG+2    || Stel positioneer-
               VER    R1,TIENDZD     || opdracht op
               OPT    R1,HUIDIG+1    ||
               OPT    R1,POS_OPD     ||
```

```
              HIA.w  R0,<VERPLAATSEN>
              BIG    R0,MODUS
              INV    R0,<PtMS>      | Schijf is KLAAR?
              VGL.w  R0,<KLAAR>
              VSP    NGEL,FOUT1
              UTV    R1,<PogMS>     | Geef opdracht
              HST    R1
              HST    R0
              KTG
      FOUT1:  ...
```

De DRAMA-code voor start_transport. Eerst wordt het DGT-adres doorgegeven. Deze opdracht wordt ogenblikkelijk verwerkt door de bestuurder. Daarna wordt de transport-opdracht (lezen of schrijven) doorgegeven aan de bestuurder. De opdrachten met licht-grijze achtergrond zijn bedoeld om de code robuust te maken. Normaal zou de bestuurder in de toestand KLAAR moeten zijn, tenzij er een onverwachte fout is gebeurd.

```
START_TRANS:  BST    R0
              BST    R1
              HIA    R1,HUIDIG+5      || Stel dgt-adres
              OPT    R1,DGT_OPD       || opdracht op
              UTV    R1,<PogMS>       | Geef opdracht
              | Normaal is deze opdracht direct verwerkt
              HIA    R1,HUIDIG+4      ||
              VER    TIENDZD          ||
              OPT    R1,HUIDIG+3      ||
              HIA    R0,HUIDIG+0      || Stel transport
              VGL.w  R0,<LEZEN>       || opdracht op
              VSP    NGEL,SCHRIJF     ||
       LEES:  OPT    R1,LEES_OPD      ||
              SPR    VERDER           ||
    SCHRIJF:  OPT    R1,SCHRIJF_OPD   ||
     VERDER:  INV    R0,<PtMS>        | Schijf is KLAAR?
              VGL.w  R0,<KLAAR>       |
              VSP    NGEL,FOUT2       | neen -> fout!
              UTV    R1,<PogMS>       | Geef opdracht
              HST    R1
              HST    R0
              KTG
      FOUT2:  ...
```

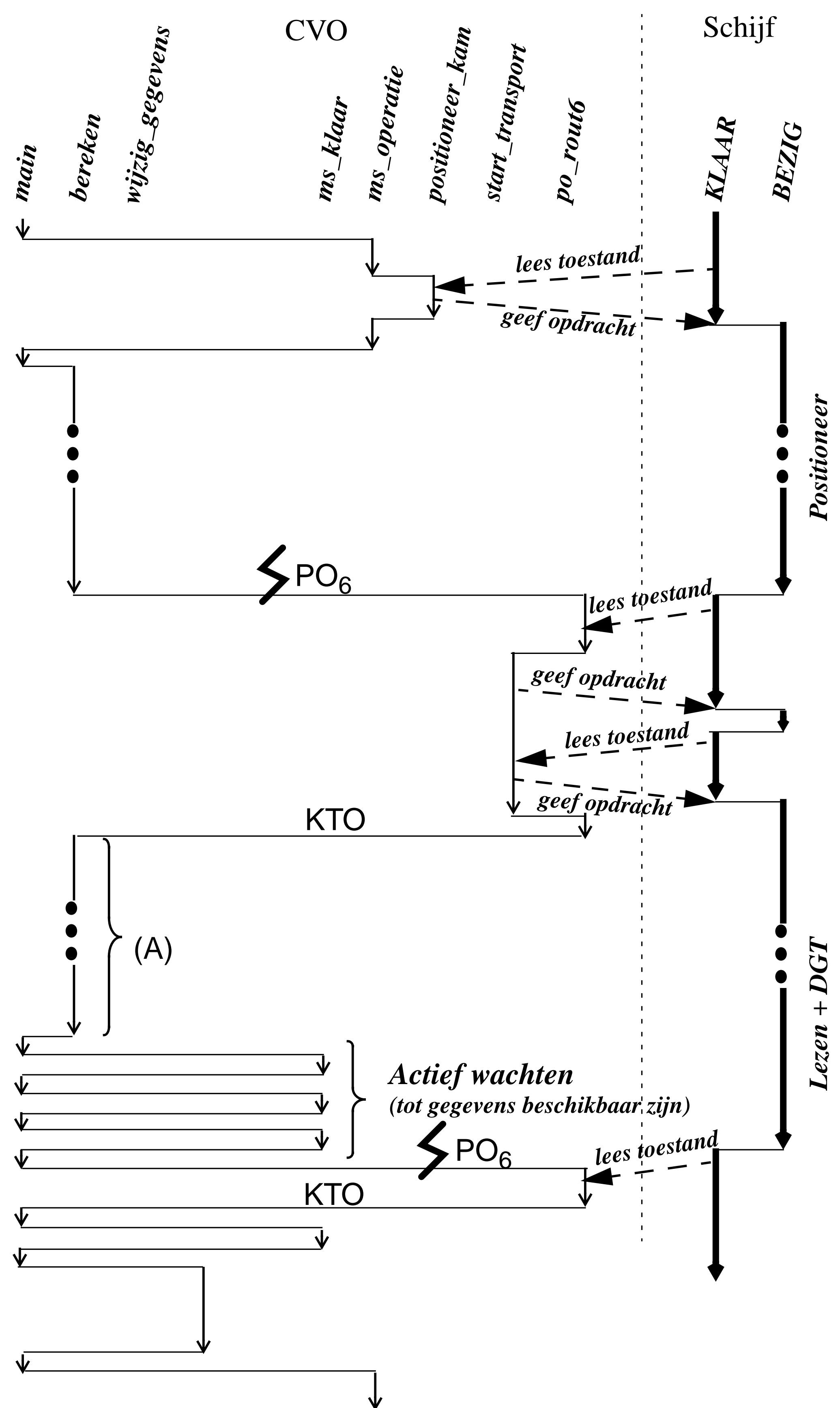
En tenslotte, de DRAMA-code voor po_rout6. De po-routine bewaart eerst de vorige waarde van modus en zet deze terug op WERKLOOS. Daarna gaat de routine na of de vorige opdracht correct is uitgevoerd (de toestand van de schijfbestuurder zou KLAAR moeten zijn), zoniet is er een fout gebeurd en moeten de nodige herstelmaatregelen getroffen worden (deze code is niet gegeven). Indien de vorige modus VERPLAATSEN was, dan moet nu het transport opgestart worden (zie procedure-oproep van start_transport). Als de vorige modus echter TRANSPORT was, dan hoeft er niets meer te gebeuren en keert de po-routine gewoon terug. De vorige invoer/uitvoer-opdracht is volledig afgewerkt.

```
  PO_ROUT6: BIG    R0,BEWAAR_PO6+0
            BIG    R1,BEWAAR_PO6+1
            HIA    R1,MODUS           | R1 = vorige modus
            HIA.w  R0,<WERKLOOS>
            BIG    R0,MODUS           | modus = WERKLOOS
            INV    R0,<PtMS>          | Schijf is KLAAR?
            VGL.w  R0,<KLAAR>
            VSP    NGEL,FOUT3
            VGL.w  R1,<VERPLAATSEN>   | vorige modus?
            VSP    NGEL,EINDE         |
            SBR    START_TRANS        ||start_transport()
     EINDE: HIA    R0,BEWAAR_PO6+0
            HIA    R1,BEWAAR_PO6+1
            KTO
BEWAAR_PO6: RESGR  2                  | plaats voor acc.
```

De code voor ms_operatie, ms_klaar en het hoofdprogramma (main) laten we over aan de lezer. Merk wel op dat in het hoofdprogramma er mogelijk actief gewacht wordt: namelijk vóór de gegevens mogen gebruikt worden (in **`wijzig_gegevens`***), moet het hoofdprogramma nagaan of de leesopdracht reeds voltooid is; mocht dit niet het geval zijn, dan zal er actief gewacht moeten worden. Na het wijzigen van de gegevens worden ze opnieuw weggeschreven op een andere plaats op schijf. Ook hier weer wordt er niet gewacht tot de gegevens op schijf staan maar kan de processor iets anders gaan doen:* **`bereken2()`**.

Figuur 4-5 (op pag. 143) toont het tijdsdiagram[1] voor de uitvoering van het hoofdprogramma (het schrijven van de gegevens is niet meer weergegeven in het diagram). Zoals je kan opmerken zijn zowel randapparaat als processor bezig met het uitvoeren van zinvolle taken (**`bereken`**() en positioneren en inlezen van sectoren). Doordat de schijfbestuurder nu directe geheugentoegang heeft, wordt er geen programma-onderbreking aangevraagd voor elk gegeven dat beschikbaar is of weggeschreven is, maar zal de schijfbestuurder zelf de gegevens in het geheugen plaatsen of uit het geheugen halen. Aangezien het in dit voorbeeld langer duurt om de gegevens in te lezen dan om de code van **`bereken`***()* uit te voeren, zal in het hoofdprogramma gedurende een zekere tijd actief wachten moeten toegepast worden.

1. De tijd loopt van boven naar beneden. Het diagram is niet op schaal getekend (zie de drie puntjes in de figuur).



FIGUUR 4-25. Tijdsdiagram bij lezen van schijf met DGT.

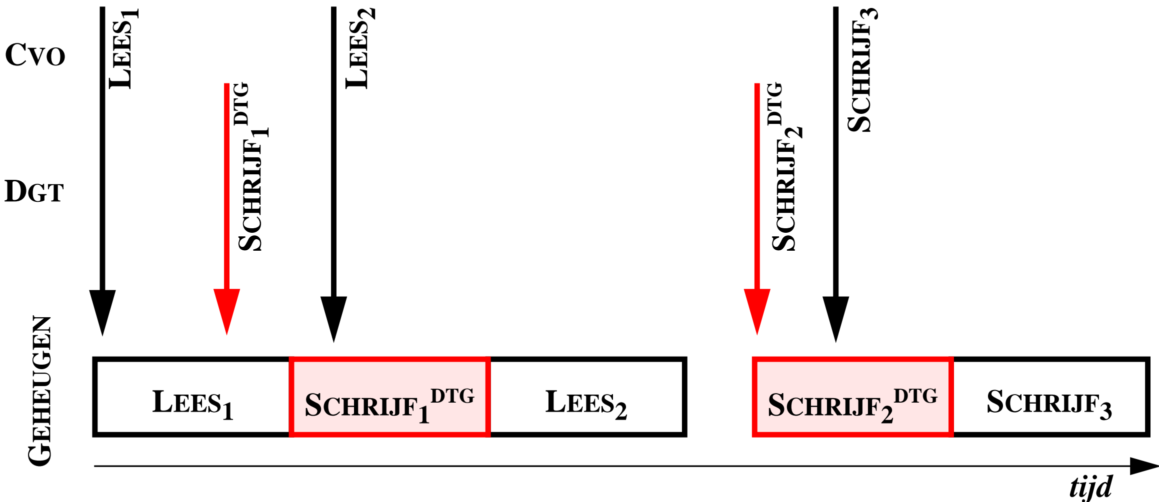
Zonder DGT zou de processor voor elke sector 100 **INV** bewerkingen moeten uitvoeren, om elk van de 100 getallen waaruit een sector bestaat in te lezen; bovendien moet vóór elke **INV**-bewerking op de gegevenspoort eerst nagekeken worden (in de toestandspoort) of de nieuwe gegevens reeds beschikbaar zijn. Dankzij DGT zal de processor nu slechts onderbroken worden (via een PO) als de vier sectoren volledig ingelezen zijn in het geheugen. Gedurende al die tijd kan hij zich met andere taken bezig houden (zie (A) in de fig. 4-25).

De tijdswinst hangt voor een groot deel af van het type schijf. De volgende factoren zijn bepalend:

- de **rotatiewachttijd** (dit is de tijd die gewacht moet worden tot de gewenste sector onder de lees/schrijfkop komt; gemiddeld genomen zal die sector immers een halve toer verwijderd zijn van de kop; dus hoe sneller de schijf ronddraait, hoe kleiner deze wachttijd),
- het **debiet** (de snelheid waarmee de gegevens kunnen afgeleverd worden; dit hangt af van de rotatiesnelheid van de schijf en van de afstand tussen de bits, d.i. de informatiedichtheid).

Aangezien elk van de gegevens door de schijfbestuurder in het geheugen moet geplaatst worden zal de tijd nodig voor de DGT minstens 400 geheugencycli zijn, hoe snel de schijf ook moge zijn.

Op de DRAMA-machine (en op elke machine met een bus-architectuur) wordt dit minimum niet bereikt, want niet alleen de schijfbestuurder vraagt toegang tot het geheugen, maar ook de processor (voor het ophalen van instructies, en het ophalen en wegbergen van gegevens tijdens de uitvoering van **bereken()**). Zie figuur 4-26

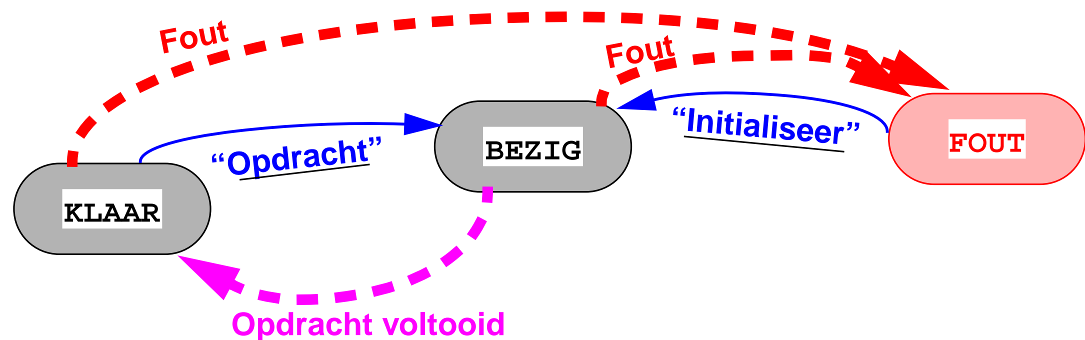


FIGUUR 4-26. De bestuurder (via dgt) steelt geheugencycli van de processor.

Aangezien de ‘bus’ maar door één apparaat tegelijk kan gebruikt worden en het geheugen meestal maar één tegelijk kan bedienen, zullen processor en schijfbestuurder soms extra lang moeten wachten alvorens hun lees- of schrijf-operatie in het geheugen uitgevoerd is. De schijfbestuurder (zoals elke bestuurder met DGT) **steelt geheugencycli** van de processor. (In het Engels spreekt men over ‘*cycle stealing*’.) De verticale pijlen tonen wanneer een geheugentoegang gevraagd wordt. De rechthoekjes geven aan wanneer een aanvraag door het geheugen behandeld wordt.

Opmerkingen:

- Bij grotere computerinstallaties zal men soms de bestuurder van een snel randapparaat via een apart gegevenspad met het geheugen verbinden. Vooral bij hoge-orde spreiding (cfr. hoofdstuk 2) kan de processor ervoor zorgen dat de geheugen-zone die door de bestuurder gebruikt wordt bij de DGT in een geheugen-module ligt die niet door de processor zelf gebruikt wordt (voor het ophalen van de instructies en het ophalen/wegbergen van de operanden). Bijgevolg zal de cyclus diefstal tot een minimum (in het beste geval tot nul) herleid worden.
- Een bestuurder met directe gegevenstoegang heeft geen **GEGEVENS**-toestand. Immers de bestuurder plaatst zelf de gegevens rechtstreeks in het geheugen. Dit heeft voor gevolg dat het toestandsdiagram zoals dit afgebeeld is in figuur 4-22 (pag. 128) heel wat eenvoudiger wordt: ofwel is de bestuurder **KLAAR** om een opdracht te ontvangen, ofwel is hij **BEZIG** met de uitvoering van een opdracht en indien zich een fout voordoet, komt hij in de **FOUT**-toestand. Zie figuur 4-27



FIGUUR 4-27. Het toestandsdiagram voor een bestuurder die directe geheugentoegang gebruikt.

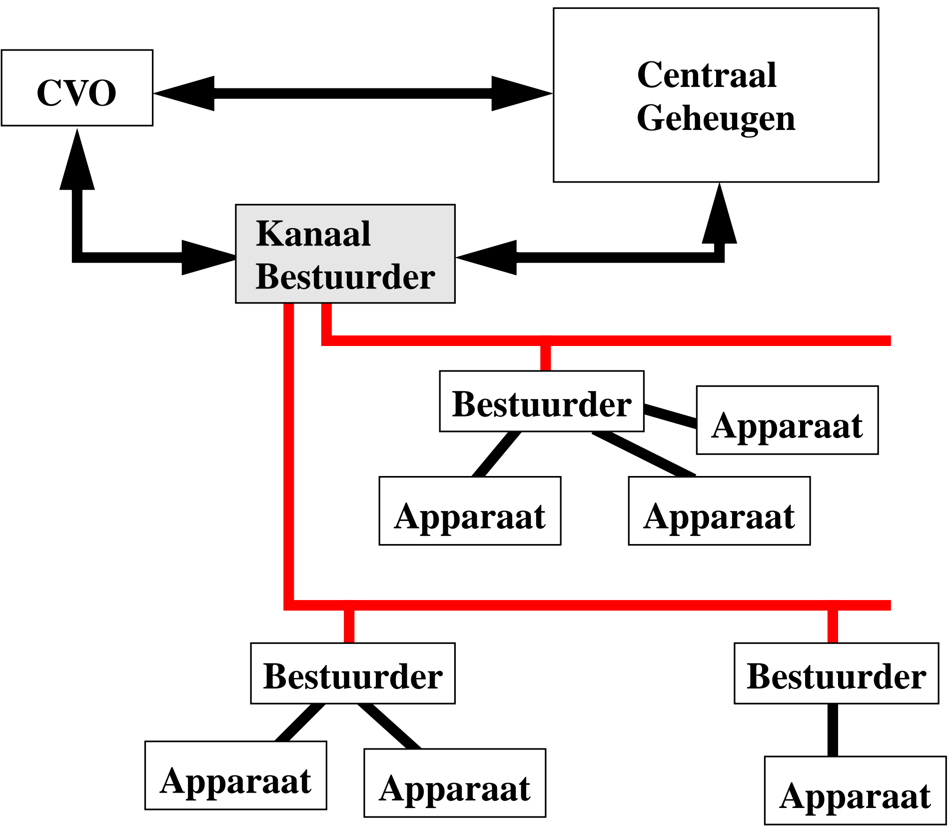
(d) Kanaalbestuurders

Voor grote computers, die heel wat randapparaten hebben, kan de invoer en uitvoer toch nog redelijk wat tijd vergen van de processor. Daarom worden speciale **invoer/uitvoer processoren** of kanaalbestuurders voorzien, die een aantal taken van de gewone processor overnemen. In plaats van zich persoonlijk met elk randapparaat bezig te houden, zal de processor een 'lijst met invoer/uitvoeropdrachten' aan de kanaalbestuurder geven. Deze laatste zal deze lijst met opdrachten volledig afwerken, en pas daarna de processor onderbreken.

Men kan de kanaalbestuurder opvatten als een soort DGT-bestuurder die uitgegroeid is tot een 'gespecialiseerde computer' met een eigen verwerkingsorgaan, een eigen geheugen en een eigen stel bevelen, ook wel kanaalopdrachten genoemd. Het CVO zal een kanaalprogramma opstellen (bestaande uit een opeenvolging van kanaalopdrachten) en dit doorgeven aan de kanaalbestuurder. Deze laatste zal dit programma volledig uitvoeren zonder dat het CVO hiervan hinder ondervindt. Dit houdt onder meer in:

- het geven van besturingsopdrachten aan de besturingseenheid waarvoor ze bedoeld zijn;
- het transporteren van de gegevens van en naar het centrale geheugen;
- het uitvoeren van testen op de correctheid van de gegevens (pariteitstesten of andere testen);
- inlichtingen doorgeven aan het CVO i.v.m. het al dan niet beschikbaar zijn van randapparaten, het correct functioneren van de bestuurders van deze apparaten, het beëindigen van een kanaalprogramma, enz. Dit gebeurt d.m.v. programma-onderbrekingen.

Figuur 4-28 schetst een mainframe met een kanaalbestuurder. Merk op dat zowel het CVO als de kanaalbestuurder een directe verbinding met het geheugen hebben (i.p.v. een busverbinding). Hierdoor wordt de frequentie van cyclusdiefstal drastisch gereduceerd, althans wanneer de kanaalbestuurder een ander deel van geheugen gebruikt dan de processor.



FIGUUR 4-28. Processor met kanaalbestuurder.

Een voorbeeld: De kanaalbestuurder van de IBM 3090-computer beslaat een bord van zes modules, in totaal ongeveer 700 chips, met gemiddeld 500 logische poorten of 16 kbit geheugen per chip. De bestuurder kan 24 kanalen bedienen.

Kanaalopdrachten

Kanaalopdrachten verschillen nogal van gewone machinebevelen: ze zijn meestal zeer lang omdat ze veel argumenten hebben. Bijvoorbeeld, een kanaalbestuurder die acht schijven bestuurt, zou het volgende kanaalprogramma kunnen krijgen van de processor: (links staat de

symbolische voorstelling, rechts de decimale voorstelling; de betekenis van de argumenten is symbolisch weergegeven via de notatie '**`ARGUMENT=`**'; merk op dat sommige kanaalbevelen (**`LEES`** en **`SCHRIJF`**) twee opeenvolgende geheugenregisters in beslag nemen op de DRAMA-machine). Het kanaalprogramma bevat twee positioneringsopdrachten (**`POS_KAM`**) en een lees-, schrijf- en stop-opdracht:

Kanaal bronprogramma (Symbolisch)		**Kanaalprogr. (Decimaal)**
POS_KAM	BESTUURDER=7, SCHIJF=0, CIL=13, SPOOR=37	6700370013
POS_KAM	BESTUURDER=8, SCHIJF=1, CIL=200, SPOOR=18	6810180200
LEES	BESTUURDER=7, SCHIJF=0, DGT-ADRES=6000,\	3700006000
	LENGTE=4, SECTOR=7	9000040007
SCHRIJF	BESTUURDER=8, SCHIJF=1, DGT-ADRES=8000,\	4810008000
	LENGTE=2, SECTOR=24	9000020024
STOP		9999999999

FIGUUR 4-29. Een kanaalprogramma.

In elke opdracht staat een aanduiding van het randapparaat waarop deze opdracht van toepassing is. Aangezien een bestuurder meerdere randapparaten kan sturen, moet zowel de **`BESTUURDER`** als de **`SCHIJF`** aangeduid worden.

Op de DRAMA-machine is geen kanaalbestuurder aanwezig. Indien er toch een was (met opdrachtpoort **`P9`**, en als enige bevel **`200000gggg`** waarbij het geheugenadres (**`gggg`**) van het kanaalprogramma wordt doorgegeven), zou het programma eruit zien zoals in het voorbeeld 4-6.

Voorbeeld 4-6. ***Invoer/uitvoer met behulp van een kanaalbestuurder.***

De processor zal het kanaalprogramma vooraf opstellen in de geheugenzone die start op adres **`kanaal_prog`**, *en daarna aan de kanaalbestuurder het adres van deze geheugenzone doorgeven via poort* **`P9`**. *De toestandspoort is* **`P8`**. *We geven eerst de C-code:*

```
#define PtK          8  /* Toestandspoort */
#define PogK         9  /* Opdracht/gegevenspoort */
int kanaal_prog [100];
int kan_opd = 2000000000;

main()
{
    maak_kanaalprog(); /* stel het kanaalprogramma op */
    int opdracht = kan_opd + (int) &kanaal_prog;
    if (getPort(PtK) == KLAAR) putPort (PogK, opdracht);
    bereken(); /* doe ondertussen berekeningen */
}
void bereken() { ... }
void maak_kanaalprog() { ...}
```

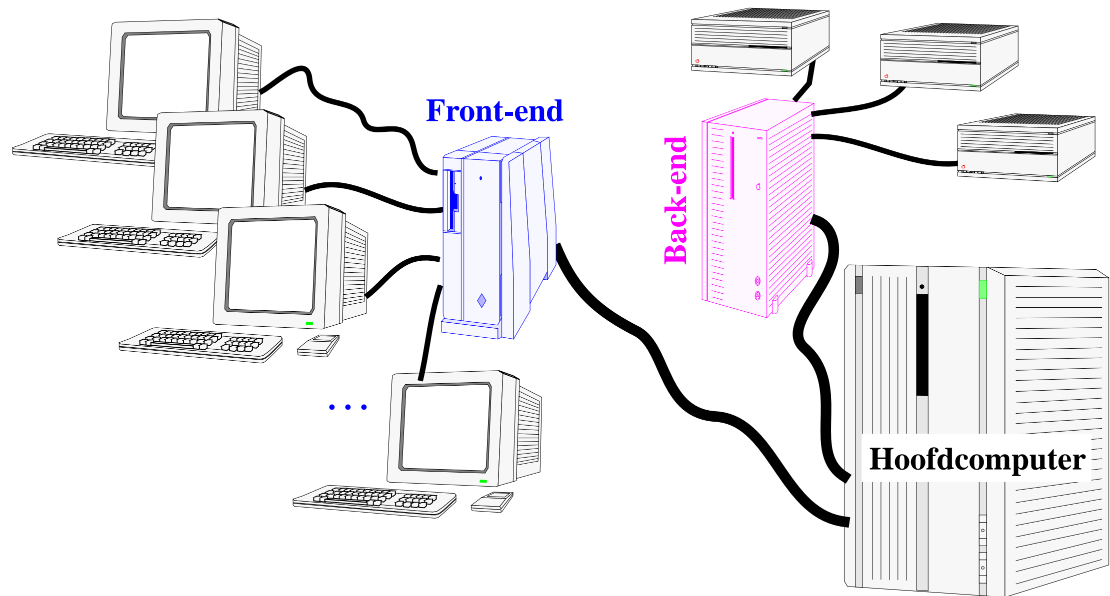
Hier volgt de drama-code:

