

**stichting
mathematisch
centrum**



REKENAFDELING

MR 141/72

NOVEMBER

D. GRUNE
BESCHRIJVING VAN DE X8-SIMULATOR X8S

2e boerhaavestraat 49 amsterdam



Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

0. Inhoudsopgave.

1. Doel.
2. Overwegingen.
 - 2.1. Redenen.
 - 2.2. De eigenschappen van de geïmplementeerde machine.
 - 2.3. Uitbreidingen.
 - 2.4. Programmacorrectheid.
3. Faciliteiten en beperkingen.
4. Beschikbare procedures en variabelen.
 - 4.1. Algemene eigenschappen.
 - 4.2. Het monitordeel.
 - 4.3. De registers.
 - 4.4. Het geheugen.
 - 4.5. Console-procedures.
 - 4.6. Hulp-procedures en -variabelen.
5. Details van de implementatie.
 - 5.1. De identificatie van de opdrachtcode.
 - 5.2. Het kerngeheugen.
 - 5.3. De geheugenprotectie.
 - 5.4. De kleine registers.
 - 5.5. De +0-preferentie.
 - 5.6. De vaste-komma-vermenigvuldiging en -deling.
 - 5.7. De schuif- en normeeropdrachten.
 - 5.8. De drijvende-komma-opdrachten.
 - 5.9. Het ingreepmechanisme.
 - 5.10. CHARON.
 - 5.11. De dynamische stoptoestand.
6. Mogelijke uitbreidingen.
 - 6.1. CHARON.
 - 6.2. De tijdmeting.
7. Lijst van foutmeldingen.
8. Het programma.
 - 8.1. Efficiëntie en geheugenbeslag.
 - 8.2. Gebruikte procedures.
 - 8.3. De tekst van het programma.
9. Het testprogramma.
 - 9.1. Algemeen.
 - 9.2. De tekst van het testprogramma.
10. Detailtests op de EL X8.

1. Doel.

Het programma X8S simuleert machine-opdrachten van de EL X8 in ALGOL 60; het is geheel in ALGOL 60 geschreven, zonder gebruikmaking van speciale codeprocedures. Het gehele opdrachtenrepertoire van de EL X8 wordt gesimuleerd, inclusief de geheugenprotectie en de besturingstoestand; van CHARON is alleen de IP-faciliteit geïmplementeerd. Voor het testen van machinecodeprogramma's zijn uitgebreide hulpmiddelen aanwezig, welke toegankelijk zijn via een speciale stuurtaal.

2. Overwegingen.

2.1. Redenen.

Het ALGOL 60 Millisysteem is ongeschikt om als basis te dienen voor het testen van machinecodeprogramma's; dit testen kan daarom alleen op speciale tijden geschieden, met alle ongemakken van dien. Een in ALGOL 60 geschreven simulator is echter een gewoon ALGOL 60-programma dat op normale tijden gedraaid kan worden, zonder daarbij de programma's van andere gebruikers in gevaar te brengen. Onder een aantal omstandigheden is de werking van het CRO van de EL X8 ongedefinieerd (zie Reference Manual A4.4.3), iets wat tijdens het testen van een programma zeker ongewenst is. Een simulator kan onder deze omstandigheden passende maatregelen nemen. Het is vrij omslachtig en bewerkelijk het preciese verloop van een machinecodeprogramma van achter de console te volgen; een simulator kan echter op elk gewenst ogenblik verslag doen over de inhoud van registers en geheugenplaatsen. Anderszijds kan men van achter de console onderzoek doen aan onverwachte verschijnselen, terwijl het uiteraard niet mogelijk is voor de simulator een stuurprogramma te schrijven dat passend reageert op onvoorziene omstandigheden. De opbouw van de simulator is echter zodanig dat een fout door de simulator waarschijnlijk eerder opgemerkt wordt dan door de EL X8, terwijl als standaardafwerking van niet door de programmeur voorziene fouten de inhouden van alle registers en beschreven geheugenplaatsen afgedrukt worden.

2.2. De eigenschappen van de geïmplementeerde machine.

Bij het schrijven van het programma deed zich herhaaldelijk de vraag voor, wat er precies geïmplementeerd moest worden. Het is duidelijk dat de simulator betere eigenschappen moet hebben dan de EL X8 wat betreft de afwerking van "undefined" situaties; echter moeten deze eigenschappen niet zo zijn dat als het ware een nieuwe betere machine ontstaat waarop een incorrect programma de schijn van correctheid zou kunnen ophouden. Zo bevat de instructie $F = M[56]$ een fout adres waarop door de machine niet gereageerd wordt. De simulator moet reageren, mag echter geen adresingreep (de normale reactie op een fout adres) geven; dit soort fouten wordt door de simulator behandeld alsof de EL X8 erdoor in een statische stop raakt.

We kunnen de afloop van een programma P op een (al dan niet gesimuleerde) machine M aanduiden als $M(P)$; deze grootte kan voor de EL X8 drie waarden hebben: "correct" (wat dat ook moge zijn), "undefined" en "statische stop". Bij het schrijven van de simulator is getracht de volgende drie regels aan te houden.

Als X8(P) = "correct" dan X8S(P) = "correct". (1)

Als X8(P) = "undefined" dan X8S(P) = "statische stop". (2)

Als X8(P) = "statische stop" dan X8S(P) = "statische stop". (3)

Op regels (1) en (3) zijn geen uitzonderingen (zie echter 2.4.), regel (2) daarentegen kon niet altijd zinvol gehandhaafd worden (zie daarvoor 3.5.). Twee bijzondere gevallen van "undefined" toestanden moeten hier genoemd worden.

Het eerste is het uitlezen van een door het onderhavige programma nog niet beschreven geheugenplaats. Het effect hiervan is uiteraard ongedefinieerd. De simulator houdt bij, welke geheugenplaats beschreven is en welke niet; hiervoor is 1 Boolean per geheugenplaats nodig. Om efficiëntieredenen is deze Boolean dezelfde als die welke als gesimuleerd geheugenpariteitsbit gebruikt wordt. Met andere woorden, een geheugenplaats is beschreven als hij een woord van juiste pariteit bevat, en is onbeschreven als hij een woord van foute pariteit bevat. Bij het starten van de simulator wordt het gehele geheugen gevuld met woorden van foute pariteit. Dit alles heeft wel tot gevolg dat in tegenspraak met regel (2) hierboven een pariteitsingreep optreedt als een niet beschreven geheugenplaats uitgelezen wordt terwijl OV true is en M[25] gevuld (voor deze laatste eis zie volgende alinea).

Het tweede geval van een bijzondere "undefined" toestand treedt op wanneer een foutingreep plaats heeft (met OV true) terwijl de bijbehorende ingreepplaats (M[25] tot M[27]) niet gevuld is. Aan de hand van de alinea hierboven zou men zich dan het volgende kunnen voorstellen:

- a. er treedt een foutingreep op
- b. OV := false
- c. de betreffende ingreepplaats wordt uitgelezen
- d. hierdoor treedt een pariteitsingreep op
- e. OV = false, dus "statische stop"

Dit heeft echter het nadeel dat alle niet voorziene ingrepen zouden eindigen in een statische stop op foute pariteit; daarom geven foutingrepen, waarbij de ingreepplaats niet gevuld is, direct aanleiding tot een statische stop, alsof OV false was.

Beide bovenstaande complicaties hadden voorkomen kunnen worden door voor "gevuld zijn" en "juiste pariteit" twee afzonderlijke Boolean's te nemen.

2.3. Uitbreidingen.

De console van de EL X8 stelt de programmeur beperkte testhulpmiddelen ter beschikking; deze hulpmiddelen zijn in de simulator gegeneraliseerd. Zo bestaat er een simultane SVA (Stop Volgend Adres) op een willekeurig aantal adressen. Verder zijn een aantal nieuwe faciliteiten toegevoegd; zo bestaan er een Stop op Schrijven op gegeven Adres, analoog aan SVA, en een circulaire lijst waarin de laatste 32 sprongadressen bewaard worden. Voor een behandeling hiervan zie 3..

2.4. Programmacorrectheid.

Bij het streven naar een correct programma deden zich twee moeilijkheden voor.

2.4.1. De onbekendheid met de EL X8.

Als bron van kennis van de EL X8 heeft voornamelijk gediend het "EL X8 Reference Manual" en wel in het bijzonder de hoofdstukken A3, A4, A5, A6 en A7. In een aantal gevallen bleek deze beschrijving echter onvolledig of onjuist; in deze gevallen werd aanvullende informatie gezocht en soms verkregen uit de tijdsdiagrammen van de EL X8, uit tests en uit de tekst van de testprogramma's die door Electrologica geleverd zijn. Bij deze analyses bleek vaak dat een beantwoorde vraag twee nieuwe vragen opwierp. Voor een volledig verslag van de analyses en voor onbeantwoord gebleven vragen zie hoofdstuk 10..

Er zijn bezwaren aan te voeren tegen het implementeren van eigenschappen die door tests en dergelijke ontdekt zijn en die niet expliciet in het EL X8 Reference Manual voorkomen. Soms zijn deze eigenschappen echter redelijk en worden ze bevestigd door de tijdsdiagrammen (bv. 10.10.), soms is zonder aanvullende informatie implementatie niet mogelijk. Zo bevat de beschrijving van de stapelende subroutine sprong (Reference Manual A4.11.3) geen informatie over het toegestane bereik van B. De tekst impliceert dat het om een STATB-adressering gaat, zodat $B = -0$ toelaatbaar is; dit blijkt niet zo te zijn, bij sommige adresvarianten van de opdracht SUBC is zelfs $B = +0$ ontoelaatbaar (zie 10.16.).

2.4.2. Efficiëntie versus correctheid.

Vaak bleken efficiëntie en correctheid elkaars vijanden te zijn. Dit gold in het bijzonder voor de shiftopdrachten. Hier heeft men de keus tussen verscheidene implementaties. Men kan de registers "ontbinden" in Boolean array's, deze schuiven zoals beschreven in het Reference Manual en vervolgens weer tot registers opbouwen. Dit is gemakkelijk correct te krijgen maar bijzonder inefficiënt. Het voor de hand liggende alternatief is om met behulp van de operator : de schuifopdrachten na te doen. Dit is zeer efficiënt, maar wel wordt dan de correctheid ernstig bedreigd door de -0 -preferente arithmetiek van de EL X8. Bij deze en soortgelijke problemen werd gepoogd een rigoureuze correcte oplossing te vinden.

3. Faciliteiten en beperkingen.

De communicatie tussen programmeur en simulator wordt onderhouden door de procedure master, welke door de simulator aangeroepen wordt zodra een bijzondere situatie ontstaat. De globale integer variabele 'error number' geeft dan aan om welke reden de procedure master aangeroepen werd; een lijst van foutnummers is te vinden in hoofdstuk 7..

De programmeur kan gebruik maken van een standaard procedure master of zelf een procedure hiertoe schrijven. In het eerste geval heeft hij de beschikking over alle faciliteiten van de stuurtaal MASTER, welke elders beschreven zal worden; in het tweede geval heeft hij de beschikking over het ALGOL 60 systeem en moet hij de bijbehorende administratie zelf voeren, zoals beschreven in hoofdstuk 4..

Een tweede communicatiemogelijkheid tussen programmeur en simulator wordt gevormd door het trace-mechanisme. Hierdoor kan de programmeur zich op verzoek doorlopend op de hoogte laten houden van het verloop van de opdrachten.

Ook hier kan de programmeur gebruik maken van de standaard trace-procedures of zelf hiertoe procedures schrijven. De benodigde procedure staan beschreven in hoofdstuk 4..

De volgende faciliteiten (met hun beperkingen) zijn in de simulator aanwezig.

- 3.1. Voor elke geheugenplaats is een Boolean aanwezig welke aan het begin op false gezet wordt, en die aangeeft of 'master' aangeroepen moet worden als de betreffende plaats als opdracht uitgevoerd zou worden.
- 3.2. Voor elke geheugenplaats is een Boolean aanwezig welke aan het begin op false gezet wordt, en die aangeeft of 'master' aangeroepen moet worden als de betreffende plaats beschreven zou gaan worden.
- 3.3. Aan het begin worden de pariteitsbits van alle geheugenplaatsen op "foute pariteit" gezet. Dit testhulpmiddel is in zekere zin beperkend; het is namelijk nu mogelijk in een machinecodeprogramma te rekenen op een pariteitsingreep die op de EL X8 (waarschijnlijk) niet zou optreden.
- 3.4. Aan het begin worden de registers van de geheugenprotectie (de DP's en GP's) allemaal op true gezet, zodat elk verlaten van de besturingstoestand (opzettelijk of niet) zonder dat een bruikbaar protectiepatroon gezet is, een protectieingreep geeft. Deze faciliteit is in dezelfde zin beperkend als die in 3.3..

- 3.5. Volgens de overwegingen in 2.2. zou elke situatie waarin de inhoud van een register "undefined" wordt, moeten leiden tot een statische stop (d.w.z. een aanroep van 'master'). In sommige gevallen is echter deze ongedefinieerdheid inherent aan de, overigens legale, situatie. Met name is bij foutingrepen niet gedefinieerd hoever de lopende opdracht afgemaakt is (zie b.v. Reference Manual A5.2.3.); zo kan na de uitvoering van de opdracht U,GOTO(BAD ADDRESS), terwijl OF 1 is en BAD ADDRESS een woord van foute pariteit bevat, OF 0 of 1 zijn. Het is niet zinvol om op dergelijke situaties te testen en dan statisch te stoppen. Een oplossing zou zijn, van de bedreigde registers (voornamelijk B, OT, OF, IV en BT) bij te houden of ze "undefined" zijn, en bij gebruik een foutmelding te geven. Dit zou echter een vrij uitgebreid programma vereisen, en niet efficiënt zijn.
- Voor dit type van ongedefinieerdheid zijn in de simulator geen voorzieningen getroffen. De volgorde waarin de handelingen binnen een opdracht worden uitgevoerd is in de simulator vrijwel altijd gelijk aan die in het Reference Manual; hieruit volgt dan direct of bij een bepaalde ingreep een bepaalde handeling wel of niet is uitgevoerd.
- 3.6. In de integer variabele 'ccs' wordt het aantal cycli (van 1.25 microsec.) sinds het begin van het programma bijgehouden. Dit aantal wordt voor de drijvende-komma-opdrachten niet precies uitgerekend; in plaats daarvan wordt een minimum- en een maximum-aantal berekend, het minimum wordt bij 'ccs' geteld en het verschil van maximum en minimum wordt bij 'ccs of' geteld. Wanneer 'ccs' een bepaalde waarde overschrijdt, wordt de procedure master aangeroepen; dit is de enige beveiliging tegen loops door sprongen.
- 3.7. De integer variabele 'dcs' telt het aantal direct op elkaar volgende DO/DOS-opdrachten, en wordt op 0 teruggezet zo gauw een niet-DO/DOS-opdracht aangeboden wordt. Overschrijdt 'dcs' een bepaalde waarde, d.w.z. was de keten van DO/DOS-opdrachten langer dan een bepaalde lengte, dan wordt de procedure master aangeroepen. Op deze manier wordt voorkomen dat het te simuleren programma in een loop van DO/DOS-opdrachten raakt.
- 3.8. Bij elke sprong-opdracht wordt de (oude) waarde van OT opgenomen in de lijst van laatste 32 sprongadressen. Bij elke subroutinesprong wordt de link opgenomen in de lijst van laatste 32 sprongadressen. Deze lijst kan door een procedure-aanroep afgedrukt worden.
- 3.9. Als de boolean variabele 'trace' true is, worden de volgende situaties in de simulator aan de programmeur kenbaar gemaakt:
- lezen uit een geheugenplaats
 - schrijven in een geheugenplaats
 - opdracht beëindigd
 - opdracht geskipt
 - DO/DOS-opdracht beëindigd
 - ingreep-opdracht begonnen.

- 3.10 Om de directe in- en uitvoer van gegevens in een machinecodeprogramma te vergemakkelijken, heeft de simulator een (pseudo-) machineopdracht die een aantal ALGOL 60-procedures toegankelijk maakt.

Het bitpatroon van deze opdracht is:

1 1 0 1 0 0 x 0 0 0 x x x x x x x x x x x x x x x x x.

De betekenis van de met x aangeduide bits is de volgende:

<u>bits</u>	<u>waarde</u>	<u>betekenis</u>
20	0	geen B-modificatie
	1	B-modificatie
16,15	00	geen conditievollgende variant
	01	geen betekenis
	10	Y-variant
	11	N-variant
14 t/m 0		n

Uit bits 14 t/m 0 en eventueel de inhoud van B wordt een procedurenummer opgebouwd; de overeenkomstige procedure wordt dan aangeroepen. De huidige implementatie bevat 16 procedurenummers, (de getallen 0 t/m 15) die vrijwel allemaal transporten via het S-register uitvoeren. Bits 1 en 0 specificeren het I/O-mechanisme:

0	uitvoer regeldrukker
1	invoer
2	uitvoer bandponser
3	uitvoer kaartponser

terwijl bit 3 en 2 aangeven hoe de inhoud van het S-register beschouwd moet worden:

0	als octade
1	als resym-waarde (zie LR1.1., hoofdstuk 6)
2	als octaal getal van 27 bits
3	als decimaal getal.

Beschrijving:

conditievollgende varianten

B-modificatie

bewerking

als m = 0

als m = 1

als m = 2

Y, N

toegestaan

t:= 1 als B-modificatie

anders t:= 0

m:= n + t × B

x:= S

de procedure new page wordt aangeroepen

x:= rehep

puhep(x)

als m = 3	col(x)
als m = 4	prsym(x)
als m = 5	x:= resym
als m = 6	pusym(x)
als m = 7	csym(x)
als m = 8	x wordt als octaal getal
	afgedrukt in 9 cijfers
als m = 9	van het invoermedium wordt
	een octaal getal gelezen
	(tussen apostrofs) en aan
	x toegekend
als m = 12	print(x)
als m = 13	x:= read
als m = 14	punch(x)
als m = 15	cpunch(x)
conditiezettende variant	niet toegestaan
undisturbed-variant	niet toegestaan
hoofd functie	S:= x

4. Beschikbare procedures en variabelen.

Hieronder volgt een lijst van procedures en variabelen die gebruikt kunnen worden bij het schrijven van andere versies van de procedure master of van de trace-procedures, en bij het wijzigen van de simulator zelf. Het is geen uitputtende lijst van alle aanwezige namen, maar de hier niet genoemde namen zijn hulpgrootheden bij de hier wel genoemde.

4.1. Algemene eigenschappen.

In het buitenblok van de simulator zijn drie variabelen beschikbaar waarmee bepaalde eigenschappen van de simulator beïnvloed kunnen worden.

4.1.1. integer aantal kasten

geeft het aantal geheugenkasten aan dat gesimuleerd wordt. Voor een verhoging van 'aantal kasten' met 1 zijn 1853 geheugenplaatsen nodig en 16384 plaatsen op de trommel.

4.1.2. integer bufs

geeft de grootte van de buffers van de geheugensimulatie (zie 5.2.) aan. De waarde van 'bufs' mag niet lager dan 64 zijn. Voor een verhoging van 'bufs' met 1 zijn 4 geheugenplaatsen nodig.

4.1.3. integer max master

geeft de grootte aan van het werkarray van de standaardprocedure 'master'. Voor de verhoging van 'max master' met 1 is 1 geheugenplaats nodig.

4.2. Het monitordeel.

4.2.1. procedure start monitor

Deze procedure wordt door de simulator aangeroepen als laatste stap in de initialisatie. Hij moet het te simuleren programma inlezen (met 'ip', 'bi' of anderszins) en starten ('lsip', 'lsbi', 'lsnb' of 'bva').

4.2.2. procedure master

Deze wordt door de simulator aangeroepen zodra interventie van het master-programma noodzakelijk wordt. De integer error number geeft dan de reden aan; deze foutnummers zijn in de volgende klassen ingedeeld:

0	-	99	bijzondere omstandigheden
100	-	199	foutieve gegevens voor een opdracht
200	-	299	opdracht onherkenbaar
300	-	399	adres onjuist
400	-	499	pariteitsfout
500	-	599	protectiefout
600	-	699	onjuist CHARON-gebruik

Voor de procedure master zijn vooral de foutnummers die een bijzondere omstandigheid aangeven, van belang:

- 0 einde opdracht terwijl 'sva' true is
- 1 het programma is in een dynamische stop geraakt
- 2 het programma heeft het maximaal toegestane aantal cycli overschreden
- 3 het programma heeft het maximaal toegestane aantal opeenvolgende DO/DOS-opdrachten overschreden
- 4 een door 'no exec' gemerkte opdracht is in OR gehaald (OF is nog niet opgehoogd)
- 5 een door 'no write' gemerkte geheugenplaats zal beschreven worden.

De procedure master mag niet normaal terugkeren naar het hoofdprogramma, maar moet eindigen met een aanroep van 'bva', 'lsip', 'lsbi' of 'lsnb', welke alle vier uiteindelijk de simulator op het juiste punt herstarten. Dit geldt niet voor aanroepen met 'error number' gelijk aan 4 of 5; hiervan mag normaal teruggekeerd worden, het lopende proces wordt dan vervolgd.

4.2.3. integer error number
zie procedure master.

4.2.4. boolean sva
vervangt de knop SVA op de console. Zie 'master', 'lsip', 'lsbi' en 'lsnb'.

4.2.5. real ccs, ccs of, ccs max
vormen de administratie van het aantal gebruikte cycli. 'ccs' is het minimaal aantal gebruikte cycli, 'ccs of' het (positieve) verschil tussen maximaal en minimaal aantal gebruikte cycli, terwijl 'ccs max' een bovengrens vormt van het aantal te gebruiken cycli. 'ccs' en 'ccs of' worden voortdurend door de simulator bijgewerkt, 'ccs max' blijft ongewijzigd. Als 'ccs' groter dan 'ccs max' blijkt te zijn, wordt 'master' aangeroepen met 'error number' gelijk aan 2.

4.2.6. real dcs, dcs max
worden gebruikt bij het testen op de lengte van DO-loops. Bij elke overgang van een DO/DOS-opdracht op de hierdoor aangewezen opdracht wordt 'dcs' met 1 opgehoogd; aan het einde van elke andere opdracht wordt hij op 0 gezet. Wordt 'dcs' groter dan 'dcs max' dan wordt 'master' aangeroepen met 'error number' gelijk aan 0.

4.2.7. integer array nowrite, noexec[0:max addr]

Beide arrays worden bij het starten van de simulatie op false geïnitieerd en worden verder niet door de simulator gewijzigd. De monitor kan bepaalde elementen true maken. Bij het schrijven in een geheugenplaats n onderzoekt de simulator nowrite[n], en als deze true is wordt 'master' aangeroepen met error number = 5; het schrijven heeft dan nog niet plaats gehad. Bij het halen van een instructie uit geheugenplaats n onderzoekt de simulator noexec[n], en als deze true is wordt 'master' aangeroepen met error number = 4; de opdracht bevindt zich dan al in OR maar OT is nog niet opgehoogd. De test op noexec wordt alleen uitgevoerd als de opdracht in OR wordt gehaald doordat OT ernaar verwijst, en niet wanneer hij uitgevoerd wordt ten gevolge van een DO-opdracht of een ingreep.