```
          MEVA  PtK,8
          MEVA  PogK,9
         MAIN:  HIA.w  R8,0
                BST    R8
                HIA    R8,R9
                SBR    MAAK_KANAALPR    | Stel kanaal prog. op
                HST    R8
                HIA.a  R6,KANAAL_PROG   || Maak opdracht
                OPT    R6,KAN_OPD       ||
                INV    R0,<PtK>         |
                VGL.w  R0,<KLAAR>       | Kanaalbest. klaar?
                VSP    NGEL,N_KLAAR     |
                UTV    R6,<PogK>        || Geef opdracht
      N_KLAAR:  BST    R8
                HIA    R8,R9
                SBR    BEREKEN
                HST    R8
                STP
         FOUT:  ...
      BEREKEN:  ...
MAAK_KANAALPR:  ...
      KAN_OPD:  2000000000
  KANAAL_PROG:  RESGR 100
                ...
```

De kanaalbestuurder zal de kanaalbevelen uit het geheugen halen en ze uitvoeren (d.w.z. ze omzetten naar de elementaire bevelen voor die randapparaten). Hij staat ook in voor de DGT. Bovendien zal hij proberen de fouten die zich voordoen op te lossen. Bijvoorbeeld, bij een leesfout kan hij de opdracht herhalen.

(e) Satelliet-computers

In plaats van kanaalbestuurders te voorzien (wat heel gespecialiseerde in- en uitvoerprocessoren zijn), kan men ook ‘gewone’ computers gebruiken om de hoofdprocessor te ontlasten. In dit geval spreekt men over **satelliet-computers**, of meer specifiek over **front-end computers** of **back-end computers**. ‘Front-end’ computers houden zich bezig met de in- of uitvoerapparaten of met communicatie-kanalen naar andere computers; men spreekt over ‘back-end’ computers wanneer ze andere gespecialiseerde taken vervullen, zoals het beheer van de hulpgeheugens of een gegevensbank (die bewaard wordt op een uitgebreide verzameling schijven, magneetbanden en andere hulpgeheugens). Zie figuur 4-30:



FIGUUR 4-30. Een grote computerinstallatie met front-end en back-end computers.

In dit voorbeeld is de front-end computer verbonden met een duizendtal terminals, en verzorgt de invoer van de toetsenborden en de uitvoer naar de schermen. Telkens wanneer een volledige lijn (afgesloten door het `return`-teken) beschikbaar is, zal de front-end computer deze lijn doorgeven aan de hoofdcomputer. Het inlezen van de individuele toetsaanslagen, het lokaal editeren (d.w.z. wissen van de laatste toetsaanslag, wissen van de hele lijn, …), het op het scherm tonen wat ingetypt is (ook wel de ‘echo’ genoemd), enz. wordt allemaal door de satelliet-computer[1] verricht. Dit drukt aanzienlijk het aantal programma-onderbrekingen voor de hoofdcomputer, alsook de tijd die de hoofdcomputer aan invoer/uitvoer moet besteden, zodat er meer tijd overblijft voor het echte ‘rekenwerk’.

In de figuur zie je ook een ‘back-end’ computer die verbonden is met heel wat schijven. Deze ‘back-end’ houdt zich bezig met het beheer van de informatie op de hulpgeheugens.

1. Een satelliet-computer bestuurt de randapparaten via een van de methodes die we eerder geschetst hebben: via actief wachten, m.b.v. programma-onderbrekingen of m.b.v. DGT. In zeer grote installaties is het zelfs mogelijk dat de satelliet-computer gebruik maakt van een aantal kanaalbestuurders.

Een satelliet-computer hoeft niet heel snel te zijn; immers de randapparaten die hij bestuurt zijn relatief traag t.o.v. de snelheid van de processor. Een kleine computer (met eigen processor en primair geheugen) volstaat dus.

Het gebruik van een gewone computer i.p.v. een kanaalbestuurder biedt duidelijk een aantal **voordelen**:

- Kanaalbestuurders zijn gespecialiseerd naar een bepaald soort randapparatuur; satelliet-computers zijn in staat om een grotere variëteit van randapparaten te besturen.
- Een satelliet-computer kan ook voor gewone berekeningen gebruikt worden; een kanaalbestuurder zal alleen invoer/uitvoer verzorgen.
- Het is eenvoudiger om de programmatuur van een satelliet-computer te vervangen, dan deze van een kanaalbestuurder.
- Bij een satelliet-computer ben je niet afhankelijk van een fabrikant. Elke computer (met voldoende aansluitmogelijkheden voor randapparatuur) kan gebruikt worden als satelliet-computer.

Het enige **nadeel**[1] dat deze organisatie met zich meebrengt is dat de betrouwbaarheid van de computerinstallatie er meestal niet op verbetert. Immers, de kans dat de computerinstallatie onbruikbaar wordt is nu groter geworden:

$$K^d(\textit{installatie}) = K^d(\textit{hoofdcomputer}) + \Sigma\, K^d(\textit{satelliet-computer})$$

waarbij $K^d(x)$ de ‘kans dat x defect is’ voorstelt. Immers zonder invoer/uitvoer faciliteiten is een computerinstallatie zo goed als onbruikbaar. Daarom zorgt men ervoor dat de satelliet-computer erg betrouwbaar is (d.w.z. dat hij een hoge ‘gemiddelde tijd tussen twee falingen’ heeft). Men kan ervoor zorgen dat er een reserve satelliet-computer (ook wel ‘backup-computer’ genoemd) beschikbaar is. Bij een defect van de satelliet-computer kan dan de reserve-computer ingeschakeld worden.

Een tweede manier om zich te beschermen tegen defecten, is het afsluiten van een soort ‘**verzekering**’. (Dit wordt vrij algemeen toegepast in de informatica wereld.) De koper van bepaalde hardware of software zal met de leverancier een‘**onderhoudscontract**’ afsluiten waarin vastgelegd is binnen welke termijn (1 werkdag, 4 uren, 1 uur, …) een onderhoudsploeg aanwezig moet zijn om het gesignaleerde defect te herstellen. Het spreekt vanzelf dat hoe ‘korter’ deze termijn is, hoe ‘duurder’ het contract zal zijn.

1. Een installatie met kanaalbestuurders kent hetzelfde nadeel: als een kanaalbestuurder uitvalt, wordt een groot deel van de randapparatuur onbruikbaar.

4.4.4 Randapparatuur-besturingsroutines

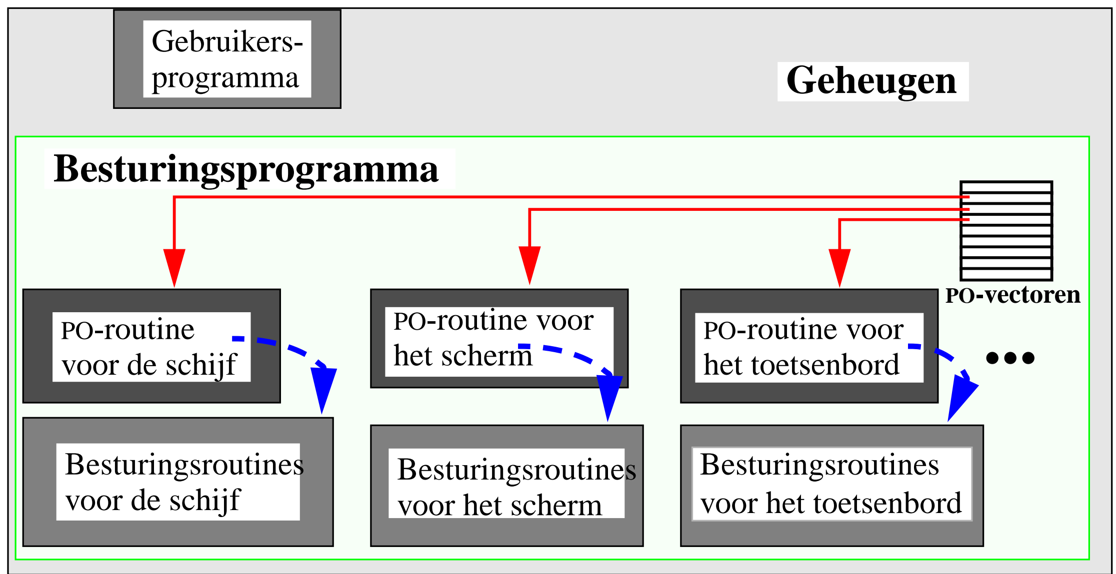
Uit de voorgaande paragrafen blijkt heel duidelijk dat het heel complex is om rechtstreeks met de randapparaten te werken, en dit om verschillende redenen:

- Je moet volledig op de hoogte zijn van **hoe** een randapparaat moet **bestuurd** worden: welke bevelen beschikbaar zijn, welke bevelenopmaak voorzien is, in welke volgorde bepaalde bevelen moeten gegeven worden, enz. Elke soort van randapparaat, zelfs elk merk verschilt. Van enige systematiek is zeker geen sprake.

  Er worden door de fabrikanten wel wat inspanningen gedaan om tot een soort 'standaard' te komen. Alle randapparaten die aan deze standaard voldoen, kunnen op dezelfde manier aangesproken worden. Helaas zijn er op dit ogenblik heel veel verschillende standaarden. Een voorbeeld is de SCSI-standaard, die op werkstations redelijk populair is. SCSI[1] staat voor 'Small Computer System Interface'.
- Indien de in- en uitvoer met behulp van programma-onderbrekingen gebeurt, zal je een vorm van **buffering** moeten voorzien. Bijvoorbeeld, indien je tien letters op het scherm wilt afdrukken, zal je die tijdelijk ergens in het geheugen moeten opslaan, zodat je bij elke PO-aanvraag vanwege de bestuurder van het scherm, een volgende letter kunt doorsturen. Buffering veronderstelt ook een goede ***boekhouding*** zodat geen informatie verloren gaat of beschadigd wordt.
- Er kunnen **tijdsafhankelijke problemen** optreden, indien je niet snel genoeg ingaat op een onderbrekingsaanvraag van een bestuurder. Bijvoorbeeld, indien de bestuurder van het klavier meldt (via een PO-aanvraag) dat een teken beschikbaar is, en de processor stelt deze PO te lang uit, dan bestaat de kans dat het teken verloren gaat, doordat er reeds een nieuw teken toegekomen is. (Er is 'overloop' opgetreden; de bestuurder zal zich in de **`FOUT`**-toestand plaatsen.)

We mogen dus terecht besluiten dat invoer en uitvoer eigenlijk moeilijke bewerkingen zijn. Om de programmeur te ontlasten in zijn programmeerwerk, zullen voor elk soort randapparaat de nodige routines geschreven worden die de in- en uitvoer regelen. Deze routines zullen we **randapparaat-besturingsroutines** noemen (in het Engels '*device drivers*'). Ze behoren tot het besturingssysteem en zullen uitgevoerd worden telkens een gebruikersprogramma iets wil inlezen of wegschrijven. Voor elk soort randapparaat hoeven ze slechts één keer ontworpen en uitgetest te worden. Eenmaal correct bevonden, kunnen ze toegevoegd worden aan het besturingsprogramma. In voorbeeld 4-5 (op pag. 143) werd reeds de aanzet gegeven tot het implementeren van deze besturingsroutines voor de harde schijf: de routines **`ms_operatie`**, **`positioneer_kam`** en **`start_transport`** zijn vereenvoudigde besturingsroutines. Figuur 4-31 toont de inhoud van het centrale geheugen. Een gedeelte wordt ingenomen door het besturingssysteem, dat o.a. voor elk soort randapparaat een afzonderlijk besturingsroutine bevat.

1. Spreek uit: "skoe-zi".



FIGUUR 4-31. Randapparaat-besturingsroutines en PO-routines in het besturingsprogramma.

Merk op dat voor elk randapparaat ook een programma-onderbrekingsroutine voorzien is. Deze routine zal uitgevoerd worden als de bestuurder van het randapparaat een programma-onderbreking aanvraagt. In de routine wordt nagegaan waarom de onderbreking aangevraagd werd (bijvoorbeeld, “de vorige opdracht is uitgevoerd” of “er is een fout opgetreden”, …), en of er een nieuwe opdracht gegeven kan worden aan het randapparaat. Indien dit zo is, zal de randapparaat-besturingsroutine opgeroepen worden. (Zie ook de stippellijnen in de figuur.)

Dus, de randapparaat-besturingsroutine wordt uitgevoerd indien:

- het gebruikersprogramma invoer wil krijgen of uitvoer wil sturen naar het randapparaat; hoe dit precies gebeurt, wordt in de volgende paragraaf uitgelegd;
- wanneer een programma-onderbreking vanwege het randapparaat is opgetreden, en de programma-onderbrekingsroutine wil het randapparaat een nieuwe opdracht geven.

Opgaven

1. Wat betekent: ‘in- en uitvoer inpassen in het geheugen’? Welke voor- en nadelen heeft deze techniek?
2. Wat is een bestuurder (besturingseenheid) van een randapparaat? Geef schematisch de werking aan. Schets het toestandsdiagram en geef de oorzaak voor elke overgang aan.
3. Hoe gebeurt de communicatie tussen processor en bestuurder van een randapparaat? Bestaan er meerdere methodes?
4. Op welke wijzen kunnen de in- en uitvoerbewerkingen gerealiseerd worden? Beschrijf bondig de kenmerken van elke methode.

5. Wat betekent 'actief wachten' ('busy waiting')?

6. In- en uitvoer m.b.v. programma-onderbrekingen kan de efficiëntie van het computersysteem verhogen. Verklaar.

7. Voor snelle randapparaten is het mechanisme van programma-onderbrekingen niet voldoende. Waarom?

8. Wat betekent directe geheugentoegang ('direct memory access')? Kan deze techniek voor elk soort randapparaat toegepast worden?

9. Wat betekent 'cyclus diefstal'? Hoe nadelig beïnvloedt dit de werking van de processor?

10. Wat is een kanaalbestuurder?

11. Wat zijn satelliet-computers? Wat is het onderscheid tussen een 'front-end' en een 'back-end' computer?

12. Vergelijk kanaalbestuurders en satelliet-computers. Welke voor- en nadelen zie je in het gebruik ervan?

13. Wat zijn randapparaat-besturingsroutines? Wanneer worden ze uitgevoerd?

14. Geef de DRAMA-code die nodig is om een getal van 10 cijfers op het scherm af te beelden (m.a.w. de `DRU`-operatie). Gebruik hiervoor de techniek van het 'actief wachten'.

15. Geef de DRAMA-code die nodig is om een getal van 10 cijfers in te lezen via het toetsenbord (m.a.w. de `LEZ`-operatie). Gebruik hiervoor de techniek van het 'actief wachten'. Zorg ervoor dat het programma de ingetypte symbolen toont op het scherm. (Men spreekt in dit geval over 'echo-transmissie').

16. Pas het vorige programma aan zodat je ook toelaat dat de gebruiker fouten bij het ingeven verbetert: hij/zij kan het laatst ingetypt symbool wissen door de ←-toets (ook wel de BS-toets genoemd ['**b**ack- **s**pace']) in te drukken. Ook het symbool op het scherm moet gewist worden: dit kan je doen door het laatste symbool te overschrijven met een 'blanko'-teken. Het verbeteren van fouten of 'editeren' is slechts mogelijk zolang de `return`-toets niet is ingedrukt! In de ASCII-code wordt het BS-teken door `008` voorgesteld en het blanko-teken door `032`. (Zie ook tabel A-11 (pag. 206).)

17. Pas het programma uit oefening 14 aan zodat je een getal op het scherm uitschrijft via programma-onderbrekingen. Zet het getal onmiddellijk om in ASCII-notatie, en bewaar deze in een geheugenzone. Schrijf het eerste cijferteken uit. Terwijl het scherm bezig is met het uitschrijven, verhoogt de processor voortdurend een teller. Bij elke programma-onderbreking wordt het volgende cijferteken naar het schermbestuurder gestuurd. Wanneer het getal volledig uitgeschreven is, schrijf je de inhoud van de teller uit (via actief wachten).

18. Pas de DRAMA-code voor de `LEZ`-operatie (oef. 15) aan zodat er gewerkt wordt met programma-onderbrekingen. Als de processor niets anders te doen heeft, verhoogt hij voortdurend een teller. Schijf de inhoud van de teller uit na het inlezen van het volledige getal.

19. Schrijf twee programma's die de priemgetallen tussen `2` en `200` berekenen en elk gevonden priemgetal afbeelden op het scherm. Elk getal moet opgesplitst worden in zijn afzonderlijke cijfers. In het eerste programma gebeurt dit afdrukken via een vorm van 'actief wachten', in het tweede via programma-onderbrekingen. Vergelijk de snelheid van de twee programma's.

20. Breid de besturingsprogramma's van voorbeeld 4-5 (op pag. 143) uit. Zorg ervoor dat een lees- of schrijfopdracht mag gegeven worden, terwijl er nog een andere bezig is. Plaats de nieuwe opdracht in een wachtrij en start de uitvoering zodra de schijf opnieuw vrij is.

4.5 Processortoestanden

In deze paragraaf zullen we de DRAMA-machine voor de laatste maal uitbreiden: eerst voorzien we een aantal processortoestanden, daarna zullen we de machinebevelen opdelen in twee groepen, de gewone en de geprivilegieerde. Op het einde van deze paragraaf beschrijven we tenslotte de volledige (en definitieve) bevelencyclus.

4.5.1 Halttoestand en uitvoeringstoestand

In paragraaf 3.7 (op pag. 92) hebben we terloops vermeld dat vroeger bij het opzetten van de spanning, de processor automatisch in de ***'halttoestand'*** werd geplaatst. Dit was nodig, want het geheugen bevatte nog geen instructies die konden uitgevoerd worden. Bij hedendaagse computers kan de processor wel onmiddellijk beginnen met de uitvoering van een programma dat zich in ROM-geheugen bevindt.

Nochtans wordt op moderne computers toch een **halttoestand** voorzien, omdat er perioden zullen zijn dat de processor werkelijk niets te doen heeft:

- er is geen programma om uit te voeren, (bijvoorbeeld, het vorige programma is volledig afgewerkt, en er is geen nieuw programma beschikbaar dat in het geheugen kan geladen worden en daarna uitgevoerd),
- het programma dat momenteel uitgevoerd wordt, heeft gegevens nodig die op een hulpgeheugen staan; de processor moet dus 'wachten' tot die gegevens ingelezen zijn[1].

Wanneer er geen werk is voor de processor, zou de processor **actief** kunnen **wachten** door bijvoorbeeld een oneindige lus uit te voeren:

```
LUS: SPR LUS
```

Op machines die geen halttoestand hebben, is dit de enige manier om de processor bezig te houden: immers de bevelencyclus gaat door! Een nadeel van deze methode is echter dat een eventueel gelijktijdig DGT-transport gestoord wordt[2].
De ontwerpers van de DRAMA-machine hebben bijgevolg twee toestanden voorzien waarin de processor zich kan bevinden:

1. **uitvoeringstoestand**, in dewelke de gewone bevelencyclus wordt uitgevoerd, (d.w.z. er worden voortdurend bevelen opgehaald, geanalyseerd en uitgevoerd),
2. **halttoestand**, in dewelke geen bevelen worden opgehaald en uitgevoerd; de processor houdt echter wel de PO-vlaggen in het oog. (Je zou kunnen zeggen dat de processor in een soort 'slaaptoestand' verkeert waaruit hij gewekt kan worden door het gerinkel van een 'bel'.)

---

1. In het voorbeeld 4-5 (op pag. 143) voerde de processor na het geven van de opdracht iets anders uit: de procedure `bereken()`. Indien echter de berekeningen gebaseerd zijn op de in-te-lezen gegevens, dan had de processor moeten wachten tot de bestuurder klaar is met de opdracht.
2. Zie ook de bemerking over 'cyclusdiefstal' op pag. 149.

Het tweede cijfer in het programmatoestandswoord (PTW_1 of **H/U** in figuur 4-32) duidt aan of de processor zich in de **h**alttoestand (**0**) bevindt of in de **u**itvoeringstoestand (**1**).

ONV	H/U	S/P	CC	OVI	SOI	◄		BT	►
0	1	2	3	4	5	6	7	8	9
G	GPF	WEK	DRK	IN	UIT	SCH	OVL	SPL	MFT
10	11	12	13	14	15	16	17	18	19

FIGUUR 4-32. De halt-/uitvoeringsindicator.

Overgangen tussen halt- en uitvoeringstoestand

Indien de processor van toestand verandert, dan ligt aan de basis van die toestandsverandering:

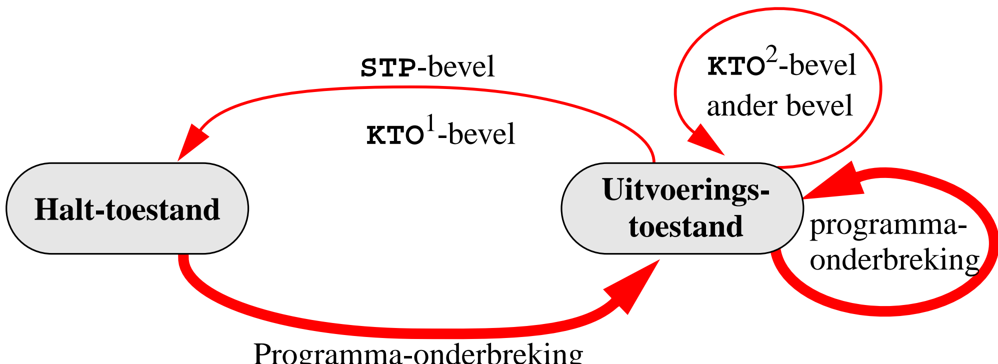
- ofwel de ***uitvoering*** van een bepaalde ***instructie***, (slechts enkele instructies kunnen de processortoestand beïnvloeden, zie verder),
- ofwel het optreden van ***programma-onderbreking***.

De eenvoudigste manier om de processor in de halttoestand te plaatsen, bestaat erin het ***stopbevel*** (**STP**) uit te voeren. De processor kan terug in de uitvoeringstoestand komen via een ***programma-onderbreking***.

Bijvoorbeeld, de processor heeft de schijfbestuurder de opdracht gegeven een sector in te lezen. Aangezien er geen ander werk te verrichten is wordt de processor in de halttoestand gebracht (**STP**-bevel) en zal de processor geen instructies meer uitvoeren tot de bestuurder een PO aanvraagt. We veronderstellen dat de aanvraag niet gemaskeerd is, dat het onderbrekingsniveau (**ONV**) laag genoeg is, d.i. kleiner dan 6 en dat er geen andere aanvragen voor een PO komen. Zodra de PO_6-vlag geplaatst is, zal een PO optreden: niet alleen wordt er een nieuwe BT geladen, maar ook het **H/U**-cijfer zal op **1** gezet worden, waardoor de processor terug in de uitvoeringstoestand komt en zal beginnen met de uitvoering van de PO-routine voor dit soort onderbreking.

Een tweede voorbeeld: stel dat de processor helemaal niets te doen heeft (zelfs geen invoer of uitvoer), dan zal de processor door de uitvoering van een **STP**-instructie zichzelf in de halttoestand brengen. Wanneer de gebruiker een programma wil laten uitvoeren, zal hij deze opdracht bijvoorbeeld via het toetsenbord ingeven. De bestuurder van het klavier zal via het PO-mechanisme aan de processor melden dat er een nieuwe ‘toetsaanslag’ beschikbaar is; hierdoor wordt de processor uit zijn halttoestand gehaald.

De overgangen tussen de twee toestanden wordt schematisch voorgesteld in figuur 4-33. Naast elke pijl staat de oorzaak van de overgang. Dikke pijlen duiden programma-onderbrekingen aan, dunne pijlen zijn het gevolg van de uitvoering van een machinebevel.



FIGUUR 4-33. De overgangen tussen de halt- en uitvoeringstoestand.

De uitvoering van de meeste bevelen laat de processor in de uitvoeringstoestand. Merk op dat de terugkeer uit een PO-routine (via de uitvoering van de **KTO**-instructie) de processor zowel in de halttoestand (**KTO**[1]) kan brengen als in de uitvoeringstoestand (**KTO**[2]) kan laten, afhankelijk van welke de toestand was op het ogenblik dat de PO optrad. Herinner je dat bij de uitvoering van het **KTO**-bevel het topelement van de stapel wordt gehaald en in $PTW_{0\text{-}9}$ wordt geladen; bijgevolg, zal ook het **H/U**-cijfer terug zijn oude waarde krijgen. Natuurlijk is het altijd mogelijk dat de processor deze ‘oude waarde’ eerst wijzigt, alvorens de **KTO**-instructie uit te voeren.

Tot hiertoe hebben we op het einde van elk programma een **STP**-bevel geschreven. Dit is niet zo verstandig, want het is mogelijk dat er nog andere programma’s moeten uitgevoerd worden. In voorbeeld 4-7 wordt getoond hoe we dit kunnen oplossen.

Voorbeeld 4-7. ***Het beëindigen van een programma.***
We veronderstellen dat er verschillende programma’s in het geheugen geladen zijn, die na elkaar worden uitgevoerd, Uiteraard heeft geen enkel programma weet van de andere programma’s. De lader houdt in het geheugen een lijst bij met de beginadressen van deze programma’s. In C ziet de declaratie van de lijst eruit als volgt:

```
struct prog {
    int * beginadres;
    struct prog * volgend;
};

struct prog * uit_te_voeren = 0; /* lijst van programma's */
```

De lader wordt aangepast, zodat hij voor elk geladen programma het volgende uitvoert::

```
struct prog * nieuw = alloc (2);
nieuw->beginadres = ... adres van het geladen programma ...;
/* vooraan toevoegen in lijst */
nieuw->volgend = uit_te_voeren;
uit_te_voeren = nieuw;
```

*Bij het einde van elk programma, wordt het STP-bevel vervangen door de volgende rij bevelen. We veronderstellen dat de variabele 'uit_te_voeren' op adres **9500** bewaard wordt:*