4.2.8. Het trace-mechanisme.

Afhankelijk van de waarde van de globale boolean trace wordt al dan niet een 'trace' geproduceerd. Deze 'trace' bestaat uit het aanroepen van een van een groep van zes procedures, telkens wanneer er iets belangrijks gebeurt. Van deze procedures wordt verwacht dat ze normaal terugkeren. De zes procedures zijn:

report reading(addr, val):
de simulator heeft zojuist de waarde 'val' gelezen uit de geheugenplaats 'addr',

report writing(addr, val):
de simulator heeft zojuist de waarde 'val' in de geheugenplaats 'addr' geschreven,

report eoi:
de simulator heeft zojuist een opdracht beeindigd,

report skip:
de simulator heeft zojuist een opdracht geskipt,

report eod(addr, instr):
de simulator heeft zojuist een DO/DOS-opdracht beeindigd, welke aangaf dat als volgende opdracht 'instr' uit geheugenplaats 'addr' uitgevoerd moet worden, en

report interrupt(ir addr):
als volgende opdracht wordt de ingreepopdracht in M[ir addr] uitgevoerd.

4.2.9. procedure terminate

wordt door de simulator aangeroepen wanneer er een fouttoestand optreedt terwijl 'in monitor = true'. Bij eventueel vervangen van deze procedure moet men er zorg voor dragen dat 'terminate' zelf zeker geen fouttoestanden kan veroorzaken, daar anders een lawine van fouten ontstaat.

4.3. De registers.

4.3.1. real fh, ft, a, s, b, ot

De registers zijn om efficiëntieredenen als reals gedeclareerd; de simulator zorgt dat ze nooit non-integer waarden gaan bevatten. Het F-register als zodanig bestaat in de simulator niet (zie 5.8.); F-kop is 'fh', F-staart is 'ft'.

4.3.2. real c, iv, lt, of, last par word, nint, ov, bt

Ook de kleine registers zijn als reals gedeclareerd, met de conventie dat true is 1 en false is 0. Een LP-register is niet aanwezig, in plaats daarvan wordt de variabele 'last par word' gebruikt, die niet de pariteitsbit van het laatst uit het geheugen gehaalde woord bevat, maar het laatst uit het geheugen gehaalde woord zelf. De waarde van LP kan hieruit verkregen worden als 'parbit(last par word)' (zie 4.6.2.), terwijl $LP := q$ ekwivalent is met $last\ par\ word := -q$ waarbij $q = 0, 1$.

4.3.3. real procedure tlink

Het T-register wordt niet als variabele gesimuleerd (OF is er wel) maar wordt, waar nodig, samengesteld door de procedure tlink, welke T aflevert met bit 26 gelijk aan 0.

4.4. Het geheugen.

4.4.1. procedure stm; real addr, m, cg;

De procedure stm bewerkstelligt het opbergen van de integerwaarde van m in de door addr aangewezen geheugenplaats. 'm' moet een gehele waarde tussen $-2 \uparrow 26 + 1$ en $2 \uparrow 26 - 1$ bevatten, terwijl addr tussen 0 en $2 \uparrow 18 - 1$ moet liggen; blijkt addr groter dan 'max addr' te zijn, dan wordt, als de opdracht door de monitor uitgevoerd wordt, het programma door een aanroep van de procedure terminate (zie 4.2.9.) beëindigd, terwijl anders een adresinggreep volgt. De waarde van cg moet 0 of 1 bedragen; als $cg = 0$ wordt het woord met juiste pariteit weggeschreven, als $cg = 1$ met foute pariteit.

4.4.2. procedure mem; real addr, m, cg;

De waarde die zich bevindt op de door addr aangewezen geheugenplaats wordt in m gehaald; addr moet aan dezelfde voorwaarden voldoen als bij stm (zie 4.4.1.). De waarde van cg bepaalt de verwerking van de pariteitsbit, en wel als volgt:
als $cg = 0$ wordt LP niet beïnvloed en veroorzaakt een pariteitsfout een pariteitsingreep,
als $cg = 1$ krijgt LP de waarde van de pariteitsbit en veroorzaakt een pariteitsfout een pariteitsingreep,
als $cg = 2$ krijgt LP de waarde van de (eventueel foutieve) pariteitsbit en wordt op een pariteitsfout niet gereageerd,
als $cg = 3$ krijgt LP de waarde van de (eventueel foutieve) pariteitsbit en krijgt C de waarde 0 als de pariteit juist en 1 als hij fout is.

- 4.4.3. integer max addr
De waarde van max addr is het grootste in de gesimuleerde machine beschikbare geheugenadres.
- 4.4.4. integer array dgp[0:max addr : 512]
Om het veelvuldige (dure) indiceren van Boolean arrays tegen te gaan, zijn de DP- en GP-registers gecombineerd tot een integer array dat voor elke aanwezige geheugenpagina (zie Reference Manual A5.2.1) 1 element bevat; de waarde van dit element is dan $(\text{if gp then } 2 \text{ else } 0) + (\text{if dp then } 1 \text{ else } 0) - 1$. De vraag of een pagina n tegen schrijven beschermd is, luidt dan $\text{dgp}[n] > 0$, terwijl de vraag of een pagina n tegen lezen beschermd is, neerkomt op $\text{dgp}[n] > 0$.
- 4.4.5. integer drum cnt
is het aantal trommeltransporten uitgevoerd ten behoeve van de geheugensimulatie.
- 4.4.6. integer array filled[0:max addr]
De variabele $\text{filled}[n]$ dient als pariteitsbit van geheugenplaats n; hij wordt aan het begin van de simulatie op false geïnitieerd.
- 4.5. Console-procedures.
Met een aantal van de hieronder vermelde procedures kunnen console-functies uitgevoerd worden; de overige stellen enkele functies beschikbaar die niet op de EL X8 voorkomen maar wel verwant zijn aan bestaande functies.
- 4.5.1. procedure bva
start het te simuleren machinecodeprogramma op het adres dat in OT staat. Door het aanroepen van deze procedure verlaat men de monitortoestand.
- 4.5.2. procedure ls
simuleert de druktoets LS (zie Reference Manual A7.2.3).
- 4.5.3. procedure lsip
simuleert het achtereenvolgens indrukken van de druktoetsen LS en IP. Een IP-band wordt ingelezen, de IF van de IP-lezer wordt true gemaakt, OT wordt $2 \uparrow 18 - 1$, en als sva false is, wordt het te simuleren programma gestart; evenals op de EL X8 volgt dan onmiddellijk een CHARON-ingreep. Als sva false is, verlaat men door een aanroep van deze procedure de monitortoestand.
- 4.5.4. procedure lsnb
simuleert het achtereenvolgens indrukken van de druktoetsen LS en NB. De IF van apparaat 38 wordt true gemaakt, OT wordt $2 \uparrow 18 - 1$, en als sva false is volgt een CHARON-ingreep; in dat geval verlaat men de monitortoestand.
- 4.5.5. procedure lsbi

is analoog aan lsip, met dien verstande dat de band niet als IP-band gelezen wordt maar als BI-band (zie MR132).

4.5.6. procedure ip
is een zuivere inleesprocedure, die onafhankelijk van de stand van de registers een IP-band inleest, de registers ongewijzigd laat en de monitortoestand niet opheft.

4.5.7. procedure bi
is analoog aan 'ip', met dien verstande dat de band niet als IP-band gelezen wordt maar als BI-band.

4.5.8. boolean key
wordt op false geïnitieerd en verder door de simulator niet gewijzigd. Wordt deze variabele door de monitor true gemaakt, dan volgt aan het eind van de lopende opdracht een sleutelgreep.

4.6. Hulp-procedures en -variabelen.

4.6.1. real procedure compf(head, tail); value head, tail; real head, tail;

De parameters head en tail moeten integer waarden hebben; de afgeleverde waarde is een real waarvan bits 53-27 gelijk zijn aan bits 26-0 van head, bit 26 gelijk is aan bit 14 van head en bits 25-0 gelijk zijn aan bits 25-0 van tail.

4.6.2. integer procedure parbit(n); value n; integer n;
De afgeleverde waarde is 1 als n als machinewoord beschouwd even pariteit heeft, en 0 als n oneven pariteit heeft. Deze procedure kan gebruikt worden om uit last par word (zie 4.3.2.) de waarde van LP te bepalen.

4.6.3. integer procedure oct(val); value val; real val;
Deze procedure kan gebruikt worden om octale constanten te specificeren. De waarde van 'val' moet een geheel getal zijn tussen 0 en 777 777 777; men denke zich dit getal neergeschreven als decimaal getal van 9 cijfers; deze string van 9 cijfers wordt nu beschouwd als de octale representatie van een machinewoord van de EL X8; de hiermee corresponderende waarde wordt door de procedure afgeleverd. Zo is oct(10) = 8, oct(75) = 61 en oct(777 777 776) = -1. De werking is ongedefinieerd als niet aan de hierboven genoemde voorwaarde is voldaan, of als de ontstane string van 9 cijfers geen representatie van een octaal getal is.

4.6.4. integer procedure reoct
leest een octaal getal in, dat voorafgegaan moet zijn en gevolgd moet worden door een apostrof. Voorafgaand aan de eerste apostrof mogen geen cijfers voorkomen. Tussen de cijfers van het octale getal mag telkens 1 spatie voorkomen.

4.6.5. procedure proct(m, dig); value m, dig; integer m, dig;

De waarde van m wordt als octaal getal afgedrukt. Als dig = 6 wordt het getal afgedrukt in 6 cijfers, anders wordt het in 9 cijfers afgedrukt, met een spatie tussen het derde en vierde cijfer. Het getal wordt voorafgegaan en gevolgd door een spatie.

- 4.6.6. procedure dump(from, to); value from, to; integer from, to;
Van het geheugentraject van 'from' tot 'to' wordt een octale dump op de regeldrukker geproduceerd. Er worden acht geheugenplaatsen per regel afgedrukt. Als een geheugenplaats niet gevuld is, wordt de overeenkomstige plaats met spaties gevuld; zijn alle acht geheugenplaatsen van een regel niet gevuld, dan wordt de regel niet afgedrukt. In de linker kantlijn worden de adressen van het eerste en het laatste woord van de regel gegeven.
- 4.6.7. procedure print regs
geeft een afdruk van tlink (zie 4.3.3.), de registers F-kop, F-staart, A, S, B en OR, de geheugenvariabelen addr en m (zie 4.4.1. en 4.4.2.), de variabele 'ir addr' die de geheugenplaats aangeeft van de laatst uitgevoerde ingreep-opdracht, een jump counter die het aantal sprongen aangeeft, de variabelen 'ccs' en 'ccs of' (zie 4.2.5.) en de variabele 'drum cnt' (zie 4.4.5.).
- 4.6.8. procedure print tlist
geeft een afdruk van de lijst van de laatste 32 sprongadressen, te beginnen met de meest recente. De vermelde adressen zijn de adressen waarvandaan gesprongen werd. Als het een subroutinesprong betrof, wordt ook de inhoud van de kleine registers gegeven, en wel als de eerste drie octale cijfers van het adres. In het geval van een gewone sprong zijn deze drie posities bezet door spaties.
- 4.6.9. procedure print dgp
geeft een afdruk van de DP- en GP-registers; de registers van elke kast worden op een nieuwe regel afgedrukt.
- 4.6.10. procedure print charon
geeft een afdruk van de AF, IF en LVIF van die apparaten waarvan IF of LVIF of beide variabel zijn.
- 4.6.11. real array tp[0:27]
tp[n] is gelijk aan $2 \uparrow n$.
- 4.6.12. integer tp0, tp1, ... tp24, tp25; real tp26, tp27;
tpN is gelijk aan $2 \uparrow N$.
- 4.6.13. integer tp14m1, tp18m3, tp18m1, tp26m1; real tp27m1;
tpNm1 is $2 \uparrow N - 1$, tp18m3 is $2 \uparrow 18 - 3$.

5. Details van de implementatie.

In dit hoofdstuk wordt de implementatie van een aantal saillante punten behandeld, samen met de overwegingen die tot deze implementatie hebben geleid.

5.1. De identificatie van de opdrachtcode.

De opdrachtcode van de EL X8 is in grote lijnen orthogonaal opgebouwd; hiervan is zoveel mogelijk gebruik gemaakt. De inhoud van OR wordt eerst gesplitst in een aantal hulpvariabelen waarvan de namen aanduiden welke bits uit OR ze bevatten. Zo bevat b21 bit 21, b14t0 bit 14 tot 0, b24c21 bit 24 gevolgd door bit 21 (c van concatenated), b18c17 bit 18 gevolgd door bit 17 en zou b18c21t18c3 achtereenvolgens de bits 18, 21, 20, 19, 18 en 3 bevatten. OR nu wordt gesplitst in b26t22, b21, b20c19, b18c17, b16c15 en b14t0. De opdracht wordt eerst gedifferentieerd naar de eerste 5 bits, d.w.z. naar b26t22; dit levert dus 32 klassen. Twee klassen (te weten 11010 en 11110) zijn ongedefinieerd en twee klassen (11011 en 11111) betreffen adresloze opdrachten die anders behandeld worden. Bij de adreshebbende opdrachten duidt b21 bijna altijd het al of niet inverteren van de geheugenoperand aan; dit geldt niet bij de opdrachten GOTO(:DYN) en GOTOR(:DYN), de klassen 10110 en 10111 en de opdrachten F/x en G/x. Vele van de 28 klassen identificeren direct een opdracht. Aan een aantal klassen moet echter meer zorg besteed worden. De elementen van de switch sb26t22 bevatten een precieze beschrijving van de verdere identificatie. Als de operand van het type :DYN blijkt te zijn krijgt b20c19 de waarde 4; b20c19 wordt namelijk door het adresberekeningsmechanisme gebruikt.

De analyse van de adresloze opdrachten is veel eenvoudiger dan de tabellen in Reference Manual A4.18.4 zouden doen vermoeden. Er is maar een opdracht waarbij b19 = 1, te weten MEMPROT. Door een classificatie volgens b14t12 valt de rest uiteen in 3 klassen:

b14t12 = 000, werkend op de registers,

b14t12 = 110, werkend op de kleine registers, en

b14t12 = 111, werkend op de CHARON-registers.

De eerste groep kan volgens b11t5 gesplitst worden in schuifopdrachten (b11t5 = 0, 1, 2, 3), normeeropdrachten (b11t5 = 5, 7), TENS/TENAS (b11t5 = 32) en de snelle registertransporten (b11t5 = 8,9,10). Hierbij blijkt dat de normeeropdrachten NORA, NORS en NORAS grote overeenkomst vertonen met LUA(0), LUS(0) en LUAS(0).

De tweede groep bevat in b11t9 een nadere precisering van het bedoelde kleine register, 000 voor IVON/IVOFF, 001 voor OVON/OVOFF, 010 voor ITVON, 011 voor CLP en 100 voor INT.

In de derde groep bepaalt bit 11 of de opdracht een transport naar een AF-, IF- of LVIF-register is (b11 = 0) of een transport van een aantal IF- of LVIF-registers naar het A- of S-register is (b11 = 1). In beide gevallen worden de CHARON-registers bepaald door bit 10 en 9.

5.2. Het kerngeheugen.

In principe bevindt het gehele gesimuleerde kerngeheugen zich op de trommel, en wel op de eerste (aantal kasten $\times$ 16384) plaatsen. Om de efficiëntie van een geheugenaccess te vergroten wordt de meest recente inhoud van vier geheugentrajecten (in het programma "regions" genaamd) in 4 arrays, m0, m1, m2 en m3 gehouden; deze arrays zijn 'bufs' woorden lang. Het array m0 bevat altijd de geheugenplaatsen 0 tot bufs - 1; het programma neemt aan dat het D-register m0[63] is, de minimale waarde van bufs is dus 64.

Bij elk geheugenaccess wordt nagegaan of de gevraagde geheugenplaats zich in een van de array's m0, m1, m2 of m3 bevindt. Zo niet, dan wordt nagegaan of een van de array's nog nooit gebruikt is. Is dit het geval, dan is de hele bij de geheugenplaats behorende "region" nog nooit gebruikt, en kunnen we het vrije array aanwijzen als bevattende de gevraagde geheugenplaats. Op deze manier wordt bereikt dat een programma een aanzienlijk aantal geheugenplaatsen kan gebruiken voordat het eerste trommeltransport optreedt.

Zijn alle array's in gebruik, dan wordt nagegaan van welk array het laatste geheugenaccess het langst geleden is. De inhoud van dit array wordt dan naar de trommel geschreven, en vervolgens wordt het array vanaf de trommel gevuld met de "region" die het gevraagde adres bevat.

Elk geheugenaccess wordt geteld in een teller 'm cnt'; de stand van deze teller wordt opgeborgen bij het array waarin zich de gevraagde geheugenplaats blijkt te bevinden.

5.3. De geheugenprotectie.

5.3.1. De protectietoestand en de besturingstoestand van de EL X8 lopen niet altijd parallel. Zo kan bijvoorbeeld CHARON geheugenplaatsen uitlezen of beschrijven onafhankelijk van de waarde van BT; verder geschiedt het halen van de opdracht die ten gevolge van een ingreep uitgevoerd moet worden zonder geheugenprotectie, hoewel BT nog steeds 0 kan zijn; zie ook 10.15.1. De geheugenprotectie is daarom afhankelijk gesteld van een boolean 'prot' die aan het eind van elke opdracht in overeenstemming gebracht wordt met de waarde van BT, maar binnen de opdracht een eigen leven kan leiden; zie bijvoorbeeld de opdracht SUBC.

5.3.2. Op de EL X8 behoort bij elke 4096 geheugenplaatsen een DP-register, en bij elke 512 geheugenplaatsen een GP-register (zie Reference Manual A5.). Als deze als Boolean arrays geïmplementeerd zouden zijn, zou elk geheugenaccess twee delingen en twee Boolean array-indicering kosten. Dit kan als volgt tot 1 deling en 1 integer array-indicering teruggebracht worden.

Bij elke 512 geheugenplaatsen hoort een integer 'dgp' die gelijk is aan $2 \times GP + DP - 1$. Als $GP = 1$ (schrijven en lezen verboden) is dit groter dan 0, als $DP = 1$ en $GP = 0$ (alleen schrijven verboden) is dit 0, en als $DP = GP = 0$ (schrijven en lezen toegestaan) is dit negatief. De vraag of een geheugenplaats tegen lezen beschermd is, komt dus neer op de vraag of de bijbehorende dgp groter dan 0 is, terwijl de vraag of een geheugenplaats tegen schrijven beschermd is, neerkomt op de vraag of dgp groter dan 0 of gelijk aan 0 is. Dit alles heeft wel tot gevolg dat de opdracht MEMPROT en de routine voor het afdrukken van de DP's en GP's ingewikkelder worden.

5.4. De kleine registers.

De implementatie hiervan is behandeld in 4.3.2..

5.5. De +0-preferentie.

Hoewel de numerieke processen in de EL X8 -0-preferent zijn (Reference Manual A4.1.2), is de adresaritmetiek +0-preferent (Reference Manual A3.8.1). In de ALGOL 60-tekst wordt de +0-preferentie verkregen door $a + b$ te vervangen door $-(-a-b)$. Dit is echter alleen juist als niet $a = b = -0$. Bij de adresberekeningen is echter eenvoudig vast te stellen dat een van beide termen altijd positief (of +0) is.

5.6. De vaste-komma-vermenigvuldiging en -deling.

5.6.1. Bij de implementatie van de vaste-komma vermenigvuldiging $AS := S \times x + A$ worden de registers A en S en de operand x beschouwd als getallen van twee cijfers met teken in het 8192-tallig stelsel, d.w.z.:

$$\begin{aligned} S &= ts \times (t \times 2 \uparrow 13 + u) \\ x &= tx \times (y \times 2 \uparrow 13 + z) \\ A &= ta \times (b \times 2 \uparrow 13 + c) \end{aligned}$$

waarin ts , tx en ta de tekens van S , x en A zijn. Verder geldt dat $0 \leq (t, u, y, z, b, c) < 2 \uparrow 13$. De waarde van AS wordt nu:

$$\begin{aligned} AS &= (ts \times tx) \times (t \times y \times 2 \uparrow 26 + (t \times z + y \times u) \times 2 \uparrow 26 + u \times z) \\ &\quad + ta \times (b \times 2 \uparrow 13 + c) = \\ &= ts \times tx \times t \times y \times 2 \uparrow 26 + \\ &\quad (ts \times tx \times (t \times z + y \times u) + ta \times b) \times 2 \uparrow 13 + \\ &\quad (ts \times tx \times u \times z + ta \times c) \end{aligned}$$

Het blijkt nu dat de tweede en de derde term samen, met inbegrip van alle tussenresultaten kleiner is dan $2 \uparrow 40$, en dientengevolge exact door de drijvende-komma-aritmetiek van de EL X8 berekend kan worden. Voor de producten $t \times z$, $y \times u$ en $u \times z$ geldt:

$$\begin{aligned} \max(t \times z) &= \max(y \times u) = \max(u \times z) = (2 \uparrow 13 - 1) \uparrow 2 = 2 \uparrow 26 \\ &\quad - 2 \uparrow 14 + 1 \end{aligned}$$

hetgeen kleiner is dan $2 \uparrow 40$. Verder geldt:

$$P = \max (t \times z + y \times u + b) = 2 \uparrow 26 - 2 \uparrow 14 + 1 + 2 \uparrow 26 - 2 \uparrow 14 + 1 + 2 \uparrow 13 - 1 = 2 \uparrow 27 - 2 \uparrow 15 + 2 \uparrow 13 + 1 < 2 \uparrow 40.$$

$$Q = \max (u \times z + c) = 2 \uparrow 26 - 2 \uparrow 14 + 1 + 2 \uparrow 13 - 1 = 2 \uparrow 26 - 2 \uparrow 13 < 2 \uparrow 40.$$

Dit geeft voor de som van de tweede en de derde term:

$$P \times 2 \uparrow 13 + Q = 2 \uparrow 40 - 2 \uparrow 28 + 2 \uparrow 26 + 2 \uparrow 13 + 2 \uparrow 26 - 2 \uparrow 13 = 2 \uparrow 40 - 2 \uparrow 27 < 2 \uparrow 40.$$

De waarden $ts \times t$, $ts \times u$, $tx \times y$, $tx \times z$, $ta \times b$ en $ta \times c$ kunnen in ALGOL 60 verkregen worden door integer deling:

$$\begin{aligned} t &:= s \div tp13; & u &:= s - t \times tp13; \\ y &:= x \div tp13; & z &:= x - y \times tp13; \\ b &:= a \div tp13; & c &:= a - b \times tp13; \end{aligned}$$

waarin $tp13 \ 2 \uparrow 13$ is. De gevraagde term is dan:

$$(t \times z + y \times u + b) \times tp13 + u \times z + c$$

Dit kan onmiddellijk vereenvoudigd worden door b en c te elimineren:

$$(t \times z + y \times u) \times tp13 + u \times z + a$$

Dit kan herschreven worden tot:

$$\begin{aligned} t \times z \times tp13 + y \times u \times tp13 + u \times z + a &= \\ y \times u \times tp13 + (t \times tp13 + u) \times z + a &= \\ y \times u \times tp13 + s \times z + a & \end{aligned}$$

De grootheden u en z komen nog maar 1 keer voor en kunnen dus gesubstitueerd worden:

$$(s - t \times tp13) \times y \times tp13 + (x - y \times tp13) \times 3 + a$$

De vorm $y \times tp13$ komt nu tweemaal voor en kan van te voren berekend worden:

$$p := y \times tp13; (s - t) \times p + (x - p) \times 3 + a$$

Van deze vorm kan nog steeds bewezen worden dat al zijn tussenresultaten kleiner dan $2 \uparrow 40$ zijn. Zouden we nu echter de twee termen $s \times p$ tegen elkaar gaan wegstrepen, dan wordt het tussenresultaat $x \times s$ groter dan $2 \uparrow 40$.