```
                | in plaats van STP
       HIA     R0,9500        | Is er iets te doen?
       VSP     NUL,STOP
       HIA     R1,1(R0)       || Verwijder uit lijst
       BIG     R1,9500        ||
       SPR     0(R0)          | Begin uitvoering
STOP:  STP                    | Stop processor
```

4.5.2 Probleemtoestand en supervisietoestand

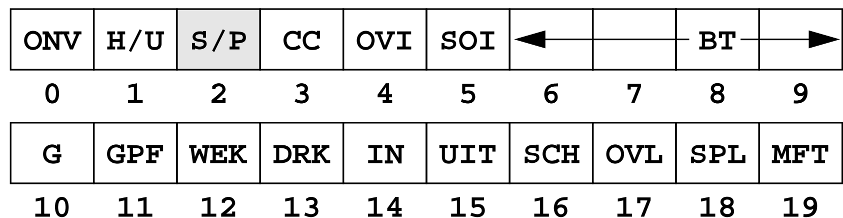
In- en uitvoer zijn heel complexe bewerkingen (zie ook paragraaf 4.4), die best overgelaten worden aan het besturingsprogramma. Het BP heeft daarom voor elk randapparaat de nodige besturingsroutines en bijbehorende PO-routine. Willen we de **correcte werking** van de computer kunnen garanderen, dan moeten we kunnen verhinderen dat de 'eigenwijze' programmeur toch zelf de randapparatuur probeert aan te spreken. Een fout is snel gemaakt. Voor een scherm of een toetsenbord is dit misschien niet zo erg, maar voor een schijf des te meer: indien per ongeluk de verkeerde sector overschreven wordt, is de oorspronkelijke inhoud onherroepelijk verloren. We moeten dus kunnen verhinderen dat een gewoon programma rechtstreeks bevelen geeft aan de besturingseenheden van de randapparaten.

Nochtans, het besturingssysteem zelf (in het bijzonder de randapparaat-besturingsroutine) moet wel in staat zijn om die opdrachten door te geven aan de besturingseenheden van de randapparaten. We moeten dus een onderscheid kunnen maken of de processor nu een gewoon programma aan het uitvoeren is, of het besturingsprogramma. We zullen daarom de uitvoeringstoestand verder onderverdelen in:

- een **supervisietoestand**[1], die aangeeft dat de processor bezig is met de uitvoering van het besturingsprogramma;
- een **probleemtoestand**, die aangeeft dat de processor bezig is met de uitvoering van een gewoon programma.

Dit onderscheid wordt aangegeven door het derde cijfer in het PTW (PTW_2 of **`S/P`**) (zie ook figuur 4-34). In de supervisietoestand heeft **`S/P-indicator`** de waarde **`0`**, in probleemtoestand de waarde **`1`**.

1. Vroeger werd het besturingsprogramma ook wel supervisieprogramma genoemd, omdat dit programma de werking van de computer superviseert.



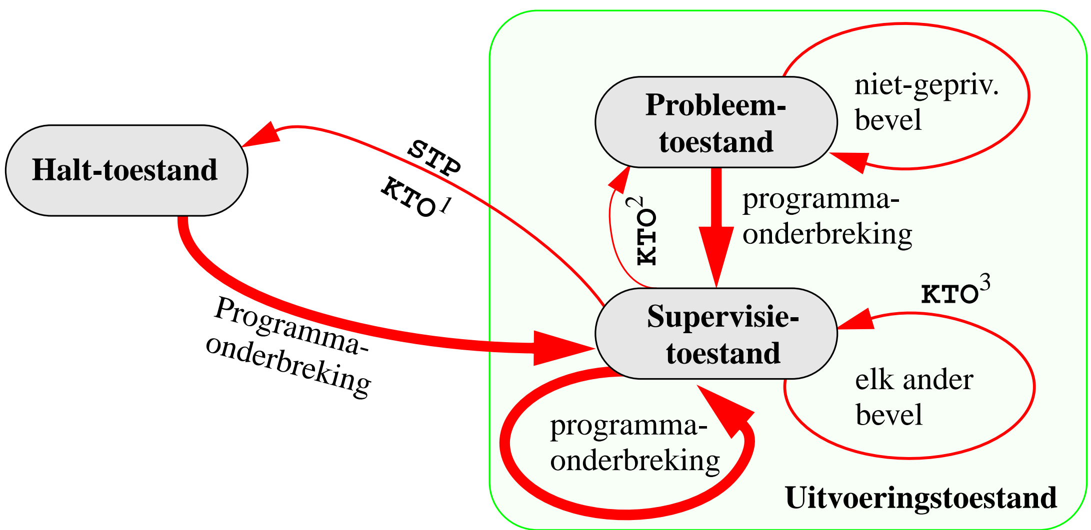
ONV	H/U	S/P	CC	OVI	SOI	←		BT	→
0	1	2	3	4	5	6	7	8	9
G	GPF	WEK	DRK	IN	UIT	SCH	OVL	SPL	MFT
10	11	12	13	14	15	16	17	18	19

FIGUUR 4-34. De supervisie-/probleemtoestand indicator.

Overgangen tussen halt-, probleem- en supervisietoestand

Figuur 4-35 geeft alle mogelijke overgangen tussen de verschillende toestanden weer[1]. Drie zaken vallen op in de figuur:

1. **Elke** programma-onderbreking zal de processor steeds naar de supervisietoestand brengen (zie de dikke pijlen).
2. De terugkeer uit de PO-routine (via de **KTO**-instructie) zal de processor opnieuw in de vorige toestand brengen (halttoestand [**KTO**^{1}], probleemtoestand [**KTO**^{2}] of supervisietoestand [**KTO**^{3}]). Immers, bij elke programma-onderbreking wordt de oude waarde van PTW$_{0..9}$ op de top van de stapel gezet; bijgevolg wordt de vorige processortoestand bewaard.
3. Het **STP**-bevel brengt de processor in de halttoestand, terwijl de uitvoering van de andere bevelen (niet **STP** of **KTO**) de toestand van de processor niet wijzigt.



FIGUUR 4-35. De overgangen tussen de verschillende processor-toestanden.

1. De aanduiding ***'elk niet-geprivilegieerd bevel'*** wordt later uitgelegd. Het volstaat hier voorlopig te lezen 'elke ander bevel'.

4.5.3 Geprivilegieerde bevelen

Supervisie- en probleemtoestand duiden dus aan of de processor (een deel van) het besturingsprogramma (BP) aan het uitvoeren is of een gewoon gebruikersprogramma. Nu moeten we nog verhinderen dat tijdens de uitvoering van een gebruikersprogramma rechtstreeks de randapparaten aangesproken worden (via **INV** en **UTV**) terwijl het BP wel die mogelijkheid moet hebben. (Het BP is dus **geprivilegieerd** t.o.v. een gewoon programma.)
De machinebevelen worden daarom opgedeeld in twee groepen:

- **geprivilegieerde** bevelen en
- **niet-geprivilegieerde** bevelen.

Geprivilegieerde bevelen mogen alleen uitgevoerd worden indien de processor zich in de supervisietoestand bevindt.

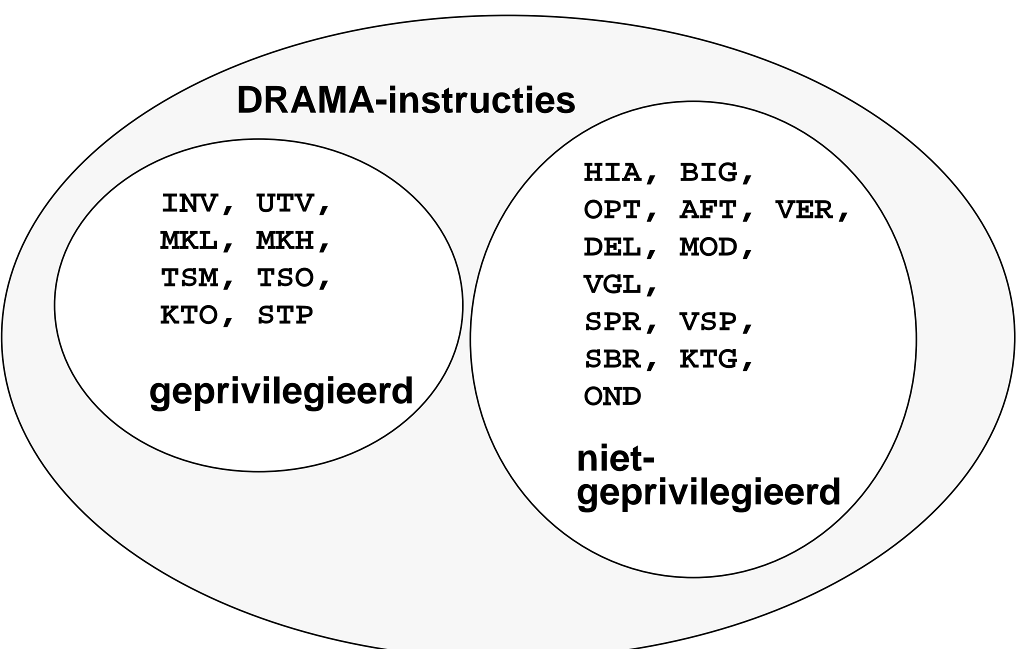
In probleemtoestand zal het geprivilegieerde bevel niet uitgevoerd worden; dit is immers niet geoorloofd. Bovendien zal in dit geval een PO optreden. Op de DRAMA-machine wordt hiervoor PO_9 gebruikt[1].

Op de DRAMA-machine zijn de volgende bevelen geprivilegieerd:

- de invoer- en uitvoerbevelen: **INV** en **UTV**,
- alle bevelen die het programma-onderbrekingsmechanisme beïnvloeden of testen: zoals **MKL**, **MKH**, **TSM** en **TSO**,
- bevelen die de processortoestand wijzigen: **KTO** en **STP**[2].

Kortom, alle bevelen die direct of indirect de toestand van de processor of het PO-mechanisme kunnen beïnvloeden, en alle bevelen die een effect hebben op de randapparatuur, de computerklok[3], eventuele wekkers, beschermingsregisters, enz. zijn geprivilegieerd. Deze bevelen kunnen immers de correcte werking van de computer ondermijnen. Zie ook figuur 4-36.

1. PO_9 is dus tegelijkertijd de PO voor ***ongeldige*** (d.i. niet-bestaande) functiecodes (wat we reeds in paragraaf 4.3.4 zegden) en voor ***ongeoorloofde*** functiecodes.
2. Het kan eigenaardig overkomen, maar een gewoon programma mag de processor niet in de halttoestand brengen. Zie verder.
3. Zie verder.



FIGUUR 4-36. Opdeling van de machine-instructies op de DRAMA-computer.

Tabel 4-7 vat de invloed van de processortoestand op de uitvoering van de bevelen nog eens samen. In supervisietoestand mogen alle bevelen uitgevoerd worden, dus zowel de geprivilegieerde als de niet-geprivilegieerde; in probleemtoestand, echter, mogen alleen de niet-geprivilegieerde instructies uitgevoerd worden.

TABEL 4-7. De uitvoering van instructies in de verschillende uitvoeringstoestanden.

	Type instructie	
Processortoestand	**Geprivilegieerd** (bv. `STP`, `INV`, `UTV`, `MKL`, ...)	**Niet-geprivilegieerd** (bv. `HIA`, `OPT`, `VGL`, `SPR`, ...)
Supervisietoestand	Uitgevoerd	Uitgevoerd
Probleemtoestand	PO_9 !!!	Uitgevoerd

4.5.4 Supervisie-oproep

Indien een programma zelf niet meer de in- en uitvoerapparaten mag besturen, hoe kan het dan gegevens inlezen of wegschrijven? Het moet hiervoor een beroep doen op het besturingsprogramma. Met andere woorden, er moet een routine van het BP uitgevoerd worden. Merk op dat we dit niet kunnen doen m.b.v. het gewone **SBR**-bevel! Ga zelf eens na waarom dit niet kan.

Antwoord: Omdat die routine dan nog steeds in 'probleemtoestand' zou worden uitgevoerd, en bijgevolg de routine geen **UTV** en **INV** bevelen zou mogen uitvoeren.

Bekijk nog eens figuur 4-35. De enige manier om over te gaan van probleemtoestand naar supervisietoestand is het laten optreden van een 'programma-onderbreking'. Het gebruikersprogramma moet dus een PO '**forceren**', en dit is mogelijk via een speciale instructie die we in paragraaf 4.3.7 (op pag. 121) geïntroduceerd hebben, namelijk: **OND**. Het effect van deze instructie is dat de PO_1-vlag geplaatst wordt, waardoor een PO zal optreden[1].

Het **OND**-bevel noemt men ook wel een '***supervisie-oproep***' (in het Engels '*supervisor call*' of '*system call*') omdat het als effect heeft dat a.h.w. een routine van het supervisie-programma (d.i. het besturingsprogramma) opgeroepen wordt (via het PO-mechanisme).
De operand van de **OND**-instructie[2] duidt aan welke 'dienst' we van het besturingsprogramma verwachten. Daarom noemden we die operand in paragraaf 4.3.7 het 'dienstnummer'. Bijvoorbeeld `OND 1` zou kunnen betekenen dat een getal ingelezen moet worden in register **R0**, `OND 2` dat de inhoud van register **R0** afgedrukt moet worden, enz. Soms zullen extra argumenten moeten doorgegeven worden aan de PO-routine (voor PO_1), namelijk wanneer complexere diensten aangevraagd worden. Bijvoorbeeld, wanneer het programma een tekst wil afdrukken op het scherm (`OND 10`), dan zal het ***adres*** van de geheugenzone die deze tekst bevat (in ASCII-voorstelling) en de ***lengte*** van die tekst als extra argumenten moeten doorgegeven worden. Deze argumenten kunnen in accumulatoren of op de stapel bijgehouden worden.

Dus waar we vroeger in DRAMA-programma's **LEZ** of **DRU** schreven, zouden we eigenlijk `OND 1` respectievelijk `OND 2` moeten gebruiken. Ook het **STP**-bevel is taboe en moet vervangen worden door `OND 9999`. De cijfers **1**, **2**,…, **9999** die als operand voor de **OND**-instructie gebruikt worden, zijn echter niet universeel. Elk besturingsprogramma zal zijn eigen conventies gebruiken. Dit betekent dat wanneer een ander BP gekozen wordt, alle programma's zouden moeten aangepast worden. Om de DRAMA-taal toch gebruiksvriendelijk te houden (voor de programmeur) heeft de ontwerper van de (voor)vertaler bepaalde geprivilegieerde

1. De PO zal alleen optreden indien het onderbrekingsniveau (**ONV**) op dat ogenblik **0** is, wat de normale waarde is voor **ONV** tijdens de uitvoering van een gebruikersprogramma. Een hogere waarde wijst er immers op dat een PO opgetreden is, en dat de processor een PO-routine aan het uitvoeren is, en alle PO-routines behoren tot het BP.

2. De **OND**-instructie heeft eigenlijk geen operand. Het enige effect van deze instructie is dat de PO_1-vlag geplaatst wordt. De ontwerpers van de DRAMA-machine hebben echter voorzien dat de 4 meest rechtse cijfers van deze instructie om het even welke waarde mogen hebben. Deze 4 cijfers kunnen dus door de programmeur gebruikt worden om er een waarde tussen **0000** en **9999** in weg te bergen. De DRAMA-processor zelf bekijkt deze cijfers niet bij het analyseren en uitvoeren van het bevel!

bevelen (zoals **STP**) en een aantal 'hoog-niveau' bevelen (zoals **LEZ**, **DRU**, **NWL**, …) gedefinieerd als macro's[1]. We zullen dergelijke macro's 'pseudo-bevelen' noemen, omdat ze het uitzicht hebben van gewone bevelen. Door deze pseudo-bevelen te gebruiken, zijn de DRAMA-programma's onafhankelijk van het gebruikte besturingsprogramma.

Voorbeeld 4-8 illustreert dat een programma heel vaak beroep doet op de assistentie van het besturingsprogramma. Links staat de originele DRAMA-code, rechts het programma zoals het door de voorvertaler omgezet is, alvorens het naar machinetaal vertaald wordt. Merk op dat zelfs in dit zeer eenvoudige voorbeeld, er vijf maal beroep gedaan wordt op het BP. Tijdens de uitvoering van een programma van enige omvang mag men verwachten dat er enkele honderden, ja zelfs enkele duizenden keren een supervisie-oproep gemaakt wordt!

Voorbeeld 4-8. ***Een gebruikersprogramma met supervisieoproepen.***

In een eenvoudig gebruikersprogramma dat een waarde X inleest via het toetsenbord en X^2 afdrukt op het scherm. Het inlezen, afdrukken en stoppen gebeurt via supervisie-oproepen.

```
| Lees X in en druk X en X**2 af
LEZ
DRU
VER   R0,R0      | X := X*X
DRU
NWL
STP
```

```
| Lees X in en druk X en X**2 af
OND   1
OND   2
VER   R0,R0         | X := X*X
OND   2
OND   3
OND   9999
```

Deze methode waarbij assistentie van het besturingsprogramma gevraagd wordt is zo belangrijk in moderne computersystemen, dat het vandaag overal aangetroffen wordt. Niet alleen voor in- en uitvoer maar ook voor tal van andere 'taken' zal een beroep gedaan worden op het besturingsprogramma (zoals het laden van een programma, het stoppen van het programma, enz.). In voorbeeld 4-9 (op pag. 168) wordt de code voor de PO-routine voor supervisie-oproepen weergegeven, de PO_1-routine (**PO_ROUT1**).

Probeer zelf eerst uit te pluizen wat de code precies doet. Vergelijk daarna je antwoord met de beschrijving op pag. 169. Vooral de bevelen die volgen na de commentaarlijn ***'Welk soort dienst gevraagd?'*** verdienen de nodige aandacht.

1. Via een speciale optie kan men aan de (voor)vertaler vragen deze macro's niet te definiëren. Indien men immers het besturingsprogramma zelf wil vertalen, dan mogen wel geprivilegieerde bevelen gebruikt worden.

Voorbeeld 4-9. ***De behandelingsroutine voor supervisie-oproepen.***