De vaste-komma-vermenigvuldiging is in bovenstaande vorm geïmplementeerd, het berekenen van de hierboven besproken som kost twee integer delingen, vier vermenigvuldigingen, vier optellingen en aftrekkingen en vier assignments.

- 5.6.2. In het Reference Manual A4. staat een algoritme vermeld voor het bepalen van de tijdsduur van een MULS/MULAS-instructie bij gegeven operand. In woorden gezegd houdt deze algoritme het volgende in. De geheugenoperand wordt van achteren af bit voor bit 'afgekalfd', in principe 1 bit per cyclus; echter, wanneer aan het eind van een cyclus blijkt dat de bit voor de volgende slag gelijk is aan de tekenbit (en er dus niets hoeft te gebeuren), wordt deze bit nog tijdens de lopende cyclus afgewerkt. Verder blijkt dat het aantal cycli alleen bepaald wordt aan de hand van het al of niet gelijk zijn van bepaalde bits aan de tekenbit; vermenigvuldigen met $-x$ kost dus precies evenveel tijd als vermenigvuldigen met $+x$. We kunnen ons dan ook beperken tot 26-bits operanden met een (impliciet) positief teken. Bij een gegeven operand, b.v.

10111000110000110000011011

kunnen we nu aan de hand van bovenstaande algoritme de indeling in cycli bepalen:

1 01 1 1 00 01 1 0 00 01 1 00 00 01 1 01 1

en de vermenigvuldiging kost dus 17 cycli.

Bij het efficient implementeren van deze algoritme doen zich twee moeilijkheden voor. Ten eerste bevat de oorspronkelijke algoritme nog de uitdrukkelijke clause dat, wanneer de algoritme stopt op bit 25, er nog een cyclus uitgevoerd moet worden. Bij nader inzien is deze clause overbodig. Laten we de algoritme in het geval van stoppen op bit 25, nog een slag uitvoeren, dan wordt de vereiste cyclus vanzelf toegevoegd; weliswaar stopt de algoritme dan op de niet-bestaande bit 27, maar dat doet geen kwaad. Het loop-criterium moet dus zijn ' $k < 26$ '; bit 25 en 26 behoeven dan geen bijzondere behandeling.

De tweede moeilijkheid zit dieper. Bovenstaande algoritme doorloopt het getal van rechts naar links; het is programmatisch echter veel efficiënter een getal van links naar rechts af te tasten (delen door de grootste integer en vervolgens voor elke bit vermenigvuldigen met 2 en de entier nemen). Gevraagd dus deze algoritme zo aan te passen dat het getal van links af ontleed wordt.

Het is duidelijk dat rechts van elke 1 een scheiding optreedt; deze is ook van links af gemakkelijk genoeg te vinden. Bij de nullen doet zich echter een schijnbaar onoplosbaar probleem voor. Zoals uit het voorbeeld blijkt, begint een even aantal nullen met een groep van een nul, een oneven aantal daarentegen met een groep van twee nullen. Dit kan van links naar rechts uiteraard onmogelijk ontdekt worden. Anders gezegd, we kunnen van links naar rechts wel de groepen 1, 00 en 01 herkennen maar niet de groep 0. Echter, in principe zijn we niet geïnteresseerd in de indeling, maar in het aantal groepen. Het blijkt nu dat het herkennen van al een 1, 00 en 01 genoeg is. Een oneven aantal nullen gevolgd door een 1, met als oorspronkelijke structuur b.v.

00 00 00 01

wordt van links naar rechts herkend als

00 00 00 01,

terwijl een even aantal nullen gevolgd door een 1, b.v.

0 00 00 01

herkend wordt als

00 00 00 1.

De algoritme luidt nu:

- a. Gooi de meest linkse bit weg.
- b. Als die bit een 0 was, gooi dan ook de volgende bit weg.
- c. Tel een cyclus.
- d. Herhaal stap a t/m c tot alle bits op zijn.

In de echte implementatie is hierop nog een verfijning aangebracht. Als stop-criterium wordt daar niet gebruikt de vraag of alle bits op zijn, maar de vraag of de overblijvende bits allemaal 0 zijn. Bij n overblijvende 0-bits moeten nog $(n+1) : 2$ cycli uitgevoerd worden.

5.6.3. De gedachtengang bij de implementatie van de vaste-komma-deling is analoog aan die bij de vaste-komma-vermenigvuldiging.

5.7. De schuif- en normeeropdrachten.

Een precieze beschrijving van de schuif- en normeeropdrachten is te vinden in het Reference Manual A4.9.1 tot en met A4.9.3. Deze beschrijving vat de registers op als Boolean arrays en gebruikt een aantal Boolean hulparrays. Toepassing van deze implementatiemethode zou resulteren in een zeer inefficiënt programma. Anderszijds bestond er een ALGOL 60-programma dat de EL X8-schuifopdrachten met behulp van integer deling en vermenigvuldiging zeer efficiënt simuleerde. Dit programma bleek echter in een aantal gevallen onjuiste resultaten te geven, in het bijzonder bij $A = +0$, $A = -0$, $S = +0$, $S = -0$ bij $n = 26$ en $n = 27$. De fouten bleken inherent aan

de gevolgde methode.

Om een met redelijke zekerheid correct programma te kunnen schrijven werden de volgende twee principes toegepast:

- a. voor de schuifinstructie worden de tekenbits van de registers verwijderd in overeenstemming met de aard van de schuifopdracht, het schuiven zelf wordt gedaan door integer deling en vermenigvuldiging en na de schuifopdracht worden de tekens hersteld.
- b. van elke optelling, aftrekking, deling en vermenigvuldiging wordt bewezen dat de operanden groter dan of gelijk aan +0 zijn en kleiner dan $2 \uparrow 40$, en dat het resultaat aan dezelfde eisen voldoet (zie ook 5.5.).

Het onder a genoemde verwijderen van het teken kan op twee manieren geschieden. Als de tekenbit meeschuift, dan wordt het register opgevat als een positieve integer ter lengte van 27 bits, met andere woorden: bit 27 wordt aritmetisch gemaakt; dit is mogelijk omdat alle registers als reals gedeclareerd zijn. De omzetting geschiedt door de opdracht:

if $1/x < 0$ then $x := x + 2 \uparrow 27 - 1$

Dit laat b.v. +0 ongewijzigd maar verandert -0 in $2 \uparrow 27 - 1$. Bit 27 kan als tekenbit hersteld worden door de opdracht:

if $x \geq 2 \uparrow 26$ then $x := x - (2 \uparrow 27 - 1)$.

Schuift de tekenbit daarentegen niet mee, dan wordt hij in een Boolean opgeslagen en bit 27 wordt nul gemaakt:

$negx := 1/x < 0;$
if $negx$ then $x := x + 2 \uparrow 26 - 1$

Dit laat +0 ongewijzigd maar verandert -0 in $2 \uparrow 26 - 1$. De oorspronkelijke tekenbit kan als volgt hersteld worden:

if $negx$ then $x := x - (2 \uparrow 26 - 1)$.

5.7.1. De schuifopdrachten zijn als volgt geïmplementeerd.

LCA, LCS, RCA en RCS

Hiervoor wordt gebruik gemaakt van de in het Milli-systeem aanwezige procedure 'circ shift'.

LCAS, LCSA, RCAS en RCSA

De beide tekens worden aritmetisch gemaakt, de bits worden verplaatst door integer deling en vermenigvuldiging, en de tekens worden hersteld.

LUA en LUS

De bits worden verplaatst door integer deling en vermenigvuldiging.

RUA en RUS

Deze zijn geprogrammeerd als een eenvoudige integer deling.

LUAS

De tekenbits van A en S worden nul gemaakt, de bits worden verplaatst, de oude tekenbits van A en S worden hersteld en de staart van S wordt met het tekenbit van A aangevuld.

LUSA

Bit 27 van A wordt aritmetisch gemaakt, dat van S wordt verwijderd, de bits worden verplaatst, bit 27 van A wordt als tekenbit hersteld, S krijgt zijn oude teken terug en de staart van A wordt met de tekenbit van S aangevuld.

RUAS

Door een integer deling wordt de nieuwe waarde van A bepaald; de tekenbits van de oude A en van S worden nul gemaakt waarna een nieuwe waarde voor S berekend wordt. S krijgt dan zijn oude tekenbit terug.

RUSA

Door een integer deling wordt de nieuwe waarde van S bepaald; bit 27 van A wordt aritmetisch gemaakt, dat van S wordt gemaakt, er wordt een nieuwe waarde van A berekend waarna bit 27 van A als tekenbit hersteld wordt.

5.7.2. De normeeropdrachten worden als volgt verwezenlijkt.

NORA en NORS

De normering wordt uitgevoerd met behulp van de procedure 'normshift'.

NORAS

Het aantal plaatsen waarover geschoven moet worden, wordt met 'normshift' bepaald. Het schuiven zelf wordt daarna door de opdracht LCAS gedaan.

5.8. De drijvende-komma-opdrachten.

De drijvende-komma-opdrachten van de EL X8 zijn direct in een ALGOL 60-programma beschikbaar in de vorm van de operatoren +, -, × en /. Hiervan wordt door de X8-simulator gebruik gemaakt. Weliswaar wordt het programma hierdoor zeer machine-afhankelijk, maar tegen het alternatief, het uitprogrammeren van de drijvende-komma-opdrachten in bitmanipulaties, bestaan twee bezwaren. Ten eerste staat de precieze werking van deze opdrachten nergens in een toegankelijke vorm beschreven (is misschien echter wel af te leiden uit de bij de EL X8 behorende tijdsdiagrammen), ten tweede zou zo'n implementatie zeer inefficiënt zijn.

5.9. Het ingreepmechanisme.

De EL X8 kent drie soorten ingrepen, de normale ingreep (CHARON-ingreep), de foutingrepen en de sleutelingreep. In al deze drie gevallen wordt buiten de gewone programma-opdrachten om een opdracht uit een van de geheugenplaatsen M[24] t/m M[28] uitgevoerd. Als deze opdracht een subroutinesprong is, wordt deze op andere wijze uitgevoerd dan wanneer de subroutinesprong op de gebruikelijke wijze aangeboden zou zijn; onder omstandigheden is namelijk bit 24 van de weggeschreven link een 1, terwijl OV al 0 te weten of deze opdracht afkomstig is van een ingreep of uit het gewone verloop van het programma voortvloeit. Deze informatie wordt door de variabele 'ingreep type' verschaft en wel als volgt:

ingreep type	betekenis:
0	geen ingreep
1	CHARON/sleutelingreep
2	foutingreep
3	foutingreep tijdens CHARON/sleutelingreep.

Hierbij moet opgemerkt worden dat tijdens een normale ingreep wel een foutingreep kan optreden (het woord in M[24] kan imparitair zijn), maar dat tijdens een foutingreep nooit een normale ingreep kan optreden. Hoe dit ligt bij een sleutelingreep is niet duidelijk.

Bij het verwerken van een ingreep wordt eerst onderzocht of het register ITV true is; in dat geval wordt foutmelding 135 gegeven. Het blijkt namelijk experimenteel dat het onder die omstandigheden 'undefined' is of tijdens de ingreepopdracht ITV nog steeds true is. Vervolgens wordt de geheugenprotectie opgeheven, zonder BT te wijzigen; de opdracht wordt gehaald en verwerkt. Aan het eind van elke opdracht wordt getest of 'ingreep type' groter dan 0 is; is dit het geval, dan was de laatste opdracht uitgevoerd ten gevolge van een ingreep. Het register BT wordt dan true gemaakt, en IV false; hiermee is de verwerking van een ingreep voltooid.

Het onderkennen van de situatie die tot een ingreep aanleiding geeft, geschiedt als volgt.

5.9.1. De normale ingreep.

Deze kan slechts optreden tussen twee opdrachten. Het onderzoek of een ingreep moet volgen, wordt gedaan na elke opdracht. Een normale opdracht moet optreden als IV minstens een hele opdracht lang true is geweest, terwijl voor minstens 1 apparaat geldt dat zijn IF true is en zijn LVIF minstens een hele opdracht lang true geweest is. Dit houdt in dat we:

- de logische operaties op alle 40 IF's en LVIF's efficiënt moeten kunnen uitvoeren, en
- informatie moeten bewaren omtrent de toestand van IV en de LVIF's aan het begin van de laatste opdracht, d.w.z. aan het eind van de een na laatste opdracht.

De een-bit registers AF(n), IF(n) en LVIF(n) zijn niet als Boolean arrays geïmplementeerd, maar als paren integers. De integers if0 en if1 bevatten alle IF's in hetzelfde formaat als waarin ze door respectievelijk de machine-opdrachten IFA(0) en IFA(1) afgeleverd worden; d.w.z. bits 25 t/m 18 van if0 bevatten IF(39) t/m IF(32), bits 17 t/m 0 van if0 bevatten IF(0) t/m IF(17), bits 25 t/m 12 van if1 bevatten IF(18) t/m IF(31) en bits 11 t/m 0 van if1 zijn 0. Hetzelfde geldt mutatis mutandis voor AF, af0, af1 en LVIF, lvif0, lvif1. Dit maakt het weliswaar moeilijker deze registers een waarde te geven (met procedure setq) of hun waarde uit te lezen (met procedure readq), maar levert wel de mogelijkheid op efficiënte wijze na te gaan of aan de voorwaarden voor een CHARON-ingreep voldaan is. Bijvoorbeeld kan de vraag of voor minstens 1 apparaat zowel IF als LVIF true zijn, aldus geformuleerd worden:

$$\text{and}(\text{if0}, \text{lvif0}) + \text{and}(\text{if1}, \text{lvif1}) > 0.$$

Wat betreft de tweede eis, aan het eind van elke opdracht wordt voor elk apparaat nagegaan of zowel IV als de LVIF van dat apparaat true zijn. Deze informatie wordt bewaard tot het eind van de volgende opdracht; als dan voor enig apparaat dit nog steeds geldt, terwijl bovendien zijn IF true is, volgt een ingreep. De informatie wordt opgeslagen in ie0 en ie1 in het hierboven beschreven formaat. Het criterium voor een CHARON-ingreep is dan:

$$\text{iv} = 1 \wedge \text{and}(\text{ie0}, \text{and}(\text{if0}, \text{lvif0})) + \text{and}(\text{ie1}, \text{and}(\text{if1}, \text{lvif1})) > 0.$$

Dit kost ongeveer 1300 mmsec, terwijl dezelfde test op basis van Boolean arrays 18000 mmsec zou kosten, nog afgezien van het samenstellen van het array 'ie'.

Ten gevolge van een CHARON-ingreep wordt het woord in M[24] als opdracht uitgevoerd; dit woord kan echter een foute pariteit hebben. Deze uitzonderingstoestand wordt in de simulator aangegeven door het feit dat 'ingreep type' 3 is; voor de werking op de EL X8 zie echter 10.7.

5.9.2. De foutingrepen.

De noodzaak van een foutingreep wordt op vele plaatsen in de simulator vastgesteld; op deze plaatsen wordt dan de procedure 'undef' aangeroepen. Deze procedure heeft een integer parameter die ontleed wordt in een foutnummer en een ingreepadres (parameter = ingreep adres $\times$ 1000 + foutnummer).

5.9.3. De sleutelingreep.

Een sleutelingreep wordt nooit door de EL X8 zelf veroorzaakt maar altijd door een 'buitengebeuren'. De noodzaak van een sleutelingreep wordt dan ook nooit door de simulator vastgesteld. Wel wordt op de daarvoor geschikte plaatsen getest of de boolean variabele 'key' true is gemaakt door de monitor. In dat geval volgt een sleutelingreep.

5.10. CHARON.

Van CHARON zijn alleen de IP-opdracht en de AF-, IF- en LVIF-registers geïmplementeerd. Wel zijn faciliteiten aanwezig om de simulator uit te breiden met (quasi-) asynchrone in- en uitvoer. Zie hiervoor 6.1.

Tot het repertoire van het centrale rekenorgaan behoren opdrachten voor het uitlezen en zetten van de CHARON-registers. De opdrachten voor het uitlezen zijn zeer eenvoudig te implementeren omdat de vorm waarin de CHARON-registers opgeborgen zijn, precies die is waarin ze door genoemde instructies afgeleverd worden (zie 5.9.1.). De IF- en LVIF-registers kunnen uitgelezen worden, de AF-registers niet, zulks in overeenstemming met de beschrijving in Reference Manual A4.15.1 en in tegenspraak met de mondelinge overlevering.

Bij opdrachten voor het zetten van de CHARON-registers ligt de zaak ingewikkelder. Bij de aanwezige apparaten kunnen zowel AF, IF als LVIF aan en af gezet worden. Bij de niet-aanwezige apparaten blijkt experimenteel dat sommige IF's niet true gemaakt kunnen worden en sommige LVIF's niet false. Aangezien de AF's niet uitgelezen kunnen worden, is iets dergelijks daarvoor niet vast te stellen. Het feit dat voor sommige niet-aanwezige apparaten IF niet true gemaakt kan worden, houdt in dat niet met elk niet-aanwezig apparaat een CHARON-ingreep geforceerd kan worden. De werking van IF en LVIF staat voor alle 40 apparaten gespecificeerd in 10.4.

Naast de integers af0, af1, if0, if1, lvif0 en lvif1 bestaan er zes integers afv0, afv1, ifv0, ifv1, lvifv0 en lvifv1 waarin van alle registers aangegeven staat of het onderhavige register variabel is. Wanneer de simulator een zet-opdracht te verwerken krijgt op een niet-variabel register, wordt een foutmelding gegeven. Het variabel-zijn van IF en LVIF werd bepaald in overeenstemming met de toestand op de EL X8 van het Mathematisch Centrum. Van alle AF's wordt aangenomen dat ze niet variabel zijn; dit weerspiegelt het feit dat geen enkel in- of uitvoerproces geïmplementeerd is. Het variabel-zijn van een register kan in de simulator eenvoudig gewijzigd worden door in de initialisatie in een string een 0 door een 1 te vervangen of omgekeerd.

5.11. De dynamische stoptoestand.

Deze toestand wordt gekarakteriseerd door het feit dat $0 \leq 2 \uparrow 13 - 1$ of aan $2 \uparrow 13 - 2$. Afgezien van operateurshandelingen kan de machine maar op een manier uit deze toestand komen, namelijk door een CHARON-ingreep. Deze nu kan twee oorzaken hebben: (a) de sprong naar de dynamische stoptoestand werd met een GOTOR-opdracht gedaan die de machine horend maakte waardoor een reeds hangende ingreepconditie doorkomt, of (b) een asynchroon CHARON-proces wordt beëindigd waardoor een IF true wordt. Nu komt een ingreep pas als de voorwaarde ervoor minstens een hele opdracht lang vervuld is (zie 5.9.1.). Daarom moet na een sprong naar de dynamische stoptoestand nog minstens een geskippede opdracht uitgevoerd worden; is er na deze opdracht nog steeds geen ingreep gekomen, dan is het wachten alleen nog op de beëindiging van een CHARON-proces. Als er zo'n proces loopt, is 'charon teller' groter dan 0 (zie 6.1.); de voorgeschreven wachttijd wordt dan verondersteld verstreken te zijn en CHARON wordt gewekt; hierdoor

kan dan weer een IF true worden. Loopt er niet zo'n proces, dan kan de (gesimuleerde) machine niet meer uit de dynamische stoptoestand komen; er wordt dan een foutmelding gegeven met foutnummer 1.

6. Mogelijke uitbreidingen.

6.1. CHARON.

6.1.1. CHARON is een aparte computer die onafhankelijk van het CRO loopt en als taak heeft de AF-registers te inspecteren, aan de hand hiervan een apparaat te bedienen en als resultaat hiervan eventueel een IF-register true te maken. Het zou dus als apart programma geïmplementeerd moeten worden; parallelle programma's kunnen echter in ALGOL 60 niet geschreven worden en coroutines al evenmin. We zijn dus gedwongen of het CRO-programma of het CHARON-programma als hoofdprogramma te kiezen en het andere als procedure (of reeks procedures). Uiteraard blijft het CRO-programma hoofdprogramma. Het CHARON-programma moet nu minstens uit een procedure bestaan, die wordt aangeroepen als een van de AF-registers door het hoofdprogramma gewijzigd wordt; deze procedure heet 'attendeer charon af', en heeft 1 parameter, het apparaatnummer. Aan de hand van dit apparaatnummer en waarschijnlijk van verdere informatie uit de apparaatregisters kan nu een dienstregeling voor de te verrichten handelingen opgesteld worden. Een heel eenvoudige dienstregeling zou b.v. zijn, het gehele transport meteen af te werken, inclusief wijzigingen in IFT, AFT, IF en AF. Voor een meer realistische aanpak is het echter nodig een schatting te maken van de tijd die voor het gevraagde transport benodigd zal zijn, en pas na het verstrijken van die tijd, de wijzigingen in IFT, AFT, IF en AF uit te voeren. Het verstrijken van die tijd komt alleen tot uiting in het feit dat het hoofdprogramma verder gaat en daardoor 'ccs' (de geheugencyclus-teller) ophoogt. De procedure 'attendeer charon af' moet dus normaal terugkeren, maar aan het hoofdprogramma de mededeling doen dat na een gegeven aantal cycli het CHARON-programma weer even wil rekenen. Hiertoe wordt het aantal benodigde cycli geassigneerd aan de variabele 'charon teller'; deze wordt door het hoofdprogramma telkens met dezelfde bedragen afgelaagd als waarmee 'ccs' opgehoogd wordt. Wanneer aan het eind van een (CRO-)opdracht de 'charon teller' nul of negatief blijkt te zijn geworden, wordt de procedure 'wek charon' aangeroepen. Deze kan dan verdere administratieve werkzaamheden verrichten.

Bij het simuleren van 1 apparaat komt maar 1 opdracht tegelijkertijd voor en is het eenvoudig uit te zoeken waarom 'wek charon' werd aangeroepen. Bij het simuleren van meer dan een apparaat is het echter aan te bevelen een lijst van gebeurens (events) aan te leggen; elementen worden aan deze lijst toegevoegd (niet noodzakelijkerwijze aan het eind) door de procedure 'attendeer charon af', terwijl 'wek charon' telkens de eerste (voorste) verwerkt en verwijdt.