```
| Deze routine wordt geactiveerd via OND αβχδ
PO_ROUT1:  BIG     R0,BEW_PO1      | Bewaar inhoud registers
           …
           BIG     R9,BEW_PO1+9
           | Welk soort dienst gevraagd?
           HIA     R0,0(R9)        | top v. stapel (oude PWT)
           MOD     R0,TIENDZND     | R0 bevat vroegere BT
           AFT.w   R0,1            | Adres van OND-bevel
           HIA     R1,0(R0)        | R1 bevat het OND-bevel
           MOD     R1,TIENDZND     | R1 bevat αβχδ (dienstnr.)
 DIENST1:  VGL.w   R1,1
           VSP     NGEL,DIENST2
           SBR     LEZ_PROC        | roep de LEZ-procedure op
           SPR     TERUG1          | R0 bevat ingelezen getal
 DIENST2:  VGL.w   R1,2
           VSP     NGEL,DIENST3
           SBR     DRU_PROC        | roep de DRU-procedure op
           SPR     TERUG
 DIENST3:  …
           …                       | de andere diensten
   TERUG:  HIA     R0,BEW_PO1
  TERUG1:  HIA     R1,BEW_PO1+1
           …
           HIA     R9,BEW_PO1+9
           KTO
 BEW_PO1:  RESGR   10
TIENDZND:  10000
           …
| Routine voor het inlezen van een getal via het klavier
LEZ_PROC:  …
           …
| Andere routines die diensten aanbieden
           …
```

Beschrijving van de code uit voorbeeld 4-9:

- De routine begint met het **bewaren** van de inhouden van alle **rekenregisters**. Van dan af kan het BP de rekenregisters vrij gebruiken voor de eigen berekeningen.
- Daarna wordt nagegaan **welke dienst** aangevraagd wordt: dit wordt aangegeven door de operand van het `OND`-bevel. Die operand moet echter zelf uit het geheugen opgehaald worden; immers de DRAMA-processor houdt geen rekening met die operand. Het BP zal dus eerst de inhoud van het geheugenregister waarin het `OND`-bevel opgeslagen is, moeten ophalen. Het adres van dit geheugenregister wordt gevonden op de top van de stapel (daar werd immers de oude PTW weggeborgen); alleen de vier minst beduidende cijfers (`BT`) zijn belangrijk (`mod 10000`). Aangezien de bevelenteller reeds opgehoogd was op het ogenblik dat de `OND`-instructie werd uitgevoerd, moet er van dit adres één worden afgetrokken. Van het opgehaalde bevel worden alleen de 4 minst beduidende cijfers overgehouden (`mod 10000`). Deze waarde geeft aan welke dienst aangevraagd wordt.

  Het BP dat op de DRAMA-machine draait, gebruikt de volgende conventie:

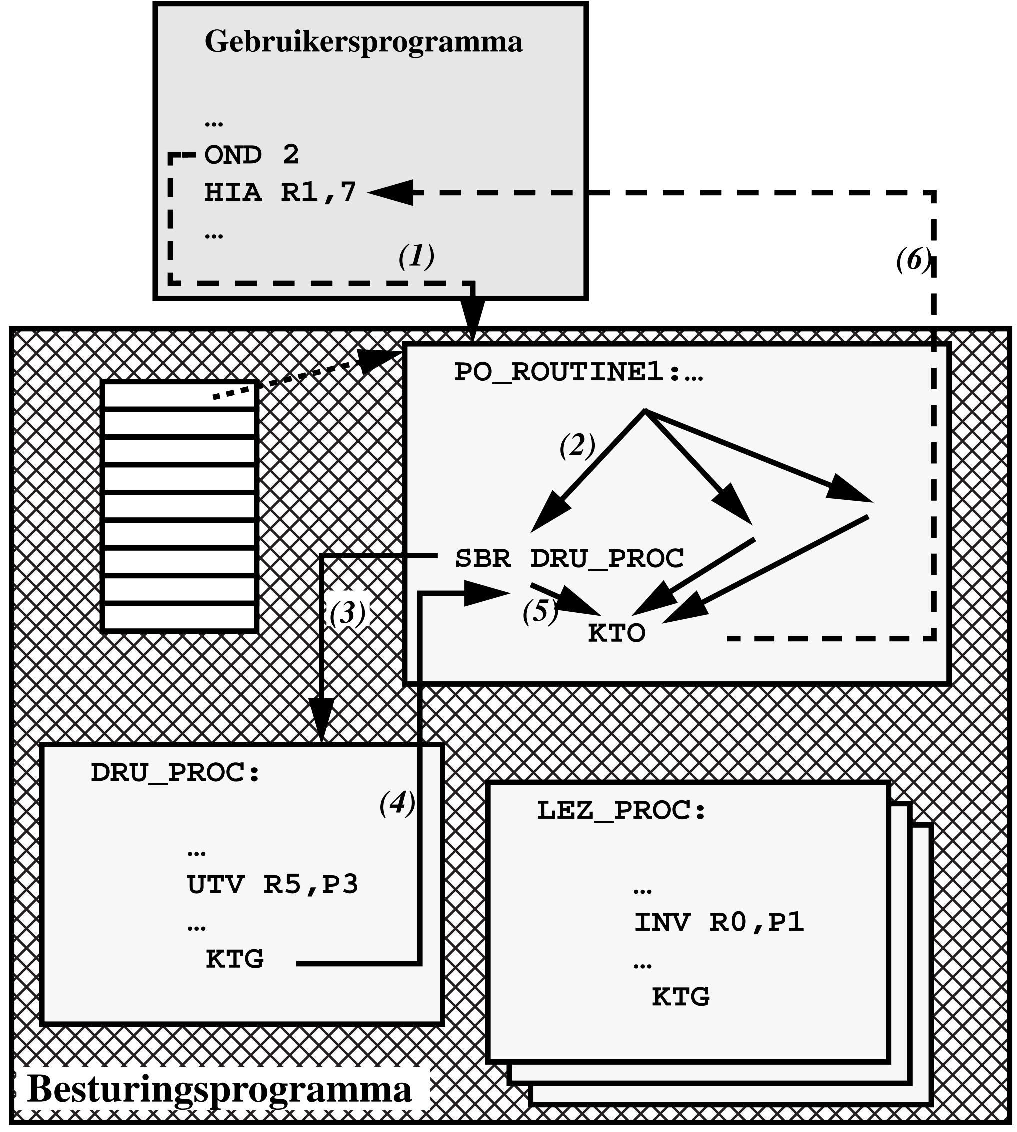
TABEL 4-8. "Dienstnummers"' voor het DRAMA-besturingsprogramma.

Nummer (in `OND`-instr.)	Aangeboden dienst (door het BP)	Verkorte notatie
1	inlezen van een geheel getal in reg. `R0`	`LEZ`
2	afdrukken van inhoud van reg. `R0`	`DRU`
3	naar de volgende lijn	`NWL`
...		
9999	het programma beëindigen	`STP`

- De code roept tenslotte de subroutine op die deze **dienst** aanbiedt.
- Na de terugkeer uit de subroutine, worden de **rekenregisters** in hun oude toestand **hersteld** (uitgezonderd deze die een resultaat bevatten, zoals register `R0` bij het inlezen van een getal (zie dienst 1), …).
- Tenslotte keert de processor terug (via `KTO`) naar het programma dat de supervisie-oproep heeft uitgevoerd.

In figuur 4-37 wordt getoond welke instructies worden uitgevoerd tijdens een typische supervisie-oproep waarbij het gebruikersprogramma een getal op het scherm wil afdrukken (`OND 2`). Voor de leesbaarheid worden de bevelen in symbolische vorm weergegeven. In werkelijkheid zouden het getallen zijn. Bestudeer deze figuur aandachtig en verklaar elke overgang (pijlen ***(1)*** t.e.m. ***(6)***).

Kan je aangeven in welke ***'toestand'*** de processor zich bevindt wanneer elk stukje code uitgevoerd wordt?



FIGUUR 4-37. Een supervisie-oproep.

Antwoord: De streepjeslijnen zijn overgangen waarbij de processortoestand verandert:

- in ***(1)*** van probleemtoestand naar supervisietoestand t.g.v. de **`OND`** opdracht en
- in ***(6)*** van supervisietoestand naar probleemtoestand t.g.v. **`KTO`**.

Bij de volle lijnen verandert de processortoestand niet:

- in ***(2)*** gaat de **`PO_ROUT1`** na welke dienst wordt aangevraagd en ***(3)*** roept de routine op die deze dienst verzorgt;
- ***(4)*** keert terug van de subroutine-oproep (**`SBR`** noch **`KTG`** veranderen de processortoestand, deze is dus nog altijd supervisietoestand);
- in ***(5)*** worden de inhouden van de rekenregisters hersteld.

Door het supervisie-oproep mechanisme wordt het besturingssysteem heel vaak ter hulp geroepen voor allerlei complexe en/of laag-niveau taken. Dit heeft belangrijke voordelen:

- De programmeur hoeft zich niet bezig te houden met laag-niveau details, maar kan zich concentreren op de oplossingsmethode. Dit bespaart heel wat **programmeertijd**.
- De complexe en/of laag-niveau taken worden **correct** geïmplementeerd. Er is dus minder kans op fouten[1]. Heel wat taken zijn complexer dan ze lijken: bijvoorbeeld, bij het ingeven van gegevens via het toetsenbord, wil de gebruiker graag op het scherm zien wat hij ingetypt heeft en of dit correct in de computer is aangekomen. Om dit mogelijk te maken wordt ***'echo-transmissie'*** toegepast: het BP zal het ontvangen symbool terug naar scherm sturen. Echter het toetsenbord en het scherm zijn verschillende randapparaten. Vaak zal ook de mogelijkheid geboden worden om de ingetypte symbolen te verbeteren.
- Het BP kan heel wat **optimisaties** voorzien, waardoor de efficiëntie van het computersysteem in zijn geheel verhoogt:
  - 'Actief wachten' wordt zoveel mogelijk vermeden. Door gebruik te maken van programma-onderbrekingen en DGT kan de processor vaak ander nuttig werk verrichten ***terwijl*** de randapparaten bezig zijn met het verwerken van de in- en uitvoer-opdrachten.
  - Wanneer het gebruikersprogramma aan het BP vraagt om iets te schrijven (op het scherm, op de schijf, …), is het mogelijk dat dit niet onmiddellijk maar pas later uitgevoerd zal worden. Bijvoorbeeld, de schijf kan nog bezig zijn met een vorige opdracht. Het BP zal de gegevens tijdelijk ergens in het geheugen opslaan, en later, als het randapparaat daar klaar voor is, de schrijf-opdracht(en) uitvoeren. Dit noemt men ook ***'uitgesteld wegschrijven'*** (in het Engels '*delayed write*'). Nadat het BP de gegevens in het geheugen geplaatst heeft, kan de processor terug verder werken aan het gebruikersprogramma. (Er wordt dus niet gewacht!)
  - Omgekeerd, op het ogenblik dat het programma vraagt om iets in te lezen van het klavier, is het best mogelijk dat die gegevens al ingelezen zijn (de gebruiker heeft ze reeds ingetypt en het BP heeft die gegevens tijdelijk in het geheugen opgeslagen). Dit noemt men ook ***'vooraf intypen'*** (in het Engels '*type ahead*').
  - Tenslotte, kan in sommige gevallen het BP proberen te voorspellen welke gegevens het gebruikersprogramma in de nabije toekomst zal inlezen van de schijf, en die gegevens reeds op voorhand inlezen. Dit noemt men ook '**vooraf inlezen**' (in het Engels '*read ahead*').

1. Geen enkel besturingssysteem is volledig foutvrij. Vandaar dat leveranciers regelmatig aanpassingen (in het Engels '*patches*') ter beschikking stellen die deze fouten verbeteren of omzeilen. Echter, indien de programmeur ook die code moest schrijven, zou de kans op fouten nog veel hoger zijn.

4.5.5 De volledige bevelencyclus

Tot slot herhalen we hier nog eens de volledige (en definitieve!) bevelencyclus, waarbij we de verschillende processortoestanden in aanmerking nemen.

1. Indien PTW_1 `= 0` (***halttoestand***), ga naar stap 4.

De volgende instructie wordt **opgehaald** op adres `BT`.

`BT` ← `BT + 1`.

2. De instructie wordt **geanalyseerd**.

- De functiecode wordt nagekeken.

  Indien het een ***geprivilegieerde instructie*** betreft, en de processor bevindt zich in probleemtoestand (PTW_2 `= 1`), dan wordt het bevel niet uitgevoerd (geen stap 3), maar wordt de PO_9-vlag geplaatst.

  Voor een ***ongeldige instructie*** (d.i. niet bestaande instructie) wordt ook de PO_9-vlag geplaatst.
- De operand wordt berekend, en eventueel uit het geheugen opgehaald.

3. De instructie wordt **uitgevoerd**.

`STP` brengt de processor in de '***halttoestand***', (PTW_1 ← `0`).

4. De processor gaat na of hij op een **programmaonderbreking** moet **ingaan**:

- indien het globaal masker (`G`, d.i. PTW_{10}) opstaat, ga verder bij stap **1** (d.w.z. begin een nieuwe bevelencyclus),
- onderzoek de PO-vlaggen waarvoor prioriteit ***groter*** is dan `ONV` (d.i. PTW_0),
- negeer deze waarvoor geen aanvraag is, of die gemaskeerd zijn,
- indien er geen overblijven, begin een nieuwe bevelencyclus (d.i. ga verder bij stap **1**),

  anders:
- kies deze met het hoogste nummer (stel PO_k):

  a. plaats $\text{PTW}_{0\text{-}9}$ op de stapel (bewaar de huidige toestand van de processor)

  b. PO-vlag[k] ← `0` (zet de 'bel' af)

  c. $\text{PTW}_{0..2}$ ← (k, `1`, `0`) (dit zijn `ONV`, `H/U`, `S/P`)

  d. $\text{PTW}_{6..9}$ ← PO-vector[k] (dit is `BT`)

  eventueel kunnen `CC`, `OVI` en `SOI` ook op nul gezet worden, maar dit is niet echt noodzakelijk:

  e. $\text{PTW}_{3..5}$ ← (`0`, `0`, `0`) (dit zijn `CC`, `OVI` en `SOI`)

Opgaven

1. Waarom hebben moderne computers een halttoestand? Is dit echt noodzakelijk? Welk alternatief bestaat er voor de processor?
2. Wat gebeurt er tijdens de 'halttoestand'? Het **STP**-bevel brengt de processor in deze toestand, bestaat er ook een mogelijkheid om dit via het **KTO**-bevel te doen? Zo ja, hoe? Hoe geraakt de processor terug in de uitvoeringstoestand?
3. Hoeveel verschillende uitvoeringstoestanden zijn er op de DRAMA-machine? Waarom hebben we meer dan één uitvoeringstoestand nodig?
4. Hoe komt de processor vanuit probleemtoestand in supervisietoestand? Zie je meer dan één mogelijkheid? Hoe komt de processor vanuit supervisietoestand terug in probleemtoestand? Geef het diagram dat alle overgangen tussen de verschillende processortoestanden beschrijft.
5. Wat zijn geprivilegieerde bevelen? Waarom zijn sommige bevelen geprivilegieerd? Geef een paar voorbeelden voor de DRAMA-machine.
6. Overloop de instructies uit figuur 4-36 (pag. 165), en argumenteer waarom een instructie al dan niet geprivilegieerd is.
7. Wat is een supervisie-oproep? Welke DRAMA-instructie komt hiermee overeen? Waartoe dient ze? Hoe worden argumenten doorgeven? Kan je een voorbeeld geven waarbij je inderdaad extra argumenten moet doorgeven?
8. Schets schematisch hoe de PO-routine eruit ziet voor supervisie-oproepen? Geef duidelijk aan in welke toestand de processor elk stukje code uitvoert.
9. Het is mogelijk dat de DRAMA-machine volledig vastloopt, d.w.z. dat de processor zich in de halttoestand bevindt en daar nooit meer uitgeraakt. Wanneer kan dit zich voordoen? Beschrijf een scenario dat aangeeft hoe de DRAMA-machine in deze toestand terecht is gekomen. Welke oplossing kan je voor dit probleem bedenken?
10. Indien er geen **OND**-bevel bestond, zou hetzelfde effect bereikt worden (namelijk het uitlokken van een PO) door een ***niet bestaande functiecode*** te gebruiken. (Sommige oudere processoren gebruiken deze methode.) Op de DRAMA-machine zou dit toch aanleiding geven tot een subtiel verschil. Welk?
11. Denk je dat het nuttig zou zijn een speciale instructie te voorzien die als enig effect heeft de toestand van de processor in supervisietoestand te brengen (bijvoorbeeld **SUP**)? Waarom of waarom niet? Zou **SUP** een geprivilegieerde instructie moeten zijn of niet? Waarom?
12. Wat zou je denken van een speciale instructie die als enig effect heeft de toestand van de processor in probleemtoestand te brengen (bijvoorbeeld **PRO**)? Is zo'n bevel nuttig of niet? Zou **PRO** een geprivilegieerde instructie moeten zijn of niet? Waarom?
13. Stel dat door een fout in het besturingsprogramma de oude $\text{PTW}_{0\text{-}9}$ waarde (die op de stapel staat) gewijzigd wordt. Maak een lijst van de mogelijke nare gevolgen die kunnen optreden na de uitvoering van de **KTO**-instructie.
14. Beschrijf de volledige bevelencyclus. Kan je verantwoorden waarom de ontwerpers van de DRAMA-machine het PTW op de beschreven wijze in twee delen (elk van 10 cijfers) hebben opgedeeld? Zouden bepaalde velden kunnen verhuizen naar het tweede gedeelte? Waarom of waarom niet?

15. Stel dat bij een PO telkens de volledige $\text{PTW}_{0\text{-}19}$ op de stapel geplaatst wordt, en bij de uitvoering van het **KTO**-bevel de volledige PTW hersteld wordt. Welke voor- of nadelen zou deze aanpassing hebben?

16. De code nodig 'om de aangevraagde dienst van een supervisie-oproep te weten te komen' (d.i. de operand van de **OND**-instructie) is vrij omslachtig. Zoek argumenten 'voor' en 'tegen' de ontwerpbeslissing bij het **OND**-bevel wel een operand toe te laten, terwijl de processor deze operand negeert. Welke alternatieven stel je voor? Hebben ze ook nadelen?

17. Vul de code voor de **LEZ**- en **DRU**-operaties (oefeningen 17 en 18 (op pag. 158)) in de supervisie-behandelingsroutine **PO_ROUT1**. Pas het programma dat de priemgetallen berekent aan zodat het gebruik maakt van supervisie-oproepen.

4.6 Multiprogrammatie

De randapparaten van een computersysteem werken traag in vergelijking met de processor (zie ook paragraaf 4.4). Bij geprogrammeerde in- en uitvoer heeft dit voor gevolg dat de processor het grootste deel van zijn tijd 'actief' staat te 'wachten' op de randapparaten. Vooral bij programma's die veel in- en uitvoer nodig hebben zoals administratieve toepassingen of interactieve programma's[1] kan de performantie[2] van het systeem tot een dramatisch laag niveau terugvallen (slechts enkele tienden van een percent). Bestudeer ook nog eens de berekening die volgt op voorbeeld 4-1 (op pag. 99).

Door gebruik te maken van programma-onderbrekingen en DGT kan de processor **terwijl de randapparaten bezig zijn** met de uitvoering van een invoer/uitvoer-opdracht **iets anders gaan doen**. Dit veronderstelt wel dat er ***iets anders te doen is!***

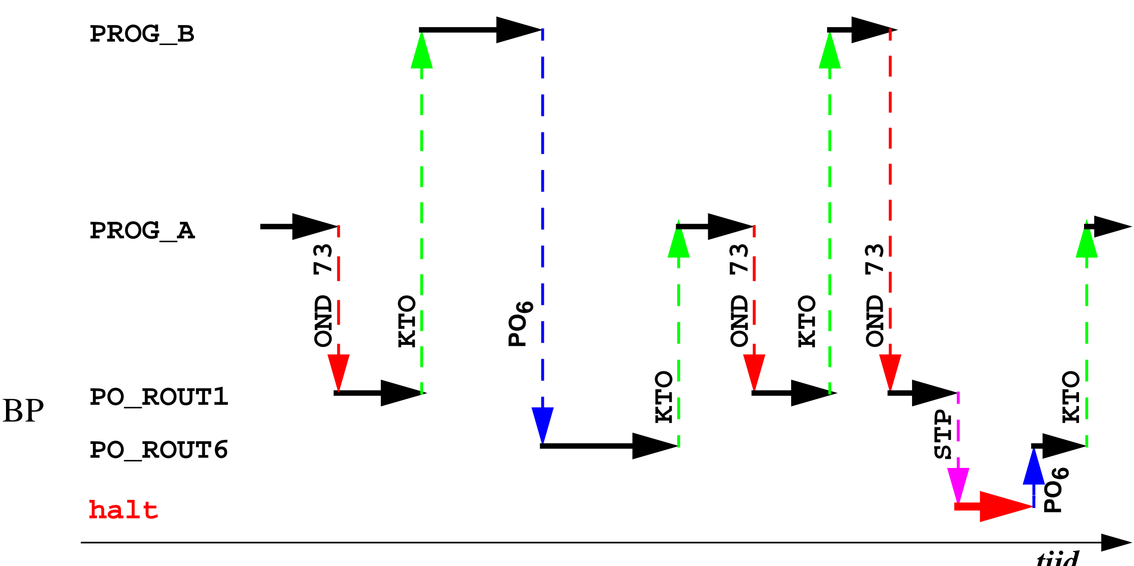
4.6.1 Het principe

Veronderstel dat het gebruikersprogramma (**`PROG_A`**) een sector wil inlezen van de schijf. Daartoe zal het programma een supervisie-oproep uitvoeren (bv. `OND 73`). Het BP zal de opdracht doorgeven aan de bestuurder van de schijf, en kan daarna iets anders gaan uitvoeren. De processor kan echter niet verder gaan met **`PROG_A`**, vermits het programma de gegevens die ingelezen moeten worden nodig heeft voor zijn berekeningen. Het BP heeft ook niets anders te doen. In voorbeeld 4-5 lieten we de processor na het starten van de leesopdracht berekeningen uitvoeren die onafhankelijk waren van de invoer, maar wat als er geen andere taken uit te voeren zijn? Dat is hij dus werkloos. Bijgevolg zal de processor zichzelf in de 'halttoestand' plaatsen door het **`STP`**-bevel uit te voeren (zie ook voorbeeld 4-7 (op pag. 161)). Het alternatief is het uitvoeren van een oneindige lus, maar dit stoort de DGT van de schijfbestuurder. Zodra de schijfbestuurder klaar is met de leesopdracht zal een PO de processor terug op gang brengen.

De situatie verandert echter indien er naast het gebruikersprogramma **`PROG_A`** nog een ander gebruikersprogramma, **`PROG_B`**, in het geheugen aanwezig is. Na het geven van de nodige opdrachten aan de schijfbestuurder, kan de processor beginnen (of verder gaan) met de uitvoering van **`PROG_B`**. Wanneer **`PROG_B`** gegevens nodig heeft van een of ander randapparaat, bestaat de kans dat de schijfbestuurder klaar is met de leesopdracht voor **`PROG_A`**, zodat de processor nu verder kan werken aan **`PROG_A`**.

Figuur 4-38 schetst het verloop in de tijd van de uitvoering van de verschillende programma's. We veronderstellen hierbij dat **`PROG_A`** een hogere prioriteit heeft dan **`PROG_B`**, zodat het BP de processor zal laten verwerken aan **`PROG_A`** zodra de schijf klaar is met de inlees-operatie (zie het **`KTO-`**bevel na de afhandeling van PO_6)

1. Dit zijn programma's die invoer verwachten van het toetsenbord en uitvoer sturen naar het scherm.
2. Dit is het percentage van de tijd dat de processor echt nuttige bevelen uitvoert.



FIGUUR 4-38. Multiprogrammatie.

De stippellijnen wijzen op een overgang: ofwel door een PO die veroorzaakt wordt door een supervisie-oproep (**OND**), ofwel door het **KTO**-bevel, ofwel door een andere PO. Bij elke overgang is de oorzaak aangegeven.

- **PROG_A** wil iets van schijf inlezen (**OND 73**).
- Het BP geeft de gepaste opdracht aan de schijfbestuurder[1] en geeft de processor aan **PROG_B** (via **KTO**). Merk op dat het BP ervoor moet zorgen dat juist vóór de **KTO**-instructie uitgevoerd wordt, de accumulatoren de correcte waarden hebben en dat op de top van de stapel een goede $\text{PTW}_{0..9}$ staat die deze overgang kan veroorzaken. Deze overgang wordt ook een **programmawisseling** genoemd (‘*context switch*’ in het Engels).
- **PROG_B** wordt onderbroken door een PO_6; dit betekent dat de schijfbestuurder klaar is met de vorige leesopdracht; het BP (**PO_ROUT6**) gaat na dat de lees-opdracht correct is uitgevoerd (wat we veronderstellen) en kan daarom de processor verder laten werken aan **PROG_A** (die een hogere prioriteit heeft dan **PROG_B**).
- **PROG_A** heeft nieuwe gegevens nodig en voert een nieuwe supervisie-oproep uit (**OND 73**).
- Het BP geeft opnieuw een opdracht aan de schijfbestuurder, en kan de processor verder laten werken aan **PROG_B**.
- **PROG_B** heeft ook gegevens nodig en voert een supervisie-oproep uit (**OND 73**).
- Het BP plaatst de aanvraag in een wachtrij (de schijf is immers nog bezig met de leesopdracht voor **PROG_A**). Er is niets anders te doen, dus zal het BP het **STP**-bevel uitvoeren.

1. In dit voorbeeld veronderstellen we dat voor alle leesopdrachten de koppen van de schijf reeds op de juiste cilinder gepositioneerd zijn en dat de kam dus eerst niet geherpositioneerd moet worden. De lezer kan zelf de nodige aanpassingen maken indien dit niet het geval is.

- De processor bevindt zich in de 'halttoestand' en voert geen instructies uit. Op dit ogenblik is het huidig onderbrekingsniveau **`1`**, en zijn alle maskers afgezet.
- Door een programma-onderbreking (PO_6) wordt de processor terug in gang gezet. Het BP gaat na of de vorige lees-opdracht correct is uitgevoerd, geeft de in een wachtrij geplaatste opdracht voor **`PROG_B`** door aan de schijfbestuurder en laat tenslotte de processor verder werken aan **`PROG_A`**.
- …

Dit '***gelijktijdig***[1]' uitvoeren van verschillende programma's noemt men **multiprogrammatie**. Telkens de processor zou moeten wachten wordt een ander programma uitgekozen waaraan verder kan gewerkt worden. Indien er veel programma's gelijktijdig in het geheugen gehouden worden (op een groot computersysteem kunnen dat er enkele honderden zijn) zal de processor (bijna) nooit **werkloos** ('*idle*' in het Engels) zijn.

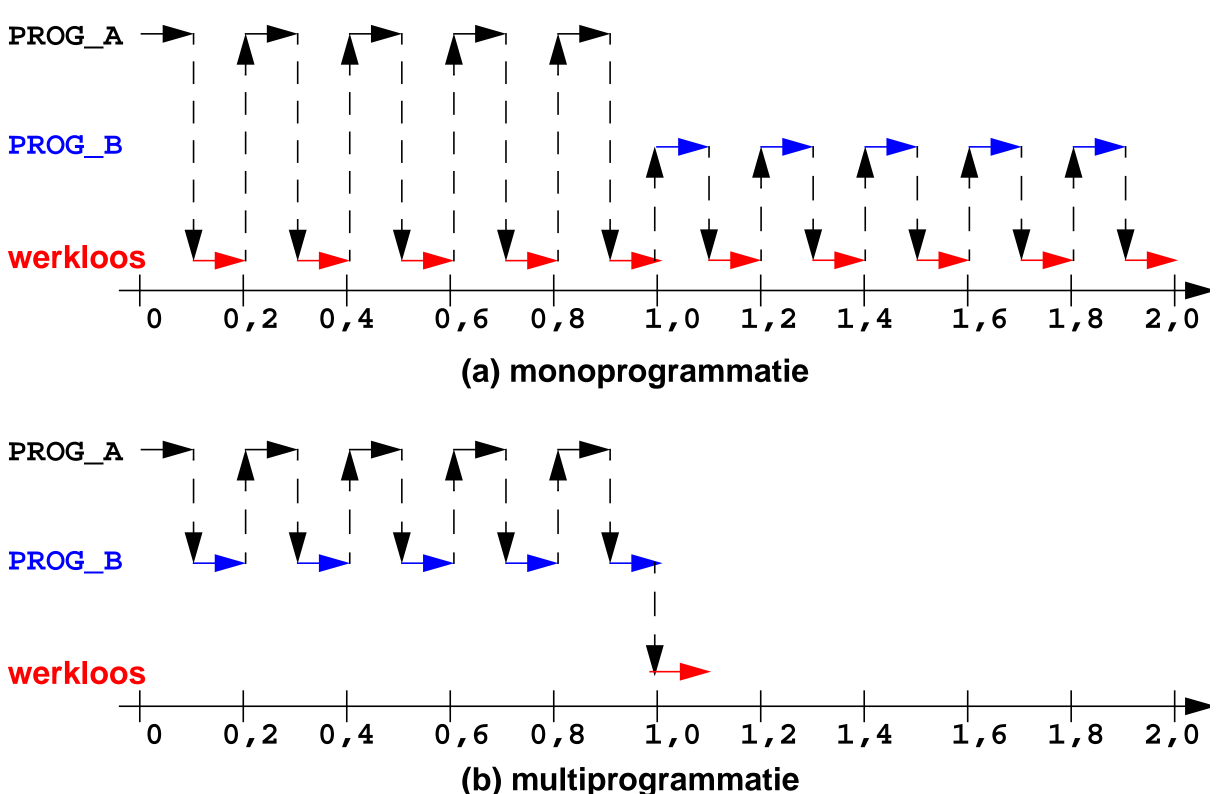
Alhoewel bij multiprogrammatie verschillende programma's afwisselend worden uitgevoerd, en dus de beschikbare processortijd verdeeld moet worden over deze programma's, zal de uitvoering van elk van deze programma's niet zoveel langer duren dan wanneer ze **alleen** of in **monoprogrammatie** zouden uitgevoerd worden. Monoprogrammatie is het tegenovergestelde van multiprogrammatie. Op elk ogenblik is er hoogstens één programma[2] in het geheugen geladen. Als synoniem voor monoprogrammatie wordt soms de Engelse term '*single task*' gebruikt.

4.6.2 De doorvoer

Veronderstel dat er twee (identieke) programma's moeten uitgevoerd worden. Elk programma begint met een berekening van 100 msec, verwerkt daarna vier gegevens, die elk eerst moeten ingelezen worden. Elke leesbewerking neemt 100 msec in beslag. De verwerking van elk gegeven duurt even lang (100 msec). Tenslotte wordt het resultaat weggeschreven (wat ook weer 100 msec in beslag neemt). De totale uitvoeringstijd voor dit ene programma is dus 1 sec (100 msec + 4 x (100 + 100) msec + 100 msec).

Indien we deze programma's in monoprogrammatie uitvoeren (d.w.z. na elkaar), duurt de totale uitvoeringstijd 2 sec. Indien deze programma's gelijktijdig in het geheugen zijn, duurt de totale uitvoeringstijd slechts 1,1 sec zoals blijkt uit figuur 4-39. Terwijl er gegevens ingelezen worden voor het ene programma, kan de processor het andere programma uitvoeren. (We verwaarlozen hier wel de instructies die nodig zijn om elke leesbewerking te starten, en deze die uitgevoerd worden wanneer de PO optreedt en de bevelen die de programmawisseling uitvoeren. Op een 1 MIPS machine zou dit hooguit enkele honderden µsec in beslag nemen.)

1. Echt gelijktijdig worden deze programma's natuurlijk niet uitgevoerd. De processor kan op elk ogenblik maar één bevel van één programma uitvoeren. Door het voortdurend wisselen, krijgen we echter wel de indruk alsof de processor de verschillende programma's gelijktijdig uitvoert. Wanneer er meerdere processoren aanwezig zijn is echte gelijktijdige uitvoering natuurlijk wel mogelijk.
2. Het besturingsprogramma tellen we niet mee.



FIGUUR 4-39. Mono- versus multiprogrammatie.

De **doorvoer** (‘*throughput*’ in het Engels) wordt gedefinieerd als ***“het aantal afgewerkte programma’s per tijdseenheid”***. In dit voorbeeld is de doorvoer dus 1 programma/s in monoprogrammatie terwijl hij bijna 2 programma’s/s in multiprogrammatie is. Een verbetering van 100%! Dus multiprogrammatie zal ertoe bijdragen dat de processor meer programma’s in hetzelfde tijdsbestek kan afwerken.

Op een PC werd vroeger vaak een besturingsprogramma gebruikt dat gebruikersprogramma’s in monoprogrammatie uitvoert (bijv. DOS). Meer ervaren gebruikers vonden het erg frustrerend dat het onmogelijk was om tijdens de uitvoering van een programma (die soms redelijk wat tijd in beslag kon nemen), de uitvoering van een ander programma te beginnen. Bijvoorbeeld, tijdens de vertaling van een groot programma, wou men een ander bestand kunnen wijzigen (ook editeren genoemd). Om hieraan tegemoet te komen werd een softwarepakket ontwikkeld (Windows[1]) dat een vorm van multiprogrammatie mogelijk maakt.

Op grotere systemen wordt monoprogrammatie zelden of nooit toegepast. Immers geen enkel programma (hoe rekenintensief ook) kan de processor voor 100% bezighouden. Dus is het altijd voordelig om nog enkele programma’s achter de hand (in het geheugen) te houden waarmee de periodes waarin de processor zonder werk dreigt te vallen, nuttig kunnen opgevuld

1. We bedoelen hier de windows die boven DOS draait. Windows95, WIndowsNT, Windows XP en Vista draaien niet boven DOS en voorzien multiprogrammatie.

worden. Toch zal ook dan de processor af en toe werkloos worden. Wanneer het computersysteem licht belast is (dit wil zeggen dat er slechts weinig programma's moeten uitgevoerd worden) zal de processor soms tot 90% van de tijd (of meer) werkloos zijn; op zeer zwaar belaste systemen daarentegen, slechts enkele procenten (bijv. 3%). Alhoewel de processor in dit laatste geval redelijk goed bezig blijft, zullen de gebruikers wel merken dat het systeem zwaar belast is: immers, de processor moet zijn tijd verdelen over zoveel programma's dat elk programma slechts een heel klein deeltje van die tijd toegemeten krijgt. Het lijkt alsof de computer ineens heel traag begint te werken!

4.6.3 Het STP-bevel

Het zou nu ook duidelijk moeten zijn waarom het **`STP`**-bevel geprivilegieerd is. Een gewoon programma zal nooit de processor in de halttoestand mogen brengen; alleen het BP weet of er al dan niet ander werk voor handen is. Daarom zal een programma —wanneer het afgewerkt is— aan het BP ***vragen om gestopt te worden*** (d.m.v. een supervisie-oproep; zie ook voorbeeld 4-8 (op pag. 167)). Het BP zal dan het programma uit het geheugen verwijderen en mogelijk een ander programma in de vrijgekomen ruimte laden. Indien het programma toch zou proberen een **`STP`**-bevel uit te voeren, zal tijdens de ontleding van het bevel een PO optreden (geprivilegieerd bevel in probleemtoestand), zodat het BP nog kan ingrijpen alvorens enig kwaad is geschied.

4.6.4 Het besturingsprogramma

Multiprogrammatie is essentieel om een computer **op een efficiënte wijze** te gebruiken. Het BP is er wel heel wat complexer door geworden. Er is een hele boekhouding nodig om ervoor te zorgen dat verschillende programma's tegelijkertijd (maar op verschillende plaatsen) in het geheugen gehouden worden en dat de programma-wisselingen correct uitgevoerd worden. Het BP zal voor elk programma dus een gegevensstructuur moeten bijhouden, waarin de toestand van elk programma opgeslagen is:

- de waarden van de accumulatoren
- de plaats waar de processor gekomen was (**`BT`**),
- de waarde van de conditiecode, de indicatoren, …

Uit figuur 4-38 (pag. 176) blijkt duidelijk dat de routines van het besturingsprogramma steeds via een programma-onderbreking opgeroepen worden (eventueel een geprogrammeerde programma-onderbreking: **`OND`**). Er is slechts één uitzondering: bij het opstarten van de computer.

Het opstarten van de computer

Wanneer de spanning wordt opgezet, wordt code die in ROM-geheugen zit uitgevoerd. Deze bevelen zullen eerst enkele elementaire testen uitvoeren (of het geheugen en de processor correct werken, …). Daarna wordt het besturingsprogramma geladen. Zoals we beschreven in paragraaf 3.7 (op pag. 92), gebeurt dit in verschillende stapjes. Eerst wordt vanaf een vaste

plaats van een bepaald randapparaat (bijvoorbeeld de eerste schijf) de ***'boot-sector'*** ingelezen en nadien uitgevoerd. Deze 'boot-sector' bevat een lader-programma dat de rest van het besturingsprogramma zal inlezen. Daarna wordt een sprong uitgevoerd naar de routine (van het BP) die de gegevensstructuren initialiseert. Tenslotte zal het BP op zoek gaan naar 'werk': ofwel moeten er één of meerdere gebruikersprogramma's[1] geladen en uitgevoerd worden, ofwel is er geen werk, en dan zal de processor in de halttoestand gebracht worden.

Van dan af zal het BP enkel nog via programma-onderbrekingen geactiveerd worden. Men zegt dat het BP 'aangedreven wordt door PO-en' ('*interrupt-driven*' in het Engels).

Opgaven

1. Wat betekent monoprogrammatie? Waarom heeft monoprogrammatie een nadelige invloed op de efficiënte van het computersysteem?
2. Wat betekent multiprogrammatie? Welke invloed heeft het op het al dan niet 'werkloos' zijn van de processor?
3. Wat betekent 'doorvoer'? Welke invloed heeft multiprogrammatie op de doorvoer? Bestaat er een lineair verband tussen de multiprogrammatiegraad (d.i. het aantal programma's dat gelijktijdig in het geheugen geladen is) en de doorvoer?
4. Stel dat na het opstarten van de computer en het laden van het BP er geen werk meer is. De processor wordt in de halttoestand geplaatst. Bedenk een scenario hoe de gebruiker zijn programma kan laten uitvoeren.
5. Het besturingsprogramma wordt aangedreven door programma-onderbrekingen. Wat wordt hiermee bedoeld?
6. Op een zwaar belast systeem (d.w.z. een computerssysteem waarbij heel veel programma's die moeten uitgevoerd worden in het geheugen geladen zijn) is de processor bijna voor 100% bezig. Waarom lijkt de computer zo traag te werken voor een gebruiker?

1. In een speciaal bestand, het configuratie-bestand, zou bijvoorbeeld aangegeven kunnen zijn dat er bepaalde programma's moeten opgestart worden.

4.7 Soorten besturingssystemen

De besturingsprogramma's kunnen opgedeeld worden in twee grote groepen:

- besturingsprogramma's voor systemen van **algemeen nut** ('*general purpose systems*' in het Engels). Dit zijn computersystemen die voor een brede waaier van doeleinden kunnen gebruikt worden.
- systemen voor een **specifiek doel** ('*dedicated systems*' in het Engels, ook soms '*embedded systems*' genoemd). Dit zijn systemen die ontworpen werden voor een zeer specifieke taak, die vaak te maken heeft met de controle over en sturing van apparaten of processen. Enkele voorbeelden: het computersysteem dat je aantreft in een vliegtuig, op een 'intensieve zorgen' afdeling van een ziekenhuis, of in een kerncentrale, …

  Een specifiek kenmerk van de meeste van deze systemen is dat ze in staat moeten zijn ***binnen een vaste tijdslimiet*** te reageren, zoniet faalt het systeem. Daarom worden ze ook vaak **reële-tijd systemen**[1] ('*real time systems*' in het Engels) genoemd.

  Bijvoorbeeld, een computer die een chemisch proces in een fabriek bewaakt, moet in staat zijn om binnen de 2 seconden een ventiel te openen indien de (continu gemeten) druk een zekere drempel overschrijdt. Kan het systeem niet binnen die 2 seconden reageren, dan zal het reactorvat exploderen.

  Reële-tijd systemen zullen meestal beperkt in omvang zijn. Het BP zal weinig diensten aanbieden in vergelijking met een systeem van algemeen nut. Voor heel wat diensten kan men moeilijk of helemaal niet berekenen hoeveel tijd het verlenen van deze dienst precies zal vergen. Bijvoorbeeld, indien het gebruikersprogramma aan het BP (van algemeen nut) vraagt (via een supervisie-oproep) om een gedeelte van een bestand in te lezen, dat zal het BP daar meer of minder tijd voor nodig hebben, afhankelijk van waar de lees/schrijfkop zich op dat ogenblik bevindt, hoe het bestand bijgehouden wordt of schijf, hoe groot het in te lezen gedeelte is, enz. Onzekerheden over de tijdsduur zijn moeilijk te verzoenen met de vaste tijdslimieten waaraan reële-tijd systemen moeten kunnen voldoen. Daarom zullen deze systemen alleen het allernoodzakelijkste bevatten.

De systemen van **algemeen nut** kunnen we verder nog onderverdelen in twee sub-klassen:

- de **niet-interactieve** besturingssystemen:

  deze noemt men vaak systemen met **stapelverwerking** (in het Engels '*batch systems*'); hier kan de programmeur tijdens de uitvoering niet rechtstreeks interageren met zijn programma;

  De benaming 'stapelverwerking' dateert nog uit de tijd dat programma's via ponskaarten moesten ingelezen worden. De programmeur overhandigde zijn pak ponskaarten aan de operator die de pakjes sorteerde (stapelde) volgens de gebruikte programmeertaal. Indien één van de stapeltjes groot genoeg was, werd de stapel (batch) in het computersysteem ingelezen. Bij de allereerste systemen werden die programma's in monoprogrammatie (d.w.z. na elkaar) uitgevoerd. Vandaag wordt ook bij systemen met stapelverwerking multiprogrammatie toegepast. De programma's worden ook niet langer via ponskaarten ingebracht, maar

1. Sommige besturingsprogramma's voor algemeen nut hebben ook faciliteiten om ***bepaalde taken*** in 'reële tijd' (d.w.z. binnen vaste tijdslimieten) uit te voeren. Nochtans noemt men dergelijke systemen geen reële-tijd systemen.

via terminals, zodat er van fysische 'stapeltjes' geen sprake meer is. Logisch gezien is er nog steeds een stapel: wanneer de gebruiker een programma wil laten uitvoeren moet hij/zij dit aan het BP aanbieden ('*submit*' in het Engels). Het BP zal niet onmiddellijk met de uitvoering beginnen maar het programma ergens in een wachtrij plaatsen ('*job queue*' in het Engels), en pas later de uitvoering starten. Het is dus mogelijk dat de uitvoering pas begint op een ogenblik dat de gebruiker reeds lang naar huis is gegaan. Vandaar dat interactie met de gebruiker niet mogelijk is. Alle gegevens die het programma nodig heeft moeten reeds in de computer aanwezig zijn (bijv. in een bestand).

- de **interactieve** besturingssystemen:

  hier kan de gebruiker interactief commando's of gegevens intypen via een terminal;
  - op ***één-gebruiker*** systemen ('*single-user systems*' in het Engels) kan slechts één gebruiker tegelijkertijd werken (vb. DOS); merk op dat het mogelijk is dat verschillende programma's gelijktijdig in uitvoering zijn, zoals dit mogelijk is bij de opvolgers van DOS, namelijk OS/2, Windows95(98)(NT)(XP);
  - op ***meerdere-gebruikers*** systemen ('*multi-user systems*' in het Engels), ook wel *time-sharing* systemen genoemd, kunnen meerdere gebruikers tegelijkertijd werken. Aangezien elke gebruiker graag zijn/haar programma's zo snel mogelijk uitgevoerd wil hebben, zal het besturingsprogramma de beschikbare tijd 'eerlijk' moeten verdelen over al deze gebruikers. Het zou onaanvaardbaar zijn indien de processor al zijn tijd zou besteden aan de uitvoering van enkele programma's van één gebruiker.

    Deze situatie is vergelijkbaar met een vergadering. Iedereen wil op tijd en stond aan het woord komen. Nochtans kan er maar één tegelijkertijd praten. Om te vermijden dat iemand de vergadering monopoliseert door een ellenlange monoloog op te voeren, krijgt de voorzitter het recht om iemand het woord te ontnemen en iemand anders aan het woord te laten.

    Op een gelijkaardige manier zal het besturingsprogramma de processor afnemen van een programma dat reeds een tijdje deze processor heeft gekregen, en de processor aan een ander programma laten verderwerken. Op deze wijze werkt de processor afwisselend voor een korte tijd (enkele tientallen of honderden milliseconden) aan de verschillende programma's en krijgt elke gebruiker de indruk dat hij/zij de processor voor zichzelf alleen heeft. De benaming 'timesharing' betekent dus dat de beschikbare computertijd verdeeld wordt over de aanwezige programma's.

    Het 'afnemen' van de processor (zoals we verder zullen zien) gebeurt via een wekker, die wanneer hij 'afloopt' een programma-onderbreking veroorzaakt. Zie ook paragraaf 4.8 en paragraaf D.6 in appendix D (pag. 218 e.v.).

Opgaven

1. Op welke wijze kan je de besturingsprogramma's opdelen?
2. Wat wordt er bedoeld met een 'interactief' programma?
3. Wat is de oorsprong van de term 'stapelverwerking' ('batch')?
4. Waarom bieden reële-tijd systemen meestal heel weinig diensten aan?
5. Verklaar het verschil tussen:
   - een timesharing systeem
   - een één-gebruiker systeem
   - een reële-tijd systeem
   - een systeem met stapelverwerking
6. Op welk soort systeem zou je graag werken indien je heel wat programma's moet ontwikkelen (d.i. ontwerpen en uittesten)? Waarom?
   - een systeem van algemeen nut
   - een reële-tijd systeem
   - een interactief systeem
   - een systeem met stapelverwerking

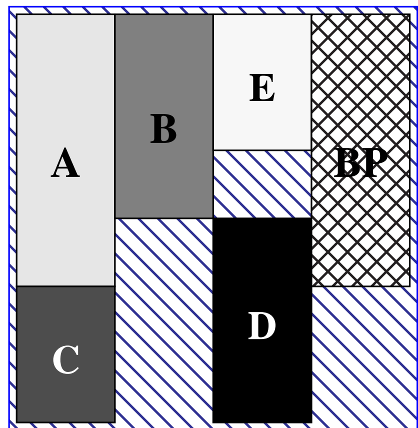
4.8 Taken/diensten v/h besturingsprogramma

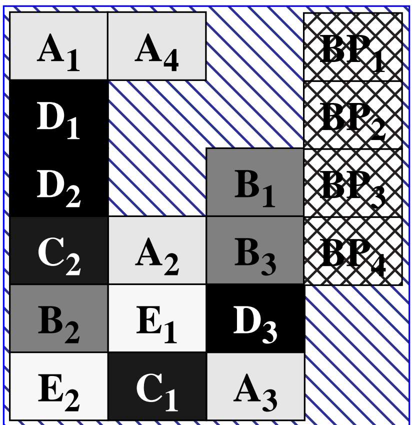
In deze paragraaf zullen we in het kort de taken beschrijven die door het besturingsprogramma vervuld worden:

1. **geheugenbeheer**, het BP moet beslissen welk stuk geheugen aan welk programma toegekend zal worden,
2. **processorbeheer**, het BP beslist aan welk programma de processor gegeven wordt en voor hoelang,
3. **beheer van in- en uitvoer**, het BP bestuurt zelf alle randapparaten,
4. **bestandenbeheer**, het BP houdt gegevens en/of programma's van gebruikers bij en stelt ze op aanvraag ter beschikking,
5. **informatiebeheer**, het BP houdt heel wat informatie bij en stelt deze informatie op aanvraag ter beschikking.

4.8.1 Geheugenbeheer

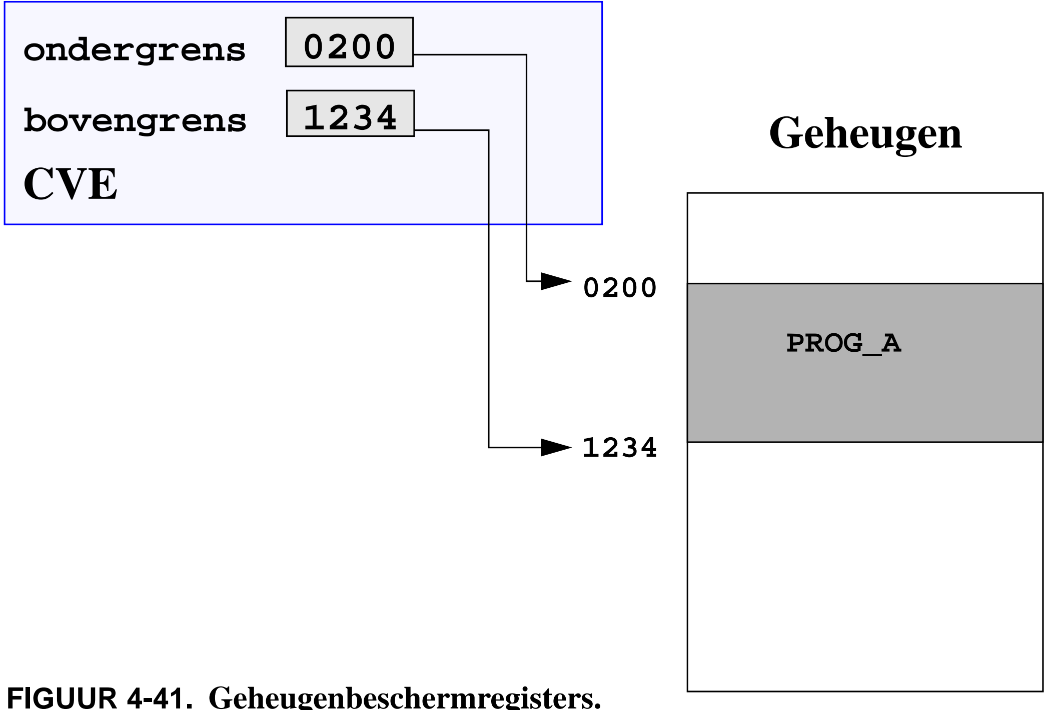
Multiprogrammatie betekent dat meerdere programma's gelijktijdig in het geheugen aanwezig zijn. Het besturingsprogramma zal dus het beschikbare geheugen eerlijk moeten verdelen over de aanwezige programma's. Hiervoor bestaan verschillende algoritmes, die in de cursus ***'Besturingssystemen'*** besproken zullen worden. In figuur 4-40 worden twee mogelijkheden getoond: links zijn de programma's aaneengesloten (ze zitten in opeenvolgende geheugenregisters); rechts zijn de programma's opgedeeld in kleine stukjes (pagina's) die elk ergens in het geheugen geplaatst worden. Het spreekt vanzelf dat in dit geval de hardware zal moeten aangepast worden om deze 'verspreide' stukjes toch als een geheel te laten voorkomen.





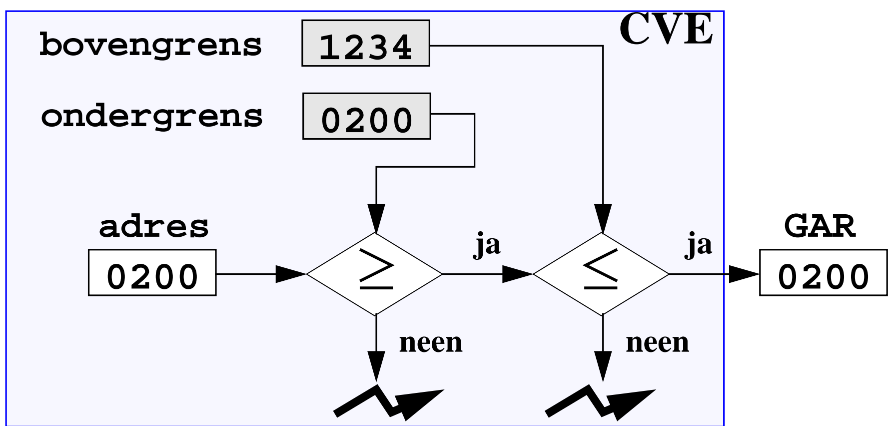
FIGUUR 4-40. Geheugentoekenning aan verschillende programma's.

Aangezien er verschillende programma's gelijktijdig in het geheugen aanwezig zijn, moet er vermeden worden dat tijdens de uitvoering van een programma (bv. **PROG_A**) geheugenregisters gewijzigd worden die behoren tot een ander programma (bv. **PROG_B**). De hardware zal opnieuw aangepast moeten worden. Indien de programma's aaneengesloten in het geheugen zitten, is het voldoende dat de processor uitgebreid wordt met twee speciale **geheugenbeschermingsregisters**, die de onder- en bovengrens van het toegekende geheugen aanduiden. (Zie ook figuur 4-41.)



FIGUUR 4-41. Geheugenbeschermregisters.

Het invullen van deze registers gebeurt door het BP via speciale (geprivilegieerde!) instructies. De hardware zal vóór een adres in het **GAR** geplaatst wordt, eerst dat adres vergelijken met de waarden in die twee registers. Valt het buiten het interval, dan wordt een adresseringsfout (d.m.v. een PO) gemeld. Bijvoorbeeld, bij **PROG_A** zal **HIA R3,400** kunnen uitgevoerd worden aangezien $200 \leq 400 \leq 1234$. Een **BIG R6,4000**, daarentegen, zal aanleiding geven tot een programma-onderbreking. (Zie figuur 4-42.)



FIGUUR 4-42. Elk geheugenadres wordt door de hardware gecontrolleerd.

Het BP zal in een tabel bijhouden welk deel van geheugen aan welk programma is toegekend. (Zie tabel 4-9.) Wanneer een programma afgewerkt is, zal de plaats die het inneemt opnieuw vrijgegeven worden (en eventueel toegekend worden aan een ander programma). Programma's kunnen ook tijdens hun uitvoering extra geheugen aanvragen of vrijgeven via een supervisie-oproep.

TABEL 4-9. De geheugen-allocatie tabel.

Beginadres	Lengte	Toegekend aan …
0000	140	PROG_A
1140	1060	*vrij*
2200	2253	PROG_B
4453	76	*vrij*
4529	1271	PROG_C
5800	1150	PROG_D
6950	50	*vrij*
7000	3000	BP

4.8.2 Processorbeheer

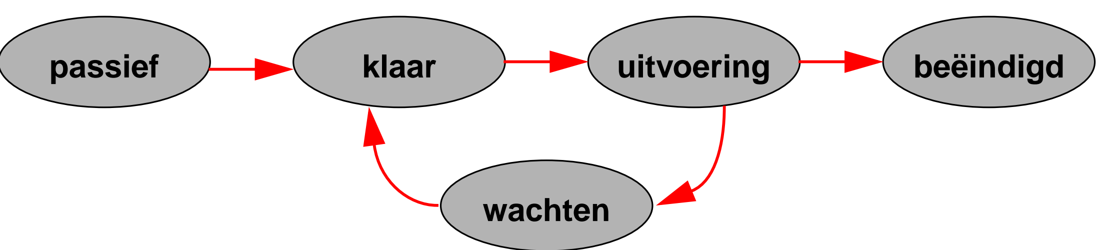
Wanneer meerdere programma's gelijktijdig in het geheugen zitten, is de kans groot dat na een PO, de processor zou kunnen gegeven worden aan meer dan één programma. Het BP zal dus een keuze moeten maken. Die keuze kan gebaseerd zijn op de hoogdringendheid (prioriteit) van het programma, of volgens een beurtrol uitgewerkt zijn. Ook hier weer zal het BP een tabel bijhouden waarin de toestand van elk programma dat in uitvoering is, bijgehouden wordt. Een programma kan zich immers in verschillende toestanden bevinden:

- wanneer het programma nog niet in het geheugen gebracht is, staat het nog **passief** te wachten in een wachtrij op een hulpgeheugen, meestal een magneetschijvengeheugen;
- het kan **paraat** staan in het centraal geheugen;
- het kan **in uitvoering** zijn (d.w.z. de processor is op dat ogenblik bezig instructies van dit programma uit te voeren);
- het kan **afgewerkt** zijn (d.w.z. alle bevelen zijn uitgevoerd, maar het BP moet nog het een en ander opruimen; bijvoorbeeld, het ingenomen geheugen moet nog vrijgegeven worden).

De paraatheidstoestand zelf kan nog onderverdeeld worden als volgt:

- het programma staat **klaar** om uitgevoerd te worden, en kan direct in actie treden als het de processor toegewezen krijgt;
- het programma is aan het **wachten**, bijvoorbeeld op de beëindiging van een in- of uitvoerbewerking voor het programma zelf (bijvoorbeeld, het programma heeft gegevens nodig die moeten ingelezen worden).

Figuur 4-43 toont de verschillende toestanden die een programma doorloopt, vanaf het ogenblik dat zijn uitvoering aangevraagd wordt, tot zijn beëindiging.



FIGUUR 4-43. Het toestandsdiagram van een programma.

Wanneer een keuze gemaakt moet worden, komen natuurlijk alleen de programma's die 'klaar' staan in aanmerking. Zie tabel 4-10. In dit voorbeeld zullen alleen **PROG_A**, **PROG_B** en **PROG_D** beschouwd worden. **PROG_A** zal de voorkeur krijgen omdat het een hogere prioriteit heeft.

In paragraaf 4.3.6 werd aangetoond dat op het ogenblik dat de uitvoering van een programma onderbroken wordt (door een PO), de inhoud van alle accumulatoren bewaard moet worden. We zullen deze waarden in dezelfde tabel bijhouden. (Elk programma heeft zijn eigen stapel, op de top hiervan staat de oude waarde van het programma-toestandswoord ($PTW_{0..9}$[1]).

TABEL 4-10. De programma-toestandstabel.

programma	toestand	prioriteit	R0	...	R9
PROG_A	*klaar*	10	0000012343	...	0000001100
PROG_B	*klaar*	7	0000002938	...	0000004450
PROG_C	*wacht (op schijf)*	3	0000000002	...	0000004600
PROG_D	*klaar*	1	0000000000	...	0000006800
PROG_E	*passief*	15	???	...	???

De toestand van een programma kan op twee manieren veranderen:

- het programma doet zelf een supervisie-oproep, waarbij gewacht zal moeten worden; bijvoorbeeld: het programma wil gegevens inlezen van schijf; aangezien deze niet onmiddellijk beschikbaar zijn, zal het programma in de wachttoestand komen;

1. In principe had de programma-onderbrekingsroutine de waarden van de registers kunnen wegbergen op de stapel; dit gebeurt hier niet, omdat stapeloverloop zou kunnen optreden.

- een randapparaat meldt via een programma-onderbreking dat het klaar is met een opdracht; bijvoorbeeld: de gegevens voor het programma zijn ingelezen; het programma zal terug in de 'klaar'-toestand komen.

Bij **timesharing** systemen werken verschillende gebruikers gelijktijdig met de computer. Het is erg gewenst dat alle gebruikers een 'eerlijk deel' van de computertijd krijgen. Een belangrijk begrip hierbij is de '**responstijd**', of de tijd die voorbijgaat tussen het geven van een opdracht (zoals het laden en uitvoeren van een programma), en het verschijnen van de eerste resultaten op scherm. Deze responstijd moet zo klein mogelijk zijn. Zoniet worden de gebruikers zenuwachtig: ***"Werkt die computer nu wel of niet?"*** Een goed timesharing systeem moet aan elke gebruiker de indruk geven dat hij/zij een persoonlijke computer ter beschikking heeft.

Om ervoor te zorgen dat elke gebruiker zijn 'eerlijk deel' van de beschikbare computertijd krijgt, zal het BP de processor maar voor een beperkte tijd geven aan elk gebruikersprogramma. Het BP moet er dus voor zorgen dat het na een bepaalde tijd, zeker de processor terugkrijgt.

Jij hebt een gelijkaardig probleem indien je in tijdnood bent en toch een aantal taken wil afwerken. Je zal je tijd moeten verdelen over de verschillende taken. Om te vermijden dat je teveel tijd besteedt aan een bepaalde taak kan je regelmatig op de klok kijken, maar de kans bestaat dat je je zo laat meeslepen dat je dit vergeet. Het is veel veiliger een wekker in te stellen. Nu kunnen er twee zaken gebeuren:

- ofwel is je taak afgewerkt voor de wekker afloopt; in dit geval zal je de wekker opnieuw instellen en verder gaan met de volgende taak;
- ofwel loopt de wekker af voor de taak beëindigd is; hierdoor word je gedwongen de huidige taak tijdelijk uit te stellen en een andere te beginnen (na het herinstellen van de wekker).

Een gelijkaardig mechanisme heeft het BP nodig. Hij moet er immers voor zorgen dat de processor niet te lang aan een bepaald programma blijft werken. De processor moet dus na een zekere tijd 'onderbroken' worden. Met andere woorden, het BP moet ervoor zorgen dat na een bepaalde tijd een PO optreedt.

Vooraleer het BP de processor laat werken aan een gebruikersprogramma, zal het ook een wekker instellen. Wanneer deze afloopt wordt een PO-vlag geplaatst, waardoor —wanneer de PO toegelaten is— een PO zal optreden. Op deze wijze kan het BP er steeds zeker van zijn dat het na een bepaalde tijd de processor terugkrijgt[1]en bijgevolg de mogelijkheid heeft om de processor nu aan een ander programma te laten werken. Dit is erg belangrijk, want het oorspronkelijke programma zou in een oneindige lus kunnen terechtgekomen zijn.

Laten we dit mechanisme even toepassen op de DRAMA-machine. We veronderstellen dat de wekker via opdrachtpoort **`P9`** geactiveerd wordt. De waarde die ingevuld wordt geeft aan na hoeveel 'tikken' van de wekker de PO_2-vlag geplaatst moet worden. Op de DRAMA-machine tikt de wekker elke µsec. In voorbeeld 4-10 wordt getoond hoe het BP de processor overdraagt aan een gebruikersprogramma: vóór de **`KTO`**-instructie wordt eerst de waarde **`10000`** in de

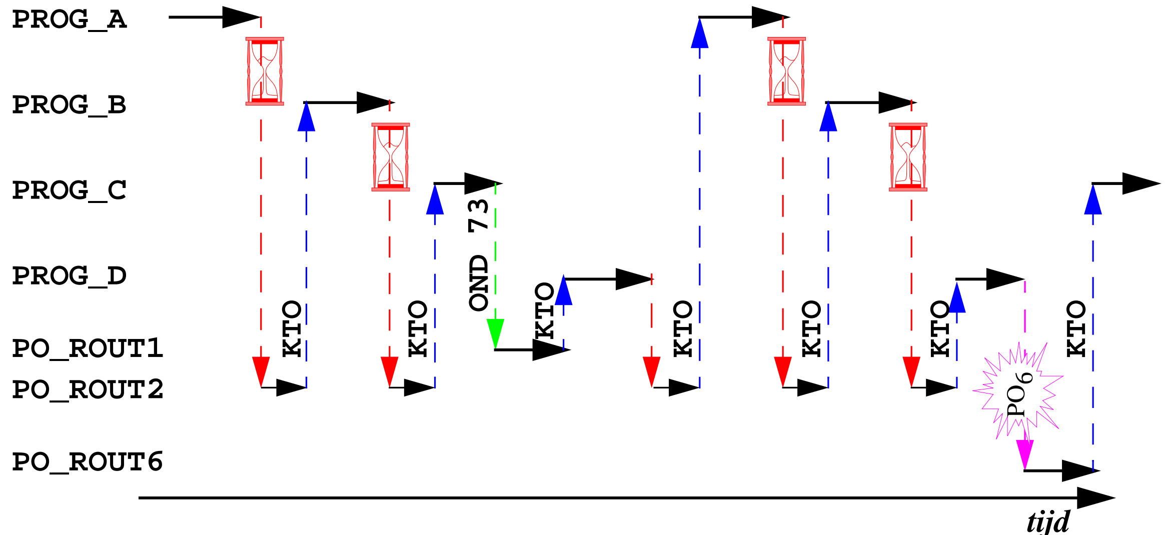
1. Indien een programma-onderbreking optreedt, wordt een routine uitgevoerd van het besturingsprogramma. Het adres van deze routine wordt in de overeenkomstige PO-vector bijgehouden.

opdrachtpoort van de wekker ingevuld. Na **10000** tikken (d.i. na 10 msec.) zal de PO_2-vlag geplaatst worden. De wekker kan afgezet worden door een negatieve waarde in te vullen. Voor een volledige beschrijving van de wekker zie paragraaf D.6 (op pag. 227).

Voorbeeld 4-10. *Het volgende fragment geeft aan hoe het besturingsprogramma de processor afstaat aan een gebruikersprogramma.*