Bij een natuurgetrouwe implementatie van de CHARON-simulatie moeten o.a. de volgende vragen beantwoord worden:

- a. Wat doet CHARON precies met de apparaatregisters, in het bijzonder, hoe worden AR3 en het adresdeel van AR2 gebruikt? Wat gebeurt er als AFT meteen al 0 is? En wat als de bit 26 van AR3 een 1 is?

- b. Wat gebeurt er als een van de apparaatregisters een woord van foute pariteit bevat? Wat gebeurt er als de werkruimte-registers pariteitsfouten genereren (m.a.w. als ze door CHARON correct beschreven worden, maar bij later teruglezen door CHARON een woord van foute pariteit blijken te bevatten.)?
- c. Wat gebeurt er als de leeswijzer naar een transportspecificatie wijst die geheel of gedeeltelijk buiten het aanwezige geheugen ligt (zowel bij de hoge als bij de lage adressen), of die geheel of gedeeltelijk over de registers heen valt?
- d. Wat gebeurt er als de transportspecificatie woorden van foute pariteit bevat?
- e. Wat gebeurt er als de transportspecificatie een traject specificeert dat geheel of gedeeltelijk buiten het aanwezige geheugen valt of dat over de registers heen ligt?
- f. Wat gebeurt er als het te transporteren traject woorden van foute pariteit bevat?
- g. Wat zijn de mogelijkheden om de inhoud van woord $M[0]$ te transporteren (in verband met het opgeven van 'eerste transportadres - 1')?

Alleen in geval f is het min of meer duidelijk dat de foutdetector in actie zal komen.

- 6.1.2. In de huidige versie van de simulator is het effect van 'attendeer charon af(n)' een aanroep van de procedure master met 'error number= 600 + n'.

6.2. De tijdmeting.

De bepaling van het aantal geheugencycli dat een opdracht kost, is onvolledig.

- 6.2.1. Bij de drijvende-komma-opdrachten worden minimum- en maximum-waarden aangegeven (zie 5.8.); de precieze waarden kunnen echter berekend worden aan de hand van de beschrijvingen in Reference Manual A4.17.2.

- 6.2.2. Bij alle adreshebbende opdrachten werd aangenomen dat de tijd benodigd voor de opdracht opgebouwd is uit twee componenten, de tijd benodigd voor het berekenen van het adres en het halen van de operand en de tijd benodigd voor de opdracht zelf. Hierbij werd aangenomen dat operanden van de types $M[n]$, $M[B+k]$, $MR[q]$ en $:MD[q]$ een 1 geheugenaccess nodig hebben (dus 2 cycli), die van de types $MD[q]$ en $:Mp[q]$ twee geheugenaccessen (dus 4 cycli), die van het type $Mp[q]$ drie geheugenaccessen (6 cycli) terwijl operanden van de types n ($:STAT$) en $:MR[q]$ helemaal geen geheugenaccess nodig hebben. Dit beeld is niet helemaal juist; de volgende gevallen wijken af:

- a. Opdrachten van het type $R = STAT$ of $R = STATB$ zijn een cyclus goedkoper indien de operand in een van de adressen 57 t/m 62 staat.
- b. Opdrachten van het type $R + :DYN$ zijn een cyclus duurder dan uit het bovenstaande volgt.

- c. Opdrachten van types $F = _ : \text{STAT}$ en $F = _ : \text{DYN}$ zijn twee cycli goedkoper.
 - d. Opdrachten van het type $F _ x$ zijn een cyclus goedkoper indien de exponenten van beide operanden beide $+ 0$ zijn en nog eens een cyclus wanneer x van type $: \text{STAT}$ is.
- Bovenstaande afwijkingen zijn niet geïmplementeerd.

7. Lijst van foutmeldingen.

De lijst bevat in de eerste kolom het nummer van de foutmelding, in de tweede kolom eventueel het adres van de geheugenplaats die uitgevoerd wordt, mocht de foutmelding als ingreep afgewerkt worden, en in de derde kolom een verklaring van de fout.

0	zie 4.2.2.
1	het programma is in een dynamische stop geraakt
2	het toegestane aantal geheugencycli is overschreden
3	het toegestane aantal opeenvolgende DO/DOS-opdrachten is overschreden
4	zie 4.2.2.
5	zie 4.2.2.
102	bij de opdracht DIVAS zijn A en S van verschillend teken
103	bij de opdracht DIVA of DIVAS is de absolute waarde van A niet kleiner dan de absolute waarde van de operand
120	bij een F / x opdracht treedt de deling 0 / 0 op
121	bij een G / x opdracht treedt de deling 0 / 0 op
130	in niet-bestuurtoestand wordt een van de instructies AFON, AFOFF, IFON, IFOFF, LVIFON, LVIFOFF aangeboden
131	in niet-bestuurtoestand wordt de instructie IVON of IVOFF aangeboden
132	in niet-bestuurtoestand wordt de instructie OVON of OVOFF aangeboden
133	in niet-bestuurtoestand wordt de instructie ITVON aangeboden
134	in niet-bestuurtoestand wordt de instructie MEMPROT aangeboden
135	de opdracht volgend op de opdracht ITVON veroorzaakt een ingreep
140	bij een schuifopdracht is het aantal plaatsen waarover geschoven moet worden kleiner dan 0 of groter dan 31
141	bij een van de opdrachten IFA, IFAC, IFS, IFSC, LVIFA, LVIFAC, LVIFS, LVIFSC is de parameter niet 0 of 1
142	bij een van de opdrachten AFON, AFOFF, IFON, IFOFF, LVIFON of LVIFOFF met B-modificatie is B negatief
143	bij een van de opdrachten AFON, AFOFF, IFON, IFOFF, LVIFON of LVIFOFF is het apparaatnummer groter dan 39
145	bij een opdracht voor het uitlezen van een CHARON-flip-flop-reeks deugt de flip-flop-aanduiding niet
146	bij een opdracht voor het zetten van een CHARON-flip-flop deugt de flip-flop-aanduiding niet
147	een AFON- of AFOFF-opdracht specificeert een niet-variabele AF
148	een IFON- of IFOFF-opdracht specificeert een niet-variabele IF
149	een LVIFON- of LVIFOFF-opdracht specificeert een niet-variabele LVIF
151	bij het lezen van een octaal getal door de pseudo-opdracht OPERATE komt voor dit getal een cijfer voor
152	een octaal getal dat door de pseudo-opdracht OPERATE ingelezen wordt, wordt niet met een apostrof afgesloten
153	een door de pseudo-opdracht OPERATE ingelezen getal overschrijdt

- de integercapaciteit
- 154 bij de uitvoering van een pseudo-opdracht OPERATE blijkt de code kleiner dan 0 of groter dan 15 te zijn
- 155 bij de uitvoering van een pseudo-opdracht OPERATE blijkt de code gelijk te zijn aan 10
- 156 bij de uitvoering van een pseudo-opdracht OPERATE blijkt de code gelijk te zijn aan 11
- 160 een IP-band bevat een ponsing met een binaire waarde groter dan 127
- 161 het aangevraagde traject bij een dump bevat een niet-aanwezige geheugenplaats
- 171 een BI-band bevat een heptade groter dan 63 en niet gelijk aan 127
- 173 een BI-band bevat een woord van foute pariteit
- 174 in een BI-band zijn de C1 en C2 van een inzetaanwijzing niet beide 0
- 175 in een BI-band zijn de C1 en C2 van een geheugenwoord niet beide 1
- 176 een BI-band bevat een woord dat in een van de registers of in een van de geheugenplaatsen M[0] tot M[23] ingezet moet worden
- 200 opdracht begint met '65'
- 201 opdracht begint met '74' of '75'
- 202 onbekende opdrachtcode, ongeveer JUMP met Z- of E-variant
- 203 onbekende opdrachtcode, ongeveer GOTO(-x)
- 204 onbekende opdrachtcode, ongeveer GOTO met Z- of E-variant
- 205 onbekende opdrachtcode, adresloos
- 206 onbekende opdrachtcode, adresloos
- 212 U-variant bij de opdracht CLP
- 211 U-variant bij de opdracht TENS of TENAS
- 213 U-variant bij een van de opdrachten AFON, AFOFF, IFON, IFOFF, LVIFON, LVIFOFF
- 214 U-variant bij de opdracht IVON of bij de opdracht IVOFF
- 215 U-variant bij de opdracht OVON of bij de opdracht OVOFF
- 216 U-variant bij de opdracht ITVON
- 217 U-variant bij de opdracht MEMPROT
- 218 U-variant bij een schuifopdracht
- 219 U-variant bij de opdracht NORAS
- 222 U-variant bij de opdracht MULAS
- 223 U-variant bij de opdracht MULS
- 224 U-variant bij de opdracht DIVAS
- 225 U-variant bij de opdracht DIVA
- 226 U-variant bij de pseudo-opdracht OPERATE
- 228 onbekende opdrachtcode, ongeveer $U, F + :STAT$
- 229 onbekende opdrachtcode, ongeveer $U, F = :STAT$
- 230 onbekende opdrachtcode, ongeveer $U, F \times :STAT$ of $U, F / :STAT$
- 231 onbekende opdrachtcode, ongeveer $U, :STAT + F$
- 232 onbekende opdrachtcode, ongeveer de pseudo-opdracht OPERATE
- 304 bij een in-opdracht blijkt de geheugenoperand een register te zijn
- 305 bij een uit-opdracht blijkt de geheugenoperand een register te zijn
- 306 bij de opdracht $x = A$ blijkt de geheugenoperand een register te zijn
- 307 bij de opdracht $x = S$ blijkt de geheugenoperand een register te zijn

- te zijn
- 308 bij de opdracht $x = B$ blijkt de geheugenoperand een register te zijn
- 309 bij de opdracht $x + A$ blijkt de geheugenoperand een register te zijn
- 310 bij de opdracht $x + S$ blijkt de geheugenoperand een register te zijn
- 311 bij de opdracht $x + B$ blijkt de geheugenoperand een register te zijn
- 312 bij de opdracht $x = G$ blijkt de geheugenoperand een register te zijn
- 320 26 de adresberekening levert een negatief adres
- 321 26 de adresberekening levert een positief niet bestaand adres
- 322 26 bij een uit-opdracht is de geheugenoperand niet-aanwezig
- 323 26 bij een in-opdracht is de geheugenoperand niet-aanwezig
- 330 bij een opdracht met MC-adressering wordt B negatief
- 331 bij een F-opdracht met MC-adressering wordt B negatief
- 332 bij een opdracht met $Mp[q]$ -adressering is D kleiner dan 64
- 333 het adres van een operand met DYN- of :DYN-adressering wijst een van de registers aan
- 335 bij een opdracht met MC-adressering of bij de opdracht MEMPROT blijkt B niet positief te zijn
- 336 bij een F-opdracht met MC-adressering blijkt B niet positief te zijn
- 341 de operand van de opdracht $G = x$ is $M[57]$ (F-kop)
- 342 de operand van een van de opdrachten $G + x$, $G - x$, $G \times x$ of G / x is $M[57]$ (F-kop)
- 343 de operand van een van de opdrachten GOTO(x), GOTOR(x), JUMP(x) of JUMPR(x) is $M[62]$ (T)
- 344 de operand van de opdracht SUBC(x) is $M[62]$ (T)
- 345 de operand van de opdracht DO(x) of DOS(x) is $M[62]$ (T)
- 350 26 het sprongadres van een van de opdrachten GOTO(x), GOTOR(x), JUMP(x) of JUMPR(x) is negatief
- 351 26 het sprongadres van een van de opdrachten GOTO(x), GOTOR(x), JUMP(x) of JUMPR(x) is niet-aanwezig
- 352 26 het sprongadres van een SUBC(x)-opdracht is negatief
- 353 26 het sprongadres van een SUBC(x)-opdracht is niet-aanwezig
- 354 26 het sprongadres van een REPn(x)-opdracht is niet-aanwezig
- 355 26 het sprongadres van een SUBn(x)-opdracht is niet-aanwezig
- 358 26 door de normale OT-ophoging gaat OT het adres van de eerste niet-aanwezige geheugenplaats bevatten
- 361 26 het protectiewoord van een MEMPROT-opdracht verwijst naar een niet-aanwezige geheugenkast
- 362 26 bij de opdracht SUBC(x) is B negatief
- 363 26 bij een indirecte stapelende subroutinesprong is B gelijk aan +0
- 400 25 pariteitsfout
- 401 25 pariteitsfout in D bij een operand met MD-adressering
- 402 25 pariteitsfout in D bij een operand met $Mp[q]$ -adressering
- 410 25 pariteitsfout in de teller bij een tellende sprongopdracht
- 500 27 poging tot schrijven in een tegen schrijven beschermde geheugenplaats