```
           | Stel de wekker in
           HIA    R0,INTERVAL
           UTV    R0,P9
           | Herstel accumulatoren
           HIA    R0,BEWAAR+0
           HIA    R1,BEWAAR+1
           ...
           HIA    R9,BEWAAR+9     | PTW[0..9] op top stapel!
           KTO
  BEWAAR:  RESGR  10              | Plaats waar reg. bewaard
INTERVAL:  10000
```

Figuur 4-44 toont hoe de processor ‘afwisselend’ werkt aan de aanwezige programma’s..



FIGUUR 4-44. Het afwisselend uitvoeren van verschillende programma’s.

Na **INTERVAL** micro-seconden zal een PO_2 de processor afnemen van het programma dat in uitvoering is. De zandloper ( ) geeft aan wanneer de wekker afloopt (een PO_2 wordt aangevraagd). In de figuur is ook te zien dat **PROG_C** een supervisie-oproep doet (**OND 73** : inlezen van schijf). Dit programma zal dus voorlopig niet meer meedoen aan het beurtrollensysteem. Pas als de inlees-operatie voltooid is (PO_6) kan het weer meedingen naar de CVE.

Programma's die afgewerkt zijn kunnen hun uitvoering niet zelf beëindigen. (Het **STP**-bevel is geprivilegieerd!) Dus zullen ze aan het BP moeten vragen (via een supervisie-oproep) om beëindigd te worden. Het BP zal dan het programma verwijderen uit het geheugen, en uit de programma-toestandstabel.

4.8.3 Het beheer van de randapparaten

In paragraaf 4.4 is reeds uitvoerig aangetoond dat alle in- en uitvoerbewerkingen door het BP geregeld worden. Programma's moeten via supervisie-oproepen aan het BP vragen om dit in hun plaats te doen.

Bijvoorbeeld, indien het programma iets wil afbeelden op het scherm van de terminal, zal het een supervisie-oproep moeten uitvoeren:

```
| Druk 4*X + 10 af voor X := 1 ... 10
        HIA.w   R1,1          | X
LUS:    HIA     R0,0(R1+)
        VER.w   4
        OPT.w   10
        OND     2             | DRU
        OND     3             | NWL
        VGL.w   R1,10
        VSP     KLG,LUS
        OND     9999          | STP
```

Voor elk randapparaat zal het BP een tabel bijhouden die aangeeft wat de toestand is van het apparaat, '**klaar**' om een opdracht te ontvangen, of '**bezig**' met een opdracht, of '**defect**'. Zie ook tabel 4-11. Bovendien zal het BP ook voor elk apparaat dat bezig is de opdrachten bijhouden die het nog moet uitvoeren. Voor het toetsenbord worden de letters bijgehouden die reeds ingetypt zijn maar nog niet opgevraagd door een programma.

Uit de tabel blijkt dat de schijfbestuurder op dit ogenblik bezig is met het wegschrijven van gegevens voor programma **PROG_A**. Daarna zal een sector moeten ingelezen worden voor programma **PROG_C**. De gebruiker aan het toetsenbord heeft reeds twee lijnen ingetypt die nog niet opgevraagd zijn door een programma. De tweede schijf is defect en het scherm is bezig met het afbeelden van de symbolenrij '**priem = 97**'.

TABEL 4-11. De randapparaat-toestandstabel.

apparaat	toestand	opdrachten
scherm	***bezig***	`PROG_A` ***schrijf*** ‘priem = 97’
toetsenbord	***klaar***	***beschikbaar*** abcde -3000
drukker	***klaar***	—

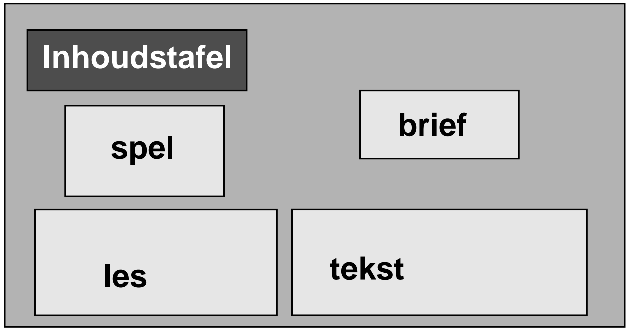
schijf	***bezig***	`PROG_A`
		schrijf (cil:4,sp:4,sect:7)
		…data …
		`PROG_C`
		lees (cil:2,sp:7,sect:2)
		adres:4900

schijf2	***defect***	—

4.8.4 Bestandenbeheer

Elke gebruiker bewaart heel wat informatie (zoals programma’s —de broncode en de vertaalde versie—, teksten, …). Het zou niet erg praktisch zijn mocht elke gebruiker zelf moeten bijhouden waar ergens (bijv. op welke schijf, in welke cilinder, in welk spoor, in welke sector) de informatie bewaard wordt. Het besturingsprogramma zal een hoog-niveau concept aanbieden aan de gebruiker: een ‘**bestand**’ (‘*file*’ in het Engels).

Een bestand is een hoeveelheid informatie (gegevens, programma’s, …) die samenhoort en via een gebruiksvriendelijke naam kan benoemd worden. (Je zou een bestand kunnen vergelijken met een boek: het heeft een naam, de informatie die erin staat is samengebundeld.) Zie ook figuur 4-45. De schijf bevat vier bestanden: **`tekst`**, **`brief`**, **`les`** en **`spel`**.



FIGUUR 4-45. Verschillende bestanden op schijf.

Het besturingsprogramma zal bijhouden waar en hoe deze bestanden bewaard worden. Op de schijf zal een 'inhoudstafel' voorzien worden. In het Engels wordt deze vaak een '*directory*' genoemd. De inhoudstafel bevat o.a. de naam van elk bestand, de lengte, en de plaats waar het opgeslagen is. (Zie ook tabel 4-12.).

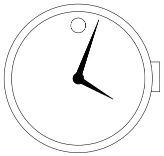
TABEL 4-12. De inhoudstafel van de schijf.

Naam	Lengte	Plaats
`tekst`	`2000`	`cil:8/spoor:3/sectoren:2-7`
`brief`	`500`	`cil:1/spoor:5/sectoren:8-9`
`les`	`1467`	`cil:7/spoor:4/sectoren:4-8`
`spel`	`500`	`cil:3/spoor:3/sectoren:1-2`

Een gebruikersprogramma kan aan het BP vragen om gegevens uit het bestand '**`tekst`**' in te lezen of erin weg te schrijven, of om het programma '**`spel`**' te laden en uit te voeren. Dit 'aanvragen' gebeurt ook weer via supervisie-oproepen.

4.8.5 Informatiebeheer

Het besturingsprogramma zal ten behoeve van de gebruikersprogramma's heel wat informatie bewaren. Zo kan een programma aan het BP vragen wat de huidige datum is, hoe laat het op dit ogenblik is (zie figuur 4-46), hoeveel programma's in uitvoering zijn, welke gebruikers op dit ogenblik op de computer werken, hoeveel schijfruimte nog vrij is, hoelang de processor al aan een bepaald programma gewerkt heeft, … Welke informatie bijgehouden wordt en (via een supervisie-oproep) opgevraagd kan worden, is sterk afhankelijk van het BP.



FIGUUR 4-46. Het besturingsprogramma houdt de huidige datum en de tijd bij.

Hoe houdt het BP de datum en tijd bij? Dit gebeurt m.b.v. een **klok**. Er bestaan twee soorten computerklokken:

- De eenvoudigste computerklokken bestaan uit een circuit dat op regelmatige tijdstippen een programma-onderbreking (ook wel een klok-tik genoemd) aanvraagt: bijvoorbeeld om de 20 msec (d.w.z. met de frequentie van de lijnspanning: 50 Hz). Het BP zal bij het opstarten van de computer aan de operator vragen wat de huidige datum en tijd is, en daarna het aantal PO's vanwege de klok blijven tellen. Uit de referentie-datum en -tijd en het aantal voorbije onderbrekingen kan het BP dan de huidige datum en tijd berekenen. De tijd is niet erg nauwkeurig omdat de frequentie van de lijnspanning wat kan fluctueren, en omdat de computer soms een PO-aanvraag van de klok zal uitstellen, zodat er 'tikken' verloren gaan.

- De tweede soort bestaat uit kristal-oscillator, die —wanneer hij onder spanning staat— een periodisch signaal van zeer hoge nauwkeurigheid aflevert. Dat signaal wordt dan naar een 'tel-circuit' geleid. De inhoud van dit tel-register kan uitgelezen worden, en door het BP omgezet worden naar de huidige datum en tijd, in de veronderstelling natuurlijk dat het tel-circuit (eenmalig) geïnitialiseerd was met een juiste referentie-waarde (cfr. verder). Om te vermijden dat bij het uitvallen van de spanning de huidige datum en tijd verloren gaat, zullen veel computers een batterij voorzien die continu voor de nodige spanning zorgt.

Ook de DRAMA-machine bevat een klok (van de tweede soort) wiens waarde kan uitgelezen worden uit poort P_8. De waarde is het aantal tienden van een seconden die verlopen zijn sinds ***1 januari 1990, middernacht***. Mits enige berekeningen en kennis van de kalender, kan het BP hieruit zowel de **datum** als de **tijd** afleiden. Voor een volledige beschrijving van de klok: zie paragraaf D.5 (op pag. 227).

De beschikbare informatie kan door elk programma opgevraagd worden via verschillende **supervisie-oproepen**. Bijvoorbeeld, `OND 100` geeft de datum (dag, maand, jaar) terug in registers **`R0`**, **`R1`** en **`R2`**. Bij `OND 101` zal het BP de huidige tijd (uur, minuten, seconden) in registers **`R0`**, **`R1`** en **`R2`** plaatsen.

Sommige programma's werken samen aan één grote taak en moeten daarom af en toe met elkaar kunnen '**communiceren**' (d.i. informatie uitwisselen). Bijvoorbeeld, het ene programma berekent opeenvolgende priemgetallen terwijl het andere programma die priemgetallen gebruikt om een groot getal te ontbinden in zijn factoren. Het ene programma moet dus die priemgetallen kunnen 'doorgeven' aan het andere programma. Het BP zal er ook voor zorgen dat dit op een ordelijke wijze gebeurt.

4.8.6 Boekhouding

Het BP houdt een **boekhouding** bij over alle programma's die worden uitgevoerd. De hoeveelheid informatie die hierin voorkomt verschilt natuurlijk van BP tot BP.

a. Ten behoeve van de facturatie:

Voor elk programma zou het BP kunnen bijhouden hoeveel geheugen het nodig had, hoeveel 'processor-tijd' het gekregen heeft, hoe dikwijls het beroep deed op de hulpgeheugens, enz. Op commercieel uitgebate systemen kan die boekhouding gebruikt worden om een billijke ***factuur*** op te maken voor de gebruiker die de uitvoering van het programma heeft aangevraagd.

b. Statistieken t.b.v. de computersysteem-planning

Ook over de werking van het computersysteem wordt heel wat **statistische informatie** verzameld, zoals: gedurende hoeveel tijd de processor werkloos is, hoe intensief de randapparaten gebruikt worden, hoe dikwijls het centraal geheugen volledig in gebruik was, enz. Deze informatie is zeer nuttig voor de systeembeheerder, omdat hij/zij kan nagaan hoe 'goed' het systeem werkt, en of er ***knelpunten*** ('*bottlenecks*' in het Engels) bestaan. Bijvoorbeeld, wanneer zou blijken dat het centraal geheugen bijna constant volledig in gebruik is, moet ernstig overwogen worden om dit uit te breiden.

De systeembeheerder zal deze informatie ook gebruiken om allerlei parameters in te stellen zodat het computersysteem goed afgeregeld wordt ('*tuning*' in het Engels). De omstandigheden waarin het computersysteem gebruikt wordt (welk soort programma's gewoonlijk uitgevoerd worden, hoeveel gelijktijdige gebruikers er zijn, enz.) bepalen mee de optimale waarde voor deze parameters. Bijvoorbeeld, soms kan de operator de multiprogrammatiegraad instellen (d.w.z. het aantal programma's dat maximaal samen in het geheugen mag gebracht worden). Indien dit te laag is, zal de processor vaak werkloos zijn, tenzij de programma's erg reken-intensief zijn. Indien de multiprogrammatiegraad te hoog ingesteld is, zal het geheugen een knelpunt worden. Zo zijn er wel tientallen parameters die kunnen ingesteld worden en die elk een invloed zullen hebben op de performantie van het systeem.

Opgaven

1. Waartoe dienen geheugenbeschermregisters? Zouden deze ook nuttig zijn wanneer het BP enkel monoprogrammatie ondersteunt (d.w.z. slechts één gebruikersprogramma tegelijkertijd in het geheugen toelaat)?
2. Bij multiprogrammatie worden verschillende programma's gelijktijdig in het geheugen gehouden. Welke informatie moet het BP bijhouden om het geheugen te kunnen beheren?
3. Een programma kan zich in verschillende toestanden bevinden. Welke? Schets het diagram dat de overgangen tussen deze toestanden aangeeft. Geef voor elke overgang een mogelijke oorzaak van die overgang. Ga na of het BP steeds op de hoogte gebracht wordt van een toestandsverandering.
4. Wat is het nut van een wekker? Hoe wordt deze op de DRAMA-machine gebruikt?
5. Wat wordt er bedoeld met een 'beurtrollensysteem'?
6. Wat betekent 'responstijd'? Waarom is dit belangrijk op een time-sharing systeem?
7. Wat is een bestand? Waarom biedt een BP dit concept aan?
8. Welke informatie bevat de inhoudstafel van een schijf?
9. Het BP houdt heel wat informatie bij. Geef hiervan enkele voorbeelden.
10. Hoe bepaalt het BP welke dag en hoe laat het is?
11. Waarom houdt het BP een boekhouding en statistieken bij?
12. Geef een bondig overzicht van de taken die door een besturingssysteem worden vervuld.
13. Schrijf twee programma's **`PRIEM1`** en **`PRIEM2`**, die respectievelijk de priemgetallen tussen **`2`** en **`100`** en de priemgetallen tussen **`100`** en **`200`** berekenen. Wissel hun uitvoering af door gebruik te maken van de wekker.

    Let op:

    - Zorg dat je geen vitale informatie verliest die op de stapel staat (kopiëer die eerst naar een veilige plaats).
    - De wekker moet geplaatst zijn vóór de uitvoering van een van de programma's gestart wordt.
    - Na de eerste PO zal je zelf een $\text{PTW}_{0\text{-}9}$ moeten samenstellen (en op de top van de stapel plaatsen) voor het andere programma.

    Experimenteer met de waarde die in de wekker-poort wordt ingevuld.

4.9 Kosten/baten aspecten

In de laatste paragraaf van dit hoofdstuk overlopen we nog eens de voordelen maar ook de nadelen van een besturingssysteem. Tenslotte maken we een berekening over het uiteindelijke 'rendement' van een computer.

4.9.1 De baten

Een BP verhoogt de **gebruiksvriendelijkheid** van het computersysteem. Immers, het ontlast de gebruiker van lastige en complexe taken.

De **correcte werking** wordt ook verbeterd. Een gebruikersprogramma wordt immers gedwongen om regelmatig een beroep te doen op het BP, vanwege het feit dat alle 'gevaarlijke' instructies geprivilegieerd zijn, en dus niet in probleemtoestand mogen uitgevoerd worden. Enkele voorbeelden:

- een gebruikersprogramma kan de processor niet in de halttoestand brengen;
- een gebruikersprogramma kan niet rechtstreeks op een schijvengeheugen schrijven (waardoor vitale informatie beschadigd zou kunnen worden) aangezien de invoer/uitvoer bewerkingen geprivilegieerd zijn;
- een gebruikersprogramma kan niet aan het gedeelte van het geheugen dat gebruikt wordt door het BP of door een ander programma, aangezien de geheugenbeheereenheid (met de geheugenbeschermingsregisters) dit verhindert en het invullen van deze beschermingsregisters geprivilegieerde bewerkingen zijn;
- een gebruikersprogramma kan de 'klok' niet instellen omdat ook dit een geprivilegieerde bewerking is.

Tenslotte zorgt een BP ervoor dat het computersysteem veel **efficiënter** gebruikt wordt, vooral door de techniek van multiprogrammatie. Hierdoor is de processor minder werkloos en krijgen de randapparaten ook sneller een nieuwe opdracht. De performantie van het computersysteem verhoogt; m.a.w. het presteert meer in dezelfde tijd.

4.9.2 De kosten

Besturingsprogramma's kunnen zeer uitgebreid en gesofistikeerd zijn. Voor een goede werking vereist dit heel wat programmatuur, maar moet ook de apparatuur erop voorzien zijn. Bijvoorbeeld, zonder een goed uitgebouwd programma-onderbrekingsmechanisme, geprivilegieerde bevelen, geheugenbeschermingsregisters, enz., kan je een BP wel vergeten.

De introductie van een BP brengt drie soorten kosten met zich mee, die we hierna kort zullen bespreken:

a. apparatuurkosten
b. programmatuurkosten,
c. overhead in tijd

(a) Apparatuurkosten

De processor is veel **complexer** geworden: er is een gesofisticeerd programma-onderbrekingsmechanisme nodig met prioriteiten en maskers; de processor moet zich in verschillende toestanden kunnen bevinden, en sommige bevelen moeten geprivilegieerd zijn, …

Een BP is al gauw enkele tientallen *mega*-bytes groot. Bovendien worden bij multiprogrammatie verschillende programma's tegelijkertijd in het geheugen gehouden. Bijgevolg is er een **groot centraal geheugen** nodig.

Er zijn voldoende **randapparaten** nodig, om simultaan te kunnen werken. Op grotere systemen heeft men '**kanaalbestuurders**' nodig, om te vermijden dat de processor teveel tijd moet besteden aan invoer en uitvoer. Zoniet zullen de randapparaten snel het knelpunt worden.

(b) Programmatuurkosten

De ontwikkeling van een modern BP vraagt vele manjaren werk. Bijvoorbeeld, de kosten van het BP voor de IBM/370 overtroffen de apparatuur-ontwikkelingskosten.

Uit (a) en (b) mogen we dus besluiten dat om de computer efficiënter en eenvoudiger te kunnen gebruiken, hij eerst een flink stuk **duurder** gemaakt wordt.

(c) Overhead aan tijd

Alhoewel een BP de efficiëntie van een computersysteem verhoogt, zal er toch af en toe tijd verspild worden aan extra instructies. In het Engels wordt dit met de term '*overhead*' aangeduid. Enkele voorbeelden illustreren deze '**verspilling**':

- Telkens een programma-onderbreking optreedt, zal de behandelingsroutine de inhouden van de rekenregisters bewaren in het geheugen, en later terug herstellen.
- Telkens de processor 'afgenomen' wordt van een programma (door een PO vanwege de wekker) en doorgegeven aan een ander programma, moeten tijdens die 'programmawisseling' heel wat instructies uitgevoerd worden: de toestand van het onderbroken programma moet veilig weggeborgen worden en de toestand van het andere programma moet hersteld worden.

  Zeer reken-intensieve programma's (die weinig invoer en/of uitvoer nodig hebben) zullen dus sneller in monoprogrammatie dan in multiprogrammatie uitgevoerd worden.

- Een BP zal steeds extra testen toevoegen om er zeker van te zijn dat alles correct blijft lopen: bijvoorbeeld, wanneer een gebruikersprogramma aan het BP vraagt om iets weg te schrijven, dan zal het BP eerst nagaan of hierdoor geen vitale informatie overschreven wordt. Indien dit zo is, zal het BP op die vraag niet ingaan. Geen enkel gebruikersprogramma wordt vertrouwd door het BP: de programmeur van het programma kon immers kwade bedoelingen hebben, of het programma kan foutief zijn.

Deze 'overhead' wordt natuurlijk meer dan gecompenseerd door de ***'tijdswinst'*** die een BP kan realiseren:

- de randapparaten kunnen gelijktijdig bezig zijn met de processor,
- door multiprogrammatie zal de processor minder werkloos zijn,
- het BP kan optimisaties doorvoeren (zoals het 'vooraf inlezen' of 'pas later wegschrijven').

Het is realistisch te stellen dat gedurende het grootste deel van de tijd (bijvoorbeeld 80% of meer) de processor bezig is met het BP. Immers, behalve voor het 'pure rekenwerk', moet een gebruikersprogramma steeds beroep doen op het BP. Dit is echter niet allemaal 'overhead': mocht er geen BP geweest zijn, dan zou elk gebruikersprogramma zelf zijn invoer/uitvoer moeten organiseren, enz. Bovendien zou in monoprogrammatie het rendement zeer laag zijn. Het relativeert echter (nogmaals) het begrip MIPS als maat voor de performantie van een computersysteem.

4.9.3 Het uiteindelijke rendement

Laten we even uitgaan van de veronderstelling dat

- 80% van de processortijd besteed wordt aan het BP, en
- 20% aan gebruikersprogramma's.

Anderzijds dient het grootste gedeelte van de apparatuur voor het 'beheer' van de apparatuur zelf (klokken, kanaalbestuurders, voorgeheugens, foutendetectie, geheugenbeheer-eenheden, PO-mechanisme, …). Slechts een fractie dient voor het uitvoeren van bevelen (d.i. de eigenlijke bestaansreden voor een computersysteem). Een goede schatting is de volgende:

- 80% van de apparatuur dient voor het 'beheer' van die apparatuur
- 20% dient voor het uitvoeren van de bevelen.

Uit de vorige twee cijfergegevens volgt dat slechts 4% van de middelen gebruikt wordt voor het uitvoeren van gebruikersprogramma's.

Ook gebruikersprogramma's gebruiken niet al hun tijd voor 'berekeningen'. Bijvoorbeeld:

- 75% van de tijd wordt besteed aan het 'organiseren' van het programma (zoals: programmalussen, indexatie, stapels, …)
- 25% voor het eigenlijke rekenwerk.

Bijgevolg kan men stellen dat slechts 1% van de middelen dient voor het echte rekenwerk.

Opgaven

1. Welke voordelen brengt het aanwenden van een BP met zich mee?
2. Een besturingsprogramma brengt ook extra kosten met zich mee. Welke?
3. Wegen de baten op tegen de kosten die een BP met zich meebrengt?
4. Wat wordt er eigenlijk met 'overhead' bedoeld?
5. Verklaar de volgende uitspraak: "Slechts 1% van de middelen dient voor het echte rekenwerk."

Appendix A
DRAMA-Machinedefinities

Tabel A-1 geeft een overzicht van de beschikbare functiecodes samen met hun overeenkomstige mnemotechnische code en instructieklasse. **LEZ**, **DRU** en **NWL** zijn eigenlijk geen echte instructies, maar worden toch voorzien indien geen gebruik gemaakt wordt van het programmaonderbrekingsmechanisme; zoniet zijn alleen **INV** en **UTV** beschikbaar. Je kan aan de simulator via een optie opgeven wat je precies verkiest. We verwijzen hiervoor naar de beschrijving van de simulator.

Tabel A-2 geeft de verschillende instructie-klassen. De opdeling is een beetje artificieel. Indien een veld op een andere manier gebruikt wordt, geeft dit aanleiding tot een nieuwe klasse. Het **modus**-veld kan normaal de waarden **1..4|1..6** aannemen; **modus*** heeft als mogelijke waarden **2..3|1..6**.

Tabel A-3 geeft meer informatie over de verschillende klassen. Het beschrijft de mogelijke '***interpretaties***', de mogelijke operanden, en of de tweede operand van de instructie een register operand mag zijn.

Tabel A-4 beschrijft de betekenis van de verschillende modus-cijfers. Ongeldig betekent dat deze waarde niet mag gebruikt worden; '***niet van toepassing***' is een waarde die gebruikt wordt indien de instructie geen modus-veld nodig heeft (bijvoorbeeld **LEZ**).

Tabel A-5 toont hoe de mnemotechnische voorwaarden vertaald worden naar een decimaal cijfer.

De betekenis van deze voorwaarden is weergegeven in tabel A-6. **CC** is de conditiecode, **OVI** de overloopindicator, en **SOI** de stapeloverloopindicator; **CC**, **OVI** en **SOI** behoren tot het programmatoestandswoord (**PTW**).

Tabel A-7 toont hoe de mnemotechnische onderbrekingsnummers vertaald worden naar een decimaal cijfer.

Tabel A-8 toont hoe de mnemotechnische register- en poortnummers vertaald worden tot een decimaal getal. Poortnummers kunnen de waarden **P0** t.e.m. **P9999** aannemen.

Tabellen A-9 en A-10 geven voor de verschillende DRAMA-bevelen de semantiek weer.

Tabel A-11 geeft aan hoe willekeurige letters en symbolen d.m.v. een drie-cijferige ASCII-code worden voorgesteld in het geheugen.

TABEL A-1. De DRAMA-machine-instructietabel.

Mnemo-technische Code	Functie-code	Instructie-klasse
HIA	11	K_1
BIG	12	K_1^*
OPT	21	K_1
AFT	22	K_1
VER	23	K_1
DEL	24	K_1
MOD	25	K_1
VGL	31	K_1
SPR	32	K_4^*
VSP	33	K_2^*
SBR	41	K_4^*
KTG	42	K_6

Mnemo-technische Code	Functie-code	Instructie-klasse
MKL	51	K_5
MKH	52	K_5
TSM	53	K_5
TSO	54	K_5
OND	61	K_7
KTO	62	K_6
LEZ	71	K_6
DRU	72	K_6
NWL	73	K_6
INV	81	K_3
UTV	82	K_3
STP	99	K_6

TABEL A-2. De DRAMA-instructieklassen

Klasse	Bevelenopmaak				
K_1	opcode	modus	acc	idx	operand
K_1^*	opcode	modus*	acc	idx	operand
K_2^*	opcode	modus*	vrw	idx	operand
K_3	opcode	9 9	acc	9	poortnr
K_4^*	opcode	modus*	9	idx	operand
K_5	opcode	9 9	ond	9	9 9 9 9
K_6	opcode	9 9	9	9	9 9 9 9
K_7	opcode	9 9	9	9	operand

TABEL A-3. Informatie tabel over de instructieklassen

Instructie-klasse	Mogelijke interpre-taties	Operanden	Register operand mogelijk?
K_1	a, w, d, i	[acc,] adres	ja
K_1^*	d, i	[acc,] adres	ja
K_2^*	d, i	[voorw,] adres	ja
K3	—	[acc,] prt-nr	neen
K_4^*	d, i	adres	ja
K_5	—	ond-nr	neen
K_6	—	—	neen
K_7	—	waarde	neen

TABEL A-4. De mogelijke waarden voor de twee modus-cijfers

1ste modus-cijfer		2de modus-cijfer	
waarde:	**betekenis**	**waarde:**	**betekenis**
0:	*ongeldig*	*0:*	*ongeldig*
1:	waarde	1:	geen indexatie
2:	waarde (modulo 10.000)	2:	indexatie
3:	adres	3:	indexatie met pre-increment
4:	indirect adres	4:	indexatie met post-increment
5:	*ongeldig*	5:	indexatie met pre-decrement
6:	*ongeldig*	6:	indexatie met post-decrement
7:	*ongeldig*	*7:*	*ongeldig*
8:	*ongeldig*	*8:*	*ongeldig*
9:	niet van toepassing	9:	niet van toepassing

TABEL A-5. De vertaling van de VSP-voorwaarden

Voorwaarde	Equivalente voorwaarde	Inwendige code
SO		0
GEL	NUL	1
GRG	NNEG	2
KLG	NPOS	3
OVL		4
GOVL		5
GR	POS	6
KL	NEG	7
NGEL	NNUL	8
GSO		9

TABEL A-6. De betekenis van de VSP-voorwaarden

Voorw.	Betekenis		Voorw.	Betekenis	
GEL	Gelijk	(CC = 0)	NUL	Nul	(CC = 0)
GR	Groter dan	(CC = 1)	POS	Positief	(CC = 1)
KL	Kleiner dan	(CC = 2)	NEG	Negatief	(CC = 2)
KLG	Kleiner of gelijk	(CC = 0 of 2)	NPOS	Niet positief	(CC = 0 of 2)
GRG	Groter of gelijk	(CC = 0 of 1)	NNEG	Niet negatief	(CC = 0 of 1)
NGEL	Niet gelijk	(CC = 1 of 2)	NNUL	Niet nul	(CC = 1 of 2)
OVL	Overloop	(OVI = 1)	GOVL	Geen overloop	(OVI = 0)
SO	Stapeloverloop	(SOI = 1)	GSO	Geen stapeloverloop	(SOI = 0)

TABEL A-7. De vertaling van de onderbrekingsnummers

Onderbrekings-nummer of masker	Vertaling
G	0
GPF	1
WEK	2
DRK	3
IN	4
UIT	5
SCH	6
OVL	7
SPL	8
MFT	9

TABEL A-8. De vertaling van de register- en poortnummers

Register-nummer	Vertaling	Poort-nummer	Vertaling
R0	**0**	P0	**0000**
R1	**1**	P1	**0001**
R2	**2**	P2	**0002**
R3	**3**	P3	**0003**
R4	**4**	P4	**0004**
R5	**5**	P5	**0005**
R6	**6**	P6	**0006**
R7	**7**	...	**...**
R8	**8**	P100	**0100**
R9	**9**	...	**...**
SR	**9**	P9999	**9999**

In de volgende twee tabellen worden de bevelen in symbolische vorm getoond omdat dit gebruiksvriendelijker is (in werkelijkheid zouden er decimale getallen moeten staan). Indien indexering van toepassing is, moet de operand op de gepaste wijze aangepast worden. We laten deze aanpassing over aan de lezer. De betekenis van de gebruikte symbolen wordt in de volgende tabel samengevat:

Geheugen[Adres]:	Geheugenregister met als adres: 'Adres' modulo 10,000
R9:	stapelwijzer (**SW**)
-R9:	verlaag de inhoud van **R9** met 1 en gebruik daarna **R9** (d.i. pre-decrement)
R9+:	gebruik inhoud van **R9** en verhoog **R9** daarna met 1 (d.i. post-increment)
a ← b:	a krijgt een nieuwe waarde: (de inhoud van) b
a ↔ b:	de waarde van a wordt vergeleken met de waarde van b

De kolom gemerkt '**Priv?**' duidt aan of de instructie geprivilegieerd is (**P**) of niet (leeg). De afkorting '**O**' staat voor 'onbestaand' (in feite zijn deze instructies pseudo-bevelen die gedefinieerd zijn als een macro).

TABEL A-9. De drama-bevelen en hun betekenis (semantiek)

Bevel		Priv?	Betekenis (semantiek)
HIA.w	Ri,Waarde		Ri ← Waarde
HIA.a	Ri,Adres		Ri ← Adres
HIA.d	Ri,Adres		Ri ← Geheugen[Adres]
HIA.i	Ri,Adres		Ri ← Geheugen[Geheugen[Adres]]
BIG.d	Ri,Adres		Geheugen[Adres] ← Ri
BIG.i	Ri,Adres		Geheugen[Geheugen[Adres]] ← Ri
OPT.w	Ri,Waarde		Ri ← Ri + Waarde
OPT.a	Ri,Adres		Ri ← Ri + Adres
OPT.d	Ri,Adres		Ri ← Ri + Geheugen[Adres]
OPT.i	Ri,Adres		Ri ← Ri + Geheugen[Geheugen[Adres]]
AFT.w	Ri,Waarde		Ri ← Ri - Waarde
VER.w	Ri,Waarde		Ri ← Ri * Waarde
DEL.w	Ri,Waarde		Ri ← Ri div Waarde
MOD.w	Ri,Waarde		Ri ← Ri mod Waarde
	de overige modi zijn analoog aan deze van het OPT-bevel		
VGL.w	Ri,Waarde		CC ← (Ri ↔ Waarde)
VGL.a	Ri,Adres		CC ← (Ri ↔ Adres)
VGL.d	Ri,Adres		CC ← (Ri ↔ Geheugen[Adres])
VGL.i	Ri,Adres		CC ← (Ri ↔ Geheugen[Geheugen[Adres]])

TABEL A-10. De drama-bevelen en hun betekenis (semantiek) (vervolg)

Bevel		Priv?	Betekenis (semantiek)
SPR.d	Adres		BT ← Adres
SPR.i	Adres		BT ← Geheugen[Adres]
VSP.d	NNEG,Adres		if (CC ≤ 1) then BT ← Adres
VSP.i	NNEG,Adres		if (CC ≤ 1) then BT ← Geheugen[Adres]
	Voor de andere voorwaarden zie tabel A-6		
SBR.d	Adres		Geheugen[-R9] ← BT; BT ← Adres
SBR.i	Adres		Geheugen[-R9] ← BT; BT ← Geheugen[Adres]
KTG			BT ← Geheugen[R9+]
HST	Ri	O	Ri ← Geheugen[R9+]
BST	Ri	O	Geheugen[-R9] ← Ri
LEZ		O	R0 ⇐ *invoer-orgaan*
DRU		O	*uitvoer-orgaan* ⇐ R0
NWL		O	*uitvoer-orgaan* ⇐ *begin-nieuwe-lijn*
STP		P	PTW[1] ← 0
Voor de omzetting van de symbolische naam voor het Masker, zie tabel A-7			
MKL	Masker	P	PTW[10+Masker] ← 0
MKH	Masker	P	PTW[10+Masker] ← 1
TSM	Masker	P	CC ← PTW[10+Masker]
TSO	Masker	P	CC ← Onderbrekingsvlag[Masker]
	Merk op: G niet toegestaan; GPF test eerste vlag		
OND	Waarde		Onderbrekingsvlag[1] ← 1 (Waarde wordt ***niet*** gebruikt!)
KTO		P	PTW[0..9] ← Geheugen[R9+]
INV	Ri,Poort	P	Ri ← Poort
UTV	Ri,Poort	P	Poort ← Ri

TABEL A-11. De ASCII-code tabel

Sym	Code	Sym	Code	Sym	Code	Sym	Code
NUL	000	SOH	001	STX	002	ETX	003
EOT	003	ENQ	005	ACK	006	BEL	007
BS	008	HT	009	NL	010	VT	011
NP	012	CR	013	SO	014	SI	015
DLE	016	DC1	017	DC2	018	DC3	019
DC4	020	NAK	021	SYN	022	ETB	023
CAN	024	EM	025	SUB	026	ESC	027
FS	028	GS	029	RS	030	US	031
SP	032	!	033	"	034	#	035
$	036	%	037	&	038	'	039
(	040	)	041	*	042	+	043
,	044	-	045	.	046	/	047
0	048	1	049	2	050	3	051
4	052	5	053	6	054	7	055
8	056	9	057	:	058	;	059
<	060	=	061	>	062	?	063
@	064	A	065	B	066	C	067
D	068	E	069	F	070	G	071
H	072	I	073	J	074	K	075
L	076	M	077	N	078	O	079
P	080	Q	081	R	082	S	083
T	084	U	085	V	086	W	087
X	088	Y	089	Z	090	[	091
\	092	]	093	^	094	_	095
`	096	a	097	b	098	c	099
d	100	e	101	f	102	g	103
h	104	i	105	j	106	k	107
l	108	m	109	n	110	o	111
p	112	q	113	r	114	s	115
t	116	u	117	v	118	w	119
x	120	y	121	z	122	{	123
|	124	}	125	~	126	DEL	127

Appendix B *Programma's in pseudo-Pascal of C.*

In deze appendix wordt een hoog-niveau beschrijving gegeven (in pseudo-PASCAL) voor de **vertaler** (*B.1*) en de **lader** (*B.2 (op pag. 209)*). Ten slotte geven we het C-programma voor de **DRAMA-simulator** (*B.3 (op pag. 211)*).

B.1 De DRAMA-vertaler

De DRAMA-vertaler werkt in twee stappen. In de eerste stap wordt de symbooltabel opgesteld; in de tweede stap wordt de code/gegevens gegenereerd.

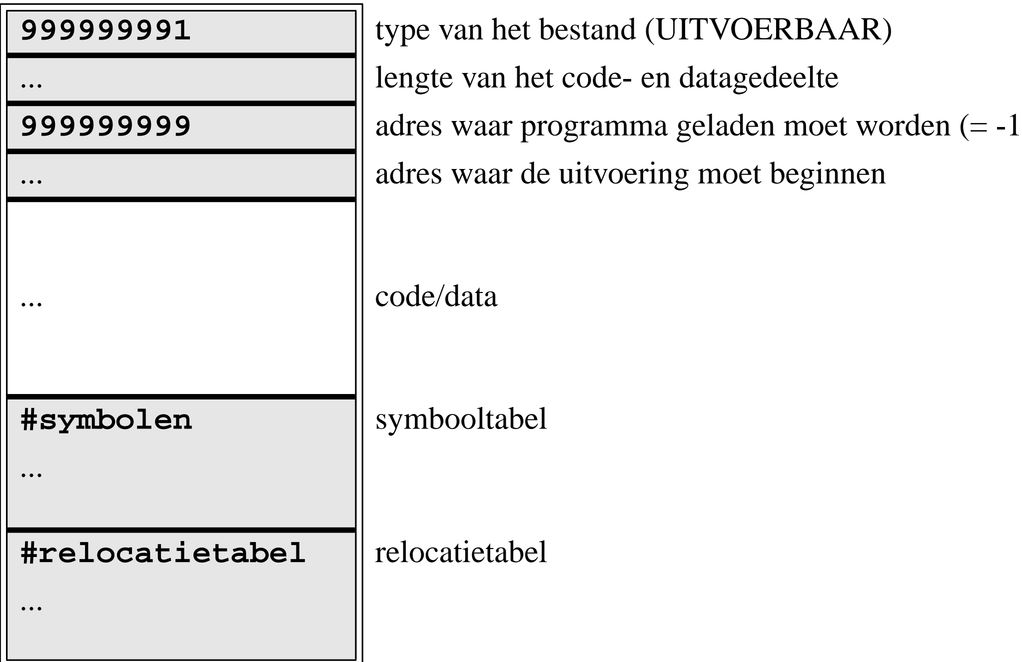
```
program vertaler (input, output);

type
   lijntype: (commentaar, dramainstr, eindpr, constante, resgr, fout);
   pa: packed array[...] of char;
   symbool_entry:  record
                      naam: packed array[1..15] of char;
                      waarde: integer;
                   end;

var
   lijnnummer, programmateller, increment: integer;
   verder: boolean;
   symbooltabel: array[...] of symbool_entry;
   lijn: pa;

function etiket_aanwezig(...) : boolean; extern;
function aantal_reservaties(...) : integer; extern;
function aantal_constanten(...) : integer; extern;
procedure voeg_toe(...); extern;
function type_lijn(...) : lijntype; extern;
procedure genereer_code(...); extern;
procedure genereer_constanten(...); extern;
```

```
procedure genereer_reservaties(...); extern;

begin
        { Stap 1: opstellen van de symbooltabel }

  lijnnummer := 1; programmateller := 0; verder := true;
  while not eof and verder do
  begin
        readln(lijn);
        if etiket_aanwezig(lijn) then
          voeg_toe (etiket(lijn), programmateller, symbooltabel);
        increment := 0;
        case type_lijn(lijn) of
          commentaar: ;
          dramainstr: increment := 1;
          eindpr:     verder := false;
          constante:  increment := aantal_constanten(lijn);
          resgr:      increment := aantal_reservaties(lijn);
          fout:       ... druk foutboodschap ...
        end
        lijnnummer := lijnnummer + 1;
        programmateller := programmateller + increment;
  end;

        { Stap 2: Codegeneratie }

  reset(input);
  lijnnummer := 1; verder := true;
  while not eof and verder do
  begin
        readln(lijn);
        case type_lijn(lijn) of
          commentaar: ;
          dramainstr: genereer_code(lijn, symbooltabel);
          eindpr:     verder := false;
          constante:  genereer_constanten(lijn);
          resgr:      genereer_reservaties(lijn);
          fout:       ... druk foutboodschap ...
        end
        lijnnummer := lijnnummer + 1;
  end
end.
```

FIGUUR B-1. Pseudo pascal-code voor de vertaler

B.2 De reloceerbare DRAMA-lader

We veronderstellen dat een uitvoerbaar programma er als volgt uitziet:



We gaan er ook van uit dat de lader elk programma steeds begint te laden vanaf adres START. Dit adres is als een constante gedefinieerd. In werkelijkheid zal deze waarde als een parameter doorgegeven worden aan de lader.

```
program lader (input, output);

const   START = laadadres;
        UITVOERBAAR = 9999999991;

var
   typ : integer;  {type van de huidige module}
   lengte : integer;  {lengte van de huidige module}
   adres : 0 .. 9999; {relatief adres in relocatietabel}
   bewerking : char;
   laadadres: integer; {het adres waar het programma geladen moet worden}
   startuitvoering : 0 .. 9999;  {begin uitvoering van het progr.}
   reloceerbaar: boolean; {het programma is al dan niet reloceerbaar}
   rest, lijn : packed array ... of char;
   {Geheugen : dit is geen variabele maar het geheugen zelf}

begin
   reloceerbaar := true;
   readln(typ);
```

```
    if typ ≠ UITVOERBAAR then Fout(...);
    readln(lengte);

    readln(laadadres);
    { indien laadadres ≠ -1 dan is het programma niet reloceerbaar! }
    if laadadres = -1 then laadadres = START;
    else reloceerbaar := false;

    readln(startuitvoering);

                { Stap 1: laden van de machinebevelen}

    for adres := laadadres to laadadres+lengte-1 do
        readln(Geheugen[adres]);

    if reloceerbaar then begin

                    { Sla symbooltabel over }
        repeat
            readln(lijn);
        until lijn = '#relocatietabel';

                    { Stap 2: relocatie}

        while not eof do
        begin
            readln(adres, bewerking, rest);
            if bewerking = '+' then
                Geheugen[laadadres+adres] = Geheugen[laadadres+adres] + laadadres;
            else
                Geheugen[laadadres+adres] = Geheugen[laadadres+adres] - laadadres;
        end

        startuitvoering := startuitvoering + laadadres; { reloceer startuitvoering }
    end;

                {Stap 3: start uitvoering}

    goto startuitvoering; {dit is niet in PASCAL te schrijven}
end.
```

FIGUUR B-2. Pseudo-Pascal-code voor de relocerende lader

B.3 De DRAMA-simulator

Dit programma simuleert de bevelencyclus van de DRAMA-machine in de ***while(true)***-lus. Het stopt nooit en kan alleen beëindigd worden met CTRL-C. Indien de simulator geen programmaonderbrekingen moet kunnen ondersteunen, dan kan de code heel wat vereenvoudigd worden.

```
long BR;     /* bevelenregister */
long OR;   /* operandregister */
int PTW[20] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
int po_vlag [10] = {-1, 0, 0, 0, 0, 0, 0, 0, 0 };
int register [10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};       /* de rekenregisters van de processor */
int poort[100];             /* de poorten van de randapparaten */
long geheugen[10000];     /* het DRAMA-geheugen */

/* De code voor volgende routines wordt niet gegeven */
void setBT(int) { ... }           /* vul integer BT in PTW in */
int getBT() { ...}                /* haal bevelenteller op uit PTW en vorm om naar integer */
void setCC(int) { ... }           /* geef conditiecode nieuwe waarde */
long expandeer (int) { ... }    /* expandeer 4-cijfergetal naar 10 cijfers */

int main()
{
        int modus, modus1, modus2, acc, idx; /* instructievelden */
        int br;   /* bevelenteller, uit PTW gehaald */

        /* Hier de code voor de absolute/relocerende lader inlassen */

        while (true)
        {
                if (! PTW[1])   /* In uitvoeringstoestand */
                {
                        /* -----------------------------------------------------
                         * Stap 1: Ophalen instructie + ophogen BT
                         *-----------------------------------------------------*/
                        bt = getBT();
                        BR = geheugen[bt++];
                        setBT (bt % 10000);
                        /* -----------------------------------------------------
                         * Stap 2: Analyse van de instructie
                         *------------------------------------------------------ */
                        fc = (int) (BR / 100000000L);
                        modus = (int) (BR / 1000000L) % 100;
                        modus1 = (int) (modus / 10);
                        modus2 = modus % 10;
                        acc = (int) (BR / 100000L) % 10;
                        idx = (int) (BR / 10000L) % 10;
                        OR = expandeer(BR % 10000);
```

```
/* pre-increment/decrement */
if (modus2 == 3) register[idx]++;
if (modus2 == 4) register[idx]--;

/* indexatie? */
if (modus2 >= 2 && modus2 <= 6) OR += register[idx];

/* post-increment/decrement */
if (modus2 == 5) register[idx]++;
if (modus2 == 6) register[idx]--;

switch (modus1)
{
        case 2: /* adres */
                OR % 10000;
                break;
        case 3: /* direct adres */
                OR = geheugen[OR % 10000];
                break;
        case 4: /* indirect adres */
                OR = geheugen[geheugen[OR % 10000] % 10000];
                break;
}

/* ---------------------------------------------------
 * Stap 3: Uitvoeren van de instructie
 * --------------------------------------------------- */
switch (fc)
{
        case 11:  /* HIA */
                register[acc] = OR;
                setCC(OR);
                break;
        case 12:  /* BIG */
                geheugen[OR % 10000] = register[acc];
                setCC(register[acc]);
                break;
        case 21:  /* OPT */
                /* test op overloop en zet PTW[4] ... */
                register[acc] += OR;
                setCC(register[acc]);
                break;
        case 22:  /* AFT */ ...
        case 23:  /* VER */ ...
        case 24:  /* DEL */ ...
        case 25:  /* MOD */ ...
        case 31:  /* VGL */ ...
```

```
case 32: /* SPR */
        setBT (OR % 10000);
        break
case 33: /* VSP */
        switch (acc) /* voorwaarde */
        {
                case 0: /* SO */
                        if (PTW[5] == 1) setBT(OR % 10000);
                        break;
                case 1: /* NUL */
                        if (cc == 0) setBT(OR % 10000);
                        break;
                case 2: /* NNEG */ ...
                case 3: /* NPOS */ ...
                case 4: /* OVL */
                        if (PTW[4] == 1) setBT(OR % 10000);
                        break;
                case 5: /* GOVL */ ...
                case 6: /* POS */ ...
                case 7: /* NEG */ ...
                case 8: /* NNUL */ ...
                case 9: /* GSO */ ...
        }
        break;
case 41: /* SBR */
        geheugen[register[9]--] = bt;
        setBT (OR % 10000);
        break;
case 42: /* KTG */
        setBT(geheugen[register[9]++] % 10000);
        break;
case 51: /* MKL */
        if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
        else setCC (0);
        PTW[10+acc] = 0;
        break;
case 52: /* MKH */ ...
case 53: /* TSM */
        if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
        else setCC (PTW[10+acc]);
        break;
case 54: /* TSO */
        if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
        else setCC (po_vlag[acc]);
        break;
case 61: /* OND */
        po_vlag[1] = 1;
        break;
```

```
            case 62:  /* KTO */
                if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
                else setPTW (geheugen[register[9]++]);
                break;

            /* weg te laten instructies indien INV en UTV beschikbaar zijn */
            case 71:  /* LEZ */
                register[0] = getint();
                setCC (register[0]);
                break;
            case 72:  /* DRU */ ...
            case 73:  /* NWL */ ...
            /* einde weg te laten instructies */

            case 81:  /* INV */
                if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
                else {
                    register[acc] = poort[OR % 10000];
                    setCC(register[acc]);
                }
                break;
            case 82:  /* UTV */
                if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
                else {
                    poort[OR % 10000] = register[acc];
                    setCC(register[acc]);
                }
                break;
            case 99:  /* STP */
                if (PTW[2] == 1) po_vlag[9] = 1; /* probleemtoestand */
                else PTW[1] = 0;
                break;
            default:  /* Ongeldig */
                po_vlag[9] = 1;
                break;
        }
    }

    /* test op stapelonderloop */
    if (register[9] > 9000) && PTW[11] ==0 && PTW[18] == 0) {
        PTW[5] = 1;
        po_vlag[8] = 1;
    }
```

```
        /* ------------------------------------------------------------------
         * Stap 4: behandel aanvragen tot programmaonderbreking
         * ------------------------------------------------------------------*/
        if ((PTW[10] == 0) &&      /* Globaal Masker = LAAG*/
            (PTW[0] < 9))     /* ONV < 9 */
        {
            int prio = 0; /* prioriteit = nummer van aangevraagde po */
            int i;

            for (i = 9; i > PTW[0]; i--)
            {
                if (po_vlag[i] == 1  && PTW[10+i] == 0) {
                    if ((i != 8 && i != 7) || PTW[11] == 0) { /* GPF-masker ? */
                        prio = i;
                        break;
                    }
                }
            }
            if (prio > 0) {
                geheugen[register[9]--] = getPTW();
                PTW[0] = prio;              /* ONV */
                PTW[1] = 1;                 /* uitvoeringstoestand */
                PTW[2] = 0;                 /* supervisortoestand */
                po_vlag[prio] = 0;
                setBT (geheugen[9990 + prio]);
            }
        }
    } /* while (true) */
}
```

Appendix C
Macro's in een willekeurige context

Wanneer macro's door een voorvertaler verwerkt (geëxpandeerd) worden, kan men macro's in elke context gebruiken. Het volgende voorbeeld heeft niets met een DRAMA-programma te maken. Het is een Engels kinderliedje. Aangezien de verzen heel wat gemeenschappelijk hebben (op de diersoorten en de geluiden na), doen we er profijt aan door een macro '**`VERSE`**' te definiëren die dan voor de verschillende dieren opgeroepen wordt. Indien de macroverwerking door de DRAMA-vertaler gebeurde — i.p.v. door een voorvertaler — zou de vertaler bij de eerste expansie reeds een foutenboodschap genereren:

"`*** fout *** Old: onbestaande functiecode`".

Een voorvertaler heeft er echter geen problemen mee ….

```
     MACRO
     VERSE    ANIMALS,SOUND
Old MacDonald had a farm, E-I-E-I-O,
And on this farm he had some <ANIMALS>, E-I-E-I-O!
With a <SOUND> <SOUND> here and a <SOUND> <SOUND> there,
here a <SOUND> there a <SOUND> everywhere a <SOUND> <SOUND>,
Old MacDonald had a farm, E-I-E-I-O.
     MCREINDE

VERSE chicks,cheep

VERSE ducks,quack

VERSE turkeys,gobble
```

De uitvoer van de voorvertaler is dan:

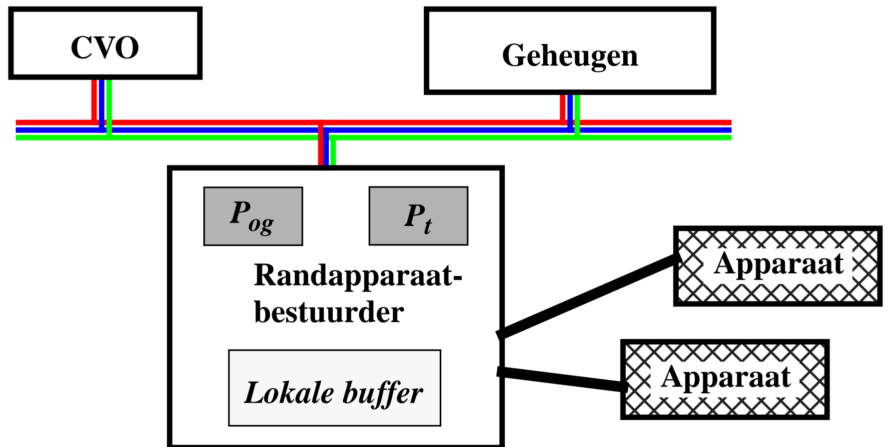
```
Old MacDonald had a farm, E-I-E-I-O,
And on this farm he had some chicks, E-I-E-I-O!
With a cheep cheep here and a cheep cheep there,
here a cheep there a cheep everywhere a cheep cheep,
Old MacDonald had a farm, E-I-E-I-O.

Old MacDonald had a farm, E-I-E-I-O,
And on this farm he had some ducks, E-I-E-I-O!
With a quack quack here and a quack quack there,
here a quack there a quack everywhere a quack quack,
Old MacDonald had a farm, E-I-E-I-O.

Old MacDonald had a farm, E-I-E-I-O,
And on this farm he had some turkeys, E-I-E-I-O!
With a gobble gobble here and a gobble gobble there,
here a gobble there a gobble everywhere a gobble gobble,
Old MacDonald had a farm, E-I-E-I-O.
```

Appendix D
De randapparaten van de DRAMA-machine

De DRAMA-machine kent vier soorten randapparaten:

- het scherm,
- het toetsenbord,
- de schijf,
- de drukker.

Bovendien is er ook een ‘klok’ en een ‘wekker’ aanwezig. De algemene structuur wordt weergegeven in figuur D-1.



FIGUUR D-1. Een typische bestuurder van een randapparaat.

De bestuurder van een randapparaat heeft **twee poorten**:

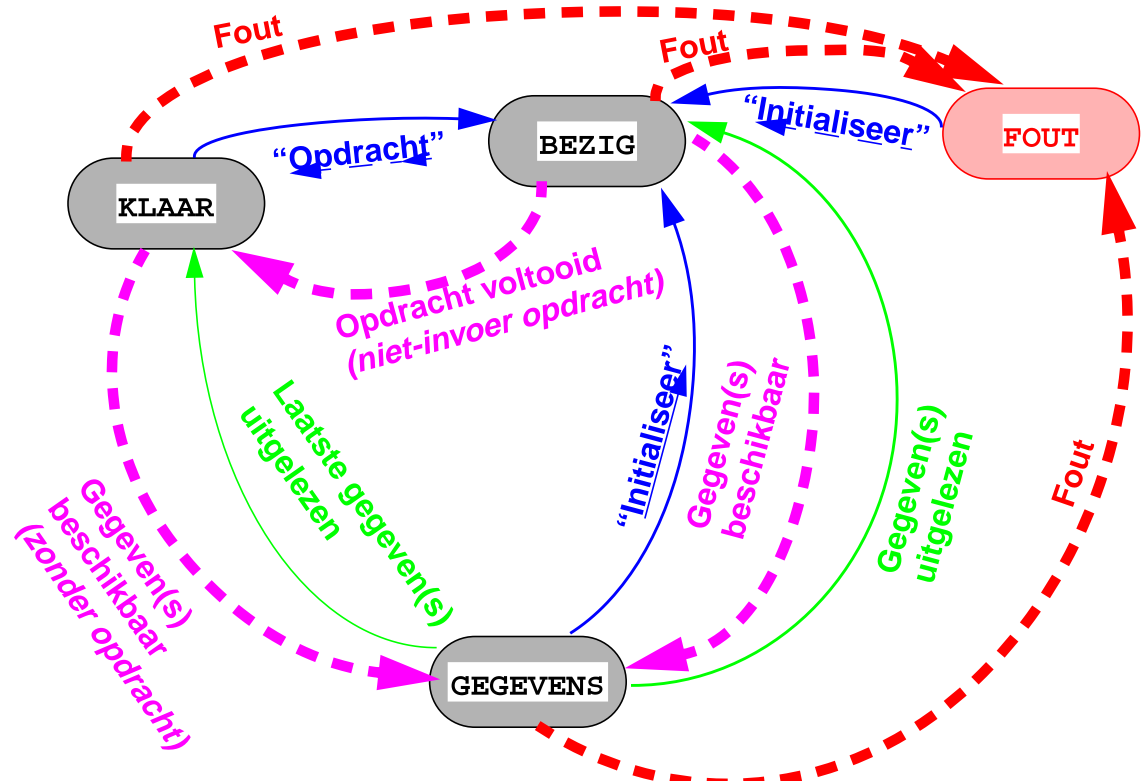
- een ***toestandspoort***, P_t, (waaruit alleen waarden kunnen gelezen worden),
- een ***gegevens- en/of opdrachtpoort***, P_{og}, (waaruit gegevens kunnen gelezen worden en waarin opdrachten kunnen geschreven worden)

Een bestuurder kan zich in **vier verschillende toestanden** bevinden:

Waarde	Naam	Verklaring
0000000000	**KLAAR**	De bestuurder is klaar voor een nieuwe opdracht.
0000000001	**BEZIG**	Het toestel is bezig met de uitvoering van een opdracht.
0000000002	**GEGEVENS**	Er zijn gegevens beschikbaar in de gegevenspoort.
9999999999	**FOUT**	Er heeft zich een fout voorgedaan.

Sommige bestuurders zullen geen **GEGEVENS**-toestand hebben, omdat ze geen gegevens ter beschikking stellen: zij besturen louter uitvoerapparaten zoals bijvoorbeeld drukkers.

Figuur D-2 geeft de mogelijke overgangen tussen de toestanden weer. De dikke pijlen geven aan wanneer de bestuurder een programmaonderbreking zal aanvragen: wanneer een opdracht uitgevoerd is, als er gegevens beschikbaar zijn, en wanneer zich een fout heeft voorgedaan.



FIGUUR D-2. Het toestandsdiagram voor een bestuurder.

In twee toestanden bevat de gegevenspoort informatie die kan uitgelezen worden:

Toestand	Verklaring
FOUT	De oorzaak van de fout staat in de gegevenspoort.
GEGEVENS	De gegevens zijn beschikbaar in de gegevenspoort.

In de volgende paragrafen wordt een gedetailleerde beschrijving gegeven van elk van de randapparaten en van de klok en wekker.

D.1 Het scherm

Kenmerken:

Toestandspoort:	P_2
Opdrachtpoort:	P_3
Snelheid:	± 1000 tekens per seconde

Het scherm kan zich in de volgende toestanden bevinden:

Waarde	Naam	Verklaring
0000000000	**KLAAR**	De bestuurder is klaar voor een opdracht.
0000000001	**BEZIG**	Het scherm is bezig met de uitvoering van een opdracht.
0000000002	**GEGEVENS**	Er is informatie beschikbaar in de gegevenspoort.
9999999999	**FOUT**	Er heeft zich een fout voorgedaan.

Merk op dat het scherm ook gegevens ter beschikking kan stellen aan de processor. Het is namelijk mogelijk om het scherm te ondervragen over de positie op het scherm waar het volgende symbool zal afgebeeld worden.

De volgende eenvoudige bevelen zijn voorzien (met 'cursor' wordt de huidige positie op het scherm bedoeld):

Opdracht	Verklaring
00001100kk	Initialiseer het scherm. **ll** = aantal lijnen, **kk** = aantal kolommen
1000000ccc	Beeld het teken ccc (ASCII-code) af op het scherm.
2000000000	Wis het volledige scherm.
3000000000	Ga naar volgende lijn (zelfde kolom).
4000000000	Ga naar het begin van de lijn (zelfde lijn).
60001100kk	Plaats de cursor op lijn **ll** en kolom **kk**.
7000000000	Geef het lijn- en kolomnummer van de huidige cursorpositie. Het resultaat in de gegevenspoort heeft het volgende formaat: **00001100kk**, waarbij **ll** het lijnnummer is en **kk** het kolomnummer is.

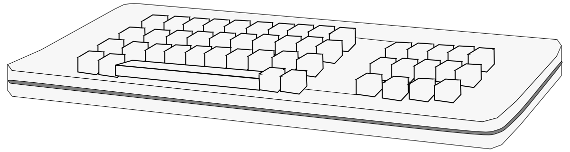
Het scherm is niet erg geavanceerd: het wissen van een teken zal moeten gebeuren door het overschrijven met een blanco. Bepaalde tekens (zoals **NL**, **VT**, …)[1] kunnen niet afgebeeld worden en zullen als een blanco weergegeven worden. Op het einde van een lijn wordt automatisch verder gegaan naar het begin van de volgende lijn (er gaan geen tekens verloren).

Indien er geen volgende lijn bestaat, schuiven alle lijnen één positie hoger (de eerste lijn gaat verloren); in het Engels noemt men dit '*scrolling*'.

Elke opdracht die aanvaard wordt zal de toestand van de bestuurder veranderen naar de **BEZIG**-toestand anders naar de **FOUT**-toestand.

1. **NL** is het 'nieuwe-lijn' teken; **VT** het 'verticale tabulatie'-teken. Het besturingsprogramma voor het scherm (d.i. de device driver) zal normaal gezien niet vragen aan de schermbestuurder om het **NL**-teken af te beelden op het scherm; daarentegen zullen de volgende twee opdrachten gegeven worden: (a) ga naar de volgende lijn en (b) ga naar het begin van de lijn. Op de DRAMA-machine is er echter geen besturingsprogramma aanwezig!

D.2 Het toetsenbord



Kenmerken:

Toestandspoort:	P_0
Gegevens/opdrachtpoort:	P_1

Het toetsenbord kan zich in de volgende toestanden bevinden:

Waarde	Naam	Verklaring
0000000000	**KLAAR**	De bestuurder is klaar voor een opdracht. Er zijn geen tekens beschikbaar.
0000000001	**BEZIG**	Het toestel is bezig met de uitvoering van een opdracht.
0000000002	**GEGEVENS**	Er is een teken beschikbaar in de gegevenspoort.
9999999999	**FOUT**	Er heeft zich een fout voorgedaan. Overloop is een mogelijke fout.

De bestuurder kent slechts één opdracht: de initialisatieopdracht:

Opdracht	Verklaring
0000000000	Initialiseer het toetsenbord.

Bij het geven van de initialisatieopdracht zal de toestand van de bestuurder wijzigen naar de **BEZIG**-toestand en uiteindelijk naar **KLAAR** of **FOUT** (indien het apparaat defect is).

Telkens een toets wordt ingedrukt zal afhankelijk van de toestand waarin de bestuurder zich bevindt het volgende gebeuren:

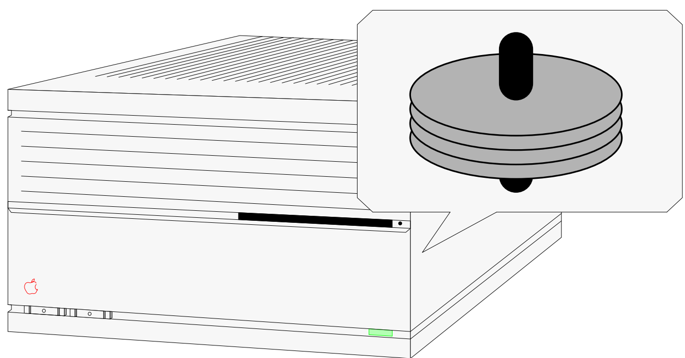
Toestand	Nieuwe toestand	Actie
KLAAR	**GEGEVENS**	De ASCII-code van de toetsaanslag is beschikbaar in P_1.
BEZIG	**BEZIG**	De toetsaanslag wordt genegeerd. De bestuurder is alleen ‘bezig’ bij het uitvoeren van de initialisatie-opdracht.

Toestand	Nieuwe toestand	Actie
GEGEVENS	**FOUT**	P_1 krijgt de waarde '**OVERLOOP**'. De toetsaanslag wordt genegeerd. **OVERLOOP = 9999999999.**
FOUT	**FOUT**	De toetsaanslag wordt genegeerd.

Bij het uitlezen van poort P_1 verandert de toestand als volgt: (— duidt op geen wijziging):

Toestand	Nieuwe toestand	Actie
KLAAR	**—**	
BEZIG	**—**	
GEGEVENS	**KLAAR**	P_1 wordt gewist (wordt op 0000000000 gezet).
FOUT	**—**	

D.3 De schijf



De DRAMA-machine kan met verschillende schijven werken. Standaard wordt slechts een kleine schijf voorzien. De grootte van een sector wordt bij het formatteren van de schijf vastgelegd.

Kenmerken:

Toestandspoort:	P_6
Opdrachtpoort:	P_7
Aantal cilinders:	300
Aantal sporen/cilinder:	40
Lengte spoor:	5000 10-cijfergetallen
Sectorlengte:	$(100 * n)$ 10-cijfergetallen $(1 \leq n \leq 50)$
Totale Capaciteit:	60.000.000 10-cijfergetallen
Rotatiesnelheid:	3000 toeren/min
Zoektijd:	min. 5 msec max. 65 msec gem. 20 msec
Rotationele wachttijd:	max. 20 msec (volledige toer) gem. 10 msec (halve toer)
Debiet:	25.000 10-cijfergetallen per seconde (zolang de kam niet verplaatst wordt)

De schijfbestuurder heeft directe geheugentoegang (DGT).

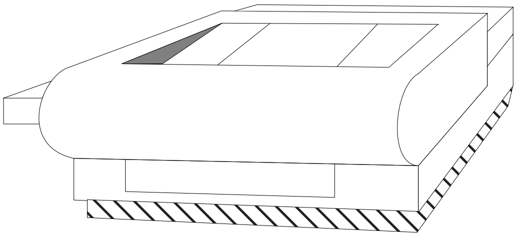
Aangezien alle gegevens rechtstreeks in het geheugen geplaatst worden, komt de **GEGEVENS**-toestand niet voor. De bestuurder kan zich in de volgende toestanden bevinden:

Waarde	Naam	Verklaring
0000000000	**KLAAR**	De bestuurder is klaar voor een opdracht.
0000000001	**BEZIG**	Het schijf is bezig met de uitvoering van een opdracht.
9999999999	**FOUT**	Er heeft zich een fout voorgedaan.

De volgende opdrachten kunnen in de opdracht-poort geschreven worden:

Opdracht	Verklaring
000lll0sss	Initialisatie. **sss** is het aantal sectoren per spoor (1 ≤ **sss** ≤ 50), **lll** is de grootte van een sector (aantal 10-cijfergetallen), **lll** moet een veelvoud zijn van 10 en **sss * lll** ≤ 500, (indien **sss * lll** < 500, wordt een deel van elk spoor niet gebruikt).
100ttt0ccc	Positioneer de lees/schrijfkoppen op cilinder **ccc** en activeer kop **ttt**.
200lll0sss	Lees **lll** opeenvolgende sectoren in te beginnen bij sector **sss**.
300lll0sss	Schrijf **lll** opeenvolgende sectoren te beginnen bij sector **sss**.
500000gggg	DGT-geheugenzone begint op adres **gggg**.
9000000000	Formatteer de schijf, volgens de ingestelde sector-grootte (zie initialisatie).

D.4 De drukker



Kenmerken:

Toestandspoort:	P_4
Opdrachtpoort:	P_5
Snelheid:	± 200 tekens per seconde

De bestuurder kan zich in de volgende toestanden bevinden (er is geen **GEGEVENS**-toestand):

Waarde	Naam	Verklaring
0000000000	**KLAAR**	De bestuurder is klaar voor een opdracht.
0000000001	**BEZIG**	De drukker is bezig met de uitvoering van een opdracht.
9999999999	**FOUT**	Er heeft zich een fout voorgedaan.

De volgende bevelen kunnen gegeven worden aan de bestuurder van de drukker:

Opdracht	Verklaring
00001100bb	Initialiseer drukker. **ll** = lengte papier (in lijnen), **bb** = breedte van het papier (in kolommen).
3000000ccc	Druk teken **ccc** (ascii-code) op de huidige positie van de drukkop, en verplaats de kop 1 positie naar rechts.
4000000ppp	Verplaats de drukkop **ppp** posities naar rechts/links. **ppp** wordt in 10-complement gegeven (negatief is naar links).
5000000ppp	Verplaats de drukkop **ppp** regels naar onder/boven. **ppp** wordt in 10-complement gegeven (negatief is naar boven).
6000000000	Breng de drukkop naar het begin van deze regel.
7000000000	Breng de drukkop naar de volgende lijn (zelfde kolom).
9000000000	Breng de drukkop naar het begin van de volgende pagina.

D.5 De klok

Kenmerken:

Toestandspoort:	—
Gegevens/opdrachtpoort:	P_8

De klok heeft geen toestandspoort. De huidige tijd (d.i. het aantal verstreken tienden van een seconde sinds 1 januari 1990, middernacht) kan uitgelezen worden uit P_8. De uitgelezen waarde is steeds positief op te vatten. Per dag neemt de waarde van P_8 dus toe met 864.000. Dus na +/-11.574 dagen (d.i. 31 jaar, 8 maanden en enkele dagen) zal zich in P_8 een overloop voordoen. De klok verhoogt de inhoud van de P_8 elk tiende van een seconde. Ook als de computer afgezet is, blijft de klok doorlopen. Een speciale batterij zorgt voor de nodige spanning. Het is mogelijk dat de klok na verloop van een zekere tijd een verschil begint te vertonen met de werkelijke tijd. (Wanneer de computer heel lang afstaat, zal de spanning in de batterij te laag worden om de klok nog correct te laten lopen.) Door een nieuwe waarde in de P_8 te schrijven kan dit hersteld worden.

D.6 De wekker

Kenmerken:

Toestandspoort:	—
Gegevens/opdrachtpoort:	P_9

De wekker heeft geen toestandspoort. Ze wordt geactiveerd door een positief getal in P_9 te schrijven. Een negatief getal zet de wekker op non-actief. Indien de wekker actief is, zal elke micro-seconde de waarde in de opdrachtpoort met één verminderd worden. Op het ogenblik dat de waarde nul wordt zal de wekker de PO_2-vlag plaatsen. Het bereik van de wekker ligt dus tussen 1 µsec en ongeveer 1u23min.

Appendix E
Woordenlijst

In deze appendix geven we de vertaling van de in de cursustekst gebruikte Nederlandse termen. Het paginanummer verwijst naar de bladzijde waar de term voor het eerst vermeld is. De Engelse benaming is in een ander *lettertype* weergegeven.

A

Term	Vertaling	Pagina
aanbieden	*submit*	182
actief wachten	*busy waiting*	132
afregelen	*tuning*	194
assembleerprogramma	assembler	2
assembler	assembleerprogramma (vertaler)	2

B

Term	Vertaling	Pagina
batch system	systeem met stapelverwerking	181
bestand	*file*	191
besturingsprogramma	*operating system*	96
bestuurder (v/e randapparaat)	*controller*	111
binder	*linkage editor*	64
bottleneck	flessehals	193
BP	OS	97
break point	breekpunt	87
breekpunt	*break point*	87
bug	logische fout	86
busy waiting	actief wachten	132

C

Term	Vertaling	Pagina
central processing unit	centraal verwerkingsorgaan	55
centrall verwerkingsorgaan	*central processing unit*	55
channel	kanaalbestuurder	111
compilator	*compiler*	18
compiler	compilator	18
context switch	programma-wisseling	176
controller	bestuurder (v/e randapparaat)	111
CPU	CVE, CVO	55
cross-compiler	kruis-compilator	20
CVE, CVO	CPU	55
cycle stealing	cyclusdiefstal	149

cyclusdiefstal . . . . . . *cycle stealing* 149

D

debugger . . . . . . speurprogramma 16
dedicated system . . . . . . systeem voor specifiek doel 181
delayed write . . . . . . uitgesteld wegschrijven 171
device driver . . . . . . randapparaat-besturingsroutine 156
device . . . . . . randapparaat 125
direct memory access . . . . . . directe geheugentoegang 141
directe geheugentoegang . . . . . . *direct memory access* 141
directory . . . . . . inhoudstafel 192
disabled interrupt . . . . . . verboden PO 114
disassemble . . . . . . omzetting van mach. taal naar lagere prog. taal . . 87
doorvoer . . . . . . *throughput* 178
dynamic address translation . . . . . . dynamische adresvertaling 55
dynamische adresvertaling . . . . . . *dynamic address translation* 55

E

enabled interrupt . . . . . . toegelaten PO 114
extra bijkomende tijd (verspilling) . . . . *overhead* 196
extra tijd . . . . . . *overhead* 137

F

file . . . . . . bestand 191
flessehals . . . . . . *bottleneck* 193

G

GBE . . . . . . MMU 55
genegeerde PO . . . . . . *ignored interrupt* 114
general purpose system . . . . . . systeem van algemeen nut 181

I

i/u door inpassing in het geheugen . . . . *memory mapped I/O* 126
idle . . . . . . werkloos 177
ignored interrupt . . . . . . genegeerde PO 114
inhoudstafel . . . . . . *directory* 192
interpreter . . . . . . vertolker 79
interrupt handler . . . . . . PO-routine 118
interrupt service routine . . . . . . programma-onderbreking(behandeling)sroutine 118
interrupt . . . . . . programma onderbreking 111

K

kanaalbestuurder . . . . . . *channel* 111

kruiscompilator	*cross-compiler*	20

L

linkage editor	binder	64
logische fout	*bug*	86

M

mask	masker	114
masker	*mask*	114
memory management unit	geheugenbeheer-eenheid	55
memory mapped I/O	i/u door inpassing in het geheugen	126
MMU	GBE	55
monoprogrammatie	*single task*	177
multi-user system	systeem voor meerdere gebruikers	182

O

omzetting van mach. taal naar lagere prog. taal	*disassemble*	87
operating system	besturingsprogramma	96
OS	BP	97
overhead	extra bijkomende tijd (verspilling)	196
overhead	extra tijd	137

P

pending interrupt	hangende PO-aanvraag	114
PO-routine	*interrupt handler*	118
pre-processor	voorvertaler	23
program status word	programma-toestandswoord	98
programma onderbreking (fout/geprog.)	*trap*	111
programma onderbreking	*interrupt*	111
programma-onderbreking(behandeling)sroutine	*interrupt service routine*	118
programma-toestandswoord	*program status word*	98
programma-wisseling	*context switch*	176

Q

queue	wachtrij	182

R

randapparaat besturingsroutine	*device driver*	156
randapparaat	*device*	125
read ahead	vooraf inlezen	171
real time system	reëele-tijd systeem	181
reëele-tijd systeem	*real time system*	181

S

single task .... monoprogrammatie 177
single-user system .... systeem voor één gebruiker 182
speurprogramma .... *debugger* 16
submit .... aanbieden 182
supervisie-oproep .... *supervisor call* 166
supervisor call .... supervisie-oproep 166
systeem met stapelverwerking .... *batch system* 181
systeem van algemeen nut .... *general purpose system* 181
systeem voor één gebruiker .... *single-user system* 182
systeem voor meerdere gebruikers .... *multi-user system* 182
systeem voor specifiek doel .... *dedicated system* 181

T

throughput .... doorvoer 178
timesharing .... verdelen van de (processor)tijd 182
toegelaten PO .... *enabled interrupt* 114
trap .... programma onderbreking (fout/geprogr.) 111
tuning .... afregelen 194
type ahead .... vooraf intypen 171

U

uitgesteld wegschrijven .... *delayed write* 171

V

verboden PO .... *disabled interrupt* 114
verboden PO-aanvraag .... *pending interrupt* 114
verdelen van de (processor)tijd .... *timesharing* 182
vertolker .... *interpreter* 79
vooraf inlezen .... *read ahead* 171
vooraf intypen .... *type ahead* 171
voorvertaler .... *pre-processor* 23

W

wachtrij .... *queue* 182
werkloos .... *idle* 177

Appendix F
GEBRUIKTE AFKORTINGEN

In deze appendix worden de in deze cursustekst vaak gebruikte afkortingen verklaard. Tevens geven we context waar de afkortingen voorkomen. In de “Index” (pag. 235 e.v.) kan men het pagina-nummer terugvinden waar deze afkortingen gedefinieerd werden.

F.1 Nederlandse afkortingen

Afkorting	Betekenis	Context
AFT	AFTrekking	DRAMA-instructie
ALVT	Argumenten en Lokale Variabelen Tabel	voorvertaler
BIG	Berg In Geheugen	DRAMA-instructie
BO	BesturingsOrgaan	processor
BP	BesturingsProgramma	
BR	BevelenRegister	processor
br	BasisRegister	GBE
BT	BevelenTeller	PTW
CC	ConditieCode	PTW
CVE	Centrale VerwerkingsEenheid	processor = CVO
CVO	Centraal VerwerkingsOrgaan	processor
DEL	DELing	DRAMA-instructie
DGT	Directe GeheugenToegang	invoer/uitvoer
DRAMA	Decimale RekenAutomaat met Meerdere Accumulatoren	modelcomputer tweede generatie
DRU	DRUkken	DRAMA-instructie
G	Globaal masker	PTW
GAR	Geheugen AdresRegister	processor - geheugen
GBE	GeheugenBeheerEenheid	processor
GBR	Geheugen BufferRegister	processor - geheugen
GET	Globale EtikettenTabel	voorvertaler

Afkorting	Betekenis	Context
`GPF`	Globaal Programmafout Masker	PTW
GVT	Globale VariabelenTabel	voorvertaler
`HIA`	Haal In Accumulator	DRAMA-instructie
`INV`	INVoer	DRAMA-instructie
`KTG`	Keer TeruG	DRAMA-instructie
`KTO`	Keer Terug na Onderbreking	DRAMA-instructie
LET	Lokale EtikettenTabel	voorvertaler
`LEZ`	LEZen	DRAMA-instructie
`MKH`	MasKer Hoog	DRAMA-instructie
`MKL`	MasKer Laag	DRAMA-instructie
`MOD`	MODulus	DRAMA-instructie
`NWL`	NieuWe Lijn	DRAMA-instructie
`OND`	ONDerbreking	DRAMA-instructie
`OPT`	OPTelling	DRAMA-instructie
OR	OperandRegister	processor
`OVI`	OVerloop-Indicator	PTW
PO	ProgrammaOnderbreking	processor
PTW	ProgrammaToestandsWoord	processor
RO	RekenOrgaan	processor
`SBR`	SuBRoutine oproep	DRAMA-instructie
`SOI`	StapelOverloopIndicator	DRAMA-instructie
`SPR`	SPRong	DRAMA-instructie
`STP`	SToP	DRAMA-instructie
`TSM`	TeSt Masker	DRAMA-instructie
`TSO`	TeSt Onderbrekingsvlag	DRAMA-instructie
`UTV`	UiTVoer	DRAMA-instructie
`VER`	VERmenigvuldiging	DRAMA-instructie
`VGL`	VerGeLijking	DRAMA-instructie
`VSP`	Voorwaardelijke SProng	DRAMA-instructie
VV	VoorVertaler	macro’s

F.2 Engelse afkortingen

Bij de Engelse afkortingen hebben we het Nederlands equivalent weergegeven (indien beschikbaar).

Afkorting	Betekenis	Nederlands equivalent
CPU	Central Processing Unit	CVO
DAT	Dynamic Address Translation	
DMA	Direct Memory Access	DTG
I/O	Input/Output	I/U
MMU	Memory Mangement Unit	GBE
OS	Operating System	BP
PSW	Program Status Word	PTW
RAM	Random Access Memory	
ROM	Read Only Memory	

Index

A

absolute lader 60
accumulator, zie rekenregister
achterwaartse referentie, zie symbolisch adres
actief wachten 146
ALVT, zie argumenten en lokale variabelen tabel
APL 98
argumenten en lokale variabelen tabel, zie voorvertaler
argumententabel, zie voorvertaler
ASCII-voorstelling 29, 220
assembleerprogramma, zie vertaler

B

back-end computer 168
backup-computer 169
basisregister 69
bestand 80, 205
bestandenbeheer 205
besturingspaneel 106
  LAADADRES-knop 106
  LAADINSTRUCTIE-knop 106
  VOERUIT-knop 106
besturingsprogramma
  bestandenbeheer 205
  boekhouding 207
  DOS 110
  één-gebruiker systeem 196
  geheugenbeheer 198
  informatiebeheer 206
  meerdere-gebruikers systeem 196
  MVS 110
  OS/2 110
  processorbeheer 200
  programma-toestandstabel 201
  randapparaat-toestandstabel 205
  reële-tijd systeem 195
  soorten 195
  stapelverwerking 195
  timesharing 196, 202
  UNIX 110
  VMS 110
besturingssysteem, zie besturingsprogramma
bestuurder 139
bevelencyclus 117, 121, 131, 186
  sequentiële besturing 118
bevelenopmaak 16, 18
  accumulator veld 18
  acc-veld, zie accumulator veld
  idx-veld, zie indexregister veld
  indexregister veld 18
  klassen 18
  mode veld 17
  opcode veld, zie operatiecode veld
  operand-veld 18
  operatiecode veld 17
  velden 17
bevelenteller 112
binder 78
  dynamisch binden 88
  hoofdprogramma 81
  programmabibliotheek 79
  start-aanduiding 81
  type
    aanduiding 82
boekhouding 207
booleaanse waarde 21
bootsector 107
bootstrap procedure 107
breekpunt, zie speurprogramma
bronprogramma 16
BT 112
bug, zie logische fout
bytecode 95

C

CC 112
centraal verwerkingsorgaan 69
central processing unit, zie centraal verwerkingsorgaan
centrale verwerkingseenheid 69
commentaar 20, 24
compilator 32
  kruiscompilator 34
  lexicale analyse 33
  semantische analyse 33
  syntactische analyse 33
  syntaxisboom 33
compiler
  compiler-compiler 34
compiler, zie compilator
conditiecode 112
constante 20, 24
conversieproblemen 98
cross-compiler, zie kruiscompilator
cve 69
cyclusdiefstal 163

D

datum 206
debugger, zie speurprogramma
debugging 100
definitie-mode, zie voorvertaler
device drivers, zie randapparaat-besturingsroutines
dienstnummer 135
direct memory access, zie directe geheugentoegang
directe geheugentoegang 155
  cyclusdiefstal 163
DMA 155
doelcomputer 95
doorvoer 191
DOS 110
DRAMA
  instructie 17
DRAMA-machine
  indexatie
    auto increment/decrement 17
  instructie
    DRU 139
    INV 140
    KTO 133
    LEZ 139
    MKH 129
    MKL 129
    OND 180
    STP 193
    TSM 129
    TSO 129
    UTV 140
  processortoestanden
    geprivilegeerde bevelen 178
    halttoestand 173
    overgangen 177
    probleemtoestand 176
    supervisietoestand 176
    uitvoeringstoestand 173
  programma-onderbreking
    dienstnummer 135
    genegeerd 128
    geprogrammeerd 135
    geweigerd 128
    hangende 128
    negeren 127
    opschorten 127
    prioriteit 130
    soorten 124
    toegelaten 128
    uitgesteld 128
    uitstellen 127
    verbieden 127
  programma-onderbrekingsmechanisme
    maskers 127
    PO-vectoren 122
    prioriteit 126
    programma-onderbrekingsvlag 120
  programma-toestandswoord
    BT 112
    CC 112
    DRK 126
    G 129
    globale maskers 129
    GPF 129
    H/U 174
    IN 126
    maskerwoord 128
    MFT 126
    OND 135
    onderbrekingsniveau 130
    ONV 130
    OVI 112, 126
    OVL 126
    S/P 176
    SCH 126
    SOI 112, 126
    SPL 126
    UIT 126
    WEK 126
DRAMA-machine
  bevelencyclus 117, 121, 131, 186
  bevelenteller 112
  conditiecode 112
  indexatie 17
  indexregister 18
  indicatoren 112
  klok 206
  overloopindicator 112
  processortoestanden 173
  programma-onderbreking 115
  programma-onderbrekingsmechanisme 119
  programma-toestandswoord 112
  sprongbevel 118
  stapeloverloop-indicator 112
  supervisie-oproep 180
  wekker 196
DRAMA-taal
  directief

EINDPR 24
MACRO 37
RESGR 24
DRAMA-vertaler
directief
MCREINDE 37
instructie
interpretatie 24
DRAMA-vertaler
codegeneratie 30
commentaar 24
constante 24, 29
directief 24
EXTERN 79
geheugenreservatie 29
GLOBAAL 80
instructie 24
invoerlijn 24
LAADGR 63, 65
lijnteller 28
machine-instructie tabel 26
programmateller 28, 29
STARTPR 81
symbolische adres 25
symbooltabel 28
tweestapsvertaling 27
drama-vertaler
symbooltabel 30
DRK 126
DRU 139
DTG 155
dynamisch binden 88

E

één-gebruiker systeem 196
EINDPR 24
emulator 99
expansie-mode, zie voorvertaler
EXTERN 79

F

factuur 207
fouten 22
front-end computer 168

G

G 129
gastcomputer 95
geheugenbeheer 198
geheugenbeheer-eenheid 69
geheugen-beschermingsregisters 199
geheugenregisters 17
geprivilegeerde bevelen 178
getallenvoorstelling
10-complement 20
booleaanse waarde 21
GLOBAAL 80
globale etiketten tabel, zie voorvertaler
globale voorvertaler variabelen tabel, zie voorvertaler
GPF 129

H

H/U 174
halttoestand 106, 173
HIB 72
hoofdprogramma 81

I

IN 126
indexatie 17
indicatoren 112
informatiebeheer 206
instructie 20, 24
interpretatie 24
mnemotechnische naam 20
interpretatie, zie instructie
interpreter, zie vertolker
INV 140
invoer/uitvoer
actief wachten 146
back-end computer 168
directe geheugentoegang 155
front-end computer 168
geprogrammeerd 145
kanaalbestuurder 164
organisatievormen 145
programma-onderbrekingen 151
satelliet-computer 168

J

Java Virtual Machine 95
JVM 95

K

kanaalbestuurder 164

klok 206
knelpunt 207
kopiëer-mode, zie voorvertaler
koppelprogramma, zie binder
kruiscompilator 34
KTO 133

L

laadadres-aanduiding 63
LAADADRES-knop, zie besturingspaneel
LAADGR 63, 65
LAADINSTRUCTIE-knop, zie besturingspaneel
lader
  absolute lader 60
  algoritmes 60
  laadadres-aanduiding 63
  relocatie- en bindingstabel 82
  relocatietabel 66
  relocerende lader 60
  start-aanduiding 81
  type-aanduiding 82
lader-algoritmes, zie lader
lader-schema, zie lader-algoritme
lexicale analyse 33
LEZ 139
listingbestand 30
logische fout 100
lokale macro etiketten tabel, zie voorvertaler

M

machine-instructie tabel 20
machinetaal 16
MACRO 37, 39
macro 37
  actuele parameter 39
  expansie 37
  formele parameter 39
  hoofding 39
  lichaam 39
  lokale variabele 44
  oproep 37, 39
  symbolisch adres
    lokaal 41
macro-definitie 37
macro-oproep 37, 39
  genesteld 41
MACRO-voorvertaler, zie voorvertaler
maskerwoord 128
MCREINDE 37, 39
meerdere-gebruikers systeem 196
MEVA 43
MFOUT 43
MFT 126
MINI-DRAMA-vertaler 19
  commentaar 20
  constante 20
  fouten 22
  instructie 20
  machine-instructie tabel 20
MKH 129
MKL 129
mnemotechnische naam 20
MNTS 43
monoprogrammatie 191
MSPR 43
multiprogrammatie 189
MVS 110
MVSP 43

O

OBJECTMODULE 82
objectmodule 82
OND 135, 180
onderbrekingsniveau 130
onderhoudscontract 169
oneindige lus 116
ONV 130
OS/2 110
overloop 115
overloopindicator 112
OVI 112, 126
OVL 126

P

PASCAL-machine 99
poort 139
pre-processor, zie voorvertaler
probleemtoestand 176
processor
  werkloos 191
processor, zie centraal verwerkingsorgaan
processorbeheer 200
processortoestanden 173
  gepriviligeerde bevelen 178
  halttoestand 173
  overgangen 177
  probleemtoestand 176
  supervisietoestand 176
  uitvoeringstoestand 173
programmabibliotheek 79

programma-onderbreking 115
  asynchroon 125
  genegeerd 128
  geprogrammeerd 135
    dienstnummer 135
  geweigerd 128
  globale maskers 129
  hangende aanvraag 128
  invoer/uitvoer 151
  maskers 127
  maskerwoord 128
  negeren 127
  onderbrekingsniveau 130
  opschorten 127
  prioriteit 126, 130
  soorten 124
  synchroon 125
  toegelaten 128
  uitgesteld 128
  uitstellen 127
  verbieden 127
programma-onderbrekingsmechanisme 119
programma-onderbrekingsroutine 132
programma-onderbrekingsvectoren 122
programma-onderbrekingsvlag 120
programma-toestandstabel 201
programma-toestandswoord 112
  bevelenteller 112
  conditiecode 112
  indicatoren 112
    overloopindicator 112
    stapeloverloop 112
programmawisseling
  multiprogrammatie
    programmawisseling 190

R

randapparaat 139
  besturingsroutines 170
  bestuurder 139
    kanaalbestuurder 164
    poort 139
randapparaat-besturingsroutines 170
randapparaat-toestandstabel 205
reële-tijd systeem 195
rekenregister 18
relocatie
  basisregister 69
  statisch 69
relocatie- en bindingstabel, zie lader
relocatietabel 66
relocerende lader 60
RESGR 24
responstijd 202

S

S/P 176
satelliet-computer 168
SCH 126
scherm
  scrolling 235
schijf
  inhoudstafel 206
semantische analyse 33
sequentiële besturing 118
SGI 72
SGU 72
simulator 95, 98
  conversieproblemen 98
sleutelwoord 40
SOI 112, 126
speurprogramma 30, 100
  bevelen 101
  breekpunt 101
  logische fout 100
SPL 126
sprongbevel 118
stapeloverloop-indicator 112
stapelverwerking 195
start-aanduiding 81
STARTPR 81
STP 193
supervisie-oproep 180
supervisietoestand 176
#symbolen 65, 66
symbolisch adres 25
  achterwaartse referentie 25
  voorwaartse referentie 25, 26, 27
symboolmanipulatie 37
symbooltabel 28, 30
syntactische analyse 33
syntaxisboom 33

T

10-complementsvoorstelling 20
tijd 206
timesharing 196, 202
  responstijd 202
TSM 129
TSO 129
type-aanduiding 82

U

UIT 126
UITVOERBAAR 82
uitvoerbaar programma 16
uitvoeringstoestand 106, 173
UNIX 110
upgrade 89
UTV 140

V

versie 89
vertaler
  directief 63, 79
  hogere programmeertaal 32
    kruiscompilator 34
  statische relocatie 69
vertaler-directief 24, 63, 79
vertolker 93
VMS 110
VOERUIT-knop, zie besturingspaneel
voorvertaler
  argumenten en lokale variabelen tabel 49
  argumententabel 39, 40
  definitie-mode 39
  directief 43
  expansie-mode 41, 54
  globale etiketten tabel 52
  globale variabele 44
  globale variabelen tabel 49
  kopiëer-mode 39, 53
  lokale etiketten tabel 52
  MACRO 39
  macro
    actuele parameter 39
    definitie 37
    expansie 37
    formele parameter 39
    genestelde oproep 41
    hoofding 39
    lokale variabele 44
    oproep 37, 39
  MCREINDE 39
  MEVA 43
  MFOUT 43
  MNTS 43
  MSPR 43
  MVSP 43
  sleutelwoord 40
  symboolmanipulatie 37
  toestandsdiagram 42
voorwaartse referentie, zie symbolisch adres

W

WEK 126
wekker 196