501 27 poging tot lezen uit een tegen lezen beschermde geheugenplaats
502 27 bij een sprong door een protectiepoort is $B < 256$
503 27 bij een SUBn-opdracht is de plaats van de LINK beschermd tegen
schrijven

600/639 een AFON- of AFOFF-opdracht specificeert een niet-geïmplementeerd
apparaat met een nummer dat 600 lager is dan het foutnummer

999 het programma is door de monitor beëindigd

8. Het programma.

8.1. Efficiëntie en geheugenbeslag.

De efficiëntie van de X8-simulator is in hoge mate afhankelijk van de aangeboden opdrachten; zo zullen drijvende-komma-opdrachten zeer goedkoop zijn, terwijl dubbele-lengte-schuifopdrachten naar verhouding duur zullen zijn. Als richtlijn kan gelden dat de ongeveer 215000 cycli van het in hoofdstuk 9 beschreven testprogramma in ongeveer 435 seconden uitgevoerd worden; dit is ongeveer 2 msec per cyclus, een factor 1500 langzamer dan de EL X8 zelf.

Voor de initialisatie zijn ongeveer 8 sec per geheugenkast nodig. Het programma is ongeveer 21000 geheugenplaatsen groot (hiervan 12000 voor de X8-simulator, en 9000 voor het masterprogramma) en heeft als werkruimte

$$935 + 1853 \times \text{aantal kasten} + 4 \times \text{bufs} + \text{max master}$$

geheugenplaatsen nodig. Bij een beschikbaar geheugen van ruim 40000 plaatsen, vier kasten en 512 geheugenplaatsen voor het masterprogramma kan 'bufs' ongeveer 3000 bedragen.

8.2. Gebruikte procedures.

De X8-simulator gebruikt de volgende procedures uit het Milli-systeem (zie L.R. 1.1.).

abs	fixt	random
absfixt	from drum	read
and	head of	rehep
bit	line number	resym
bitstring	new page	set
circ shift	nler	set random
col	norm shift	sign
compose	print	space
cpunch	print pos	string symbol
csym	printtext	tail of
date	prsym	time
entier	puhep	to drum
even	punch	xor
exit	pusym	

8.3. De tekst van het programma.

```

begin comment x8 simulator: 17-12-71;

  int aantal kasten, max addr, bufs, max master;
  aantal kasten:= 4;
  bufs:= 3000;
  max master:= 512;
  max addr:= aantal kasten  $\times$  2  $\uparrow$  14 - 1;

bgn
  real fh, ft, a, s, b, ot, c, iv, lt, of, last par word,
    nint, ov, bt, or, m, addr, cg, itv, ingreep type, dyst stat;

  co ingreep type   betekenis:
      0             geen ingreep
      1             charon/sleutelingreep
      2             foutingreep
      3             foutingreep tijdens charon/sleutelingreep ;

  real val, valh, valt, ah, at, sh, st, hulp1, hulp2, hulp3;
  bool addrop, regop, memop, maddr, flag, nega, negs, subcd, gate,
    prot;

  switch dsw:= basis cyclus 1;

  int proc randbit; randbit:= random;

  int proc randint;
  bgn real x;
    x:= entier((entier(random  $\times$  tp13) + random)  $\times$  tp14);
    randint:= if x  $\geq$  tp26 then x - tp27m1 else x
  end randint;

  proc init x8;
  bgn
    fh:= randint; ft:= randint; a:= randint; s:= randint; b:= randint;
    ot:= and(randint, tp18m1); or:= randint;
    c:= randbit; iv:= randbit; lt:= randbit; of:= randbit;
    last par word:= randint; nint:= randbit; ov:= randbit; bt:= randbit;
    m:= addr:= cg:= itv:= ingreep type:= dyst stat:= 0;
    prot:= bt = 0
  end init x8;

  proc memreg;
  if mem op then mem1 else
  bgn m:= if addr < 59 then
    (if addr = 57 then fh else ft) else
    (if addr < 61 then
      (if addr = 59 then a else s) else
      (if addr = 61 then b else
        tlink + (if c = 0 then 0 else -tp26m1)

```

```

end memreg;

real proc compf(h,t); val h,t; real h,t;
compf:= compose(h,set(bit(14, h), 26, 26, t));

bool proc signinc;
signinc:= bit(14, valh) †
          (if 1/valt > 0 then 0 else 1);

proc b up1;
bgn if 1/b < 0 then undef(335); b:= b + 1 end b up1;

proc bdown1;
bgn b:= -(1 - b); if b < 0 then undef(330) end bdown1;

proc b up2;
bgn if 1/b < 0 then undef(336); b:= b + 2 end b up2;

proc bdown2;
bgn b:= -(2 - b); if b < 0 then undef(331) end bdown2;

real proc star(x); val x; real x;
bgn co only for abs(x) > tp18m1;
  y:= x - x : tp18 × tp18;
  star:= if y = 0 ∧ x > 0 then 0 else y
end star;

real proc tlink;
tlink:= ot +
  (if c = 0 then 0 else tp13) +
  (if iv = 0 then 0 else tp19) +
  (if lt = 0 then 0 else tp20) +
  (if of = 0 then 0 else tp21) +
  (if parbit(last par word) = 0 then 0 else tp22) +
  (if nint = 0 then 0 else tp23) +
  (if ov = 0 then 0 else tp24) +
  (if bt † 0 then 0 else tp25);

proc calc addr;
bgn mcaddr:= false;
  if b20c19 < 3 then
    bgn co stat, :stat, statb;
      if b20c19 < 2 then
        bgn addr:= b14t0; addrop:= b20c19 = 1 end else
        bgn addrop:= false;
        addr:= (if abs(b) > tp18m1 then b - b : tp18 × tp18
                else b) + b14t0 - tp14
      end;
      reg op:= 7addrop ∧ addr > 56 ∧ addr < 63
    end else
    bgn
      b14t9:= b14t0 : tp9; b8t0:= b14t0 - b14t9 × tp9;
      if b14t9 > 57 then

```

```

bgn co mr;
  if b14t9 = 63 then
    bgn
      if filled[63] then undef(25401);
      x := m0[63]; ccs := ccs + 2;
      if trace then report reading(63, x);
    end else
      x := if b14t9 < 61 then
        (if b14t9 = 58 then ft else
          if b14t9 = 59 then a else s) else
          if b14t9 = 61 then b else ot
      end
    end mr else
      bgn co mpq;
        if filled[63] then undef(25402);
        y := m0[63]; ccs := ccs + 2;
        if trace then report reading(63, y);
        if abs(y) > tp18m1 then y := y - y : tp18 × tp18;
        if y < 64 then undef(332);
        cg := 0; addr := y + b14t9; mem1; x := m;
      end mpq;
      addr := (if abs(x) > tp18m1 then x - x : tp18 × tp18
        else x) + b8t0 - tp8;
      if addr > 56 ∧ addr < 63 then undef(333); reg op := false;
      addrop := b20c19 = 4;
      mcaddr := b14t9 = 61 ∧ laddrop
    end dyn, :dyn;
    if addr < 0 then undef(26320) else
      if addr = 0 then addr := 0 else
        if addr > tp18m1 then undef(26321);
      mem op := 1 (addrop ∨ reg op)
    end calc addr;

proc condf;
bgn co conditiezetting op floating point getal
  in valh en valt;
  x := bit(14, valh);
  c :=
    if b18c17 = 1 then x else
      if b18c17 = 2 then
        (if valt ≠ 0 then 1 else
          if and(valh, tp14m1) = (if 1/valt > 0 then 0 else tp14m1)
            then 0 else 1) else
          if lt = x then 0 else 1;
        lt := x
      end
    end condf;

proc cond;
bgn co conditiezetting op 27-bits woord in val;
  x := if (if val = 0 then 1/val else val) > 0
    then 0 else 1;
  c := if b18c17 = 1 then x else
    if b18c17 = 2 then (if val = 0 then 0 else 1) else
      if lt = x then 0 else 1;

```

```

    lt:= x
  end cond;

co begin pariteitsberekening;

  integer array parlist[0:511];

  integer procedure parbit(bin); value bin; real bin;
  begin real x, y;
    if (if bin = 0 then 1/bin else bin) < 0 then
      bin:= bin + tp27m1;
      x:= bin : 512; y:= x : 512;
      x:= parlist[bin - x × 512] + parlist[x - y × 512] + parlist[y] + 1;
    rep:
      if x > 1 then begin x:= x - 2; goto rep end mod 2;
      parbit:= x
    end parbit;

  proc init parlist;
  begin int i, j;
    parlist[0]:= 0;
    for i:= 1, i + 1 while i < 256 do
      for j:= 0 step 1 until i - 1 do
        parlist[i+j]:= 1 - parlist[j]
      end init parlist;

co einde pariteitsberekening;

co begin memory simulation;

  proc bring in(region); real region;
  if m3p = 0 then
    bgn if m2p = 0 then
      bgn if m1p = 0 then
        bgn m1p:= region; m1l:= m cnt end else
        bgn m2p:= region; m2l:= m cnt end
      end else
        bgn m3p:= region; m3l:= m cnt end
    end unused regions else
  bgn

  proc transp(m, mp); int mp; int ar m;
  bgn
    to drum(m, mp × bufs);
    mp:= region; drum cnt:= drum cnt + 1;
    from drum(m, mp × bufs)
  end transp;

  if m1l > m2l then
    bgn
      if m2l > m3l then transp(m3, m3p)
        else transp(m2, m2p)
      end m2 of m3 else

```

```

    if m1l > m3l then transp(m3, m3p)
        else transp(m1, m1p)
end bring in;

proc stm;
if addr > 56 ∧ addr < 63 then undef(305) else stm1;

proc stm1;
if addr > max addr then undef(26322) else
bgn int region;
    if in monitor then goto skip adm;
    if prot then
        bgn
        if dgp[addr : 512] ≥ 0 then
            bgn
            if ¬ (addr < 15 ∨ addr = 63) then undef(27500)
            end
        end;
        if nowrite[addr] then
            bgn real m1, addr1, cg1;
            m1 := m; addr1 := addr; cg1 := cg; undef(5);
            m := m1; addr := addr1; cg := cg1
        end nowrite;
        m cnt := m cnt + 1;
skip adm:
    filled[addr] := cg = 0;
    region := addr : bufs; x := addr - region × bufs;
rep:
    if region = 0 then m0[x] := m else
    if region = m1p then bgn m1[x] := m; m1l := m cnt end else
    if region = m2p then bgn m2[x] := m; m2l := m cnt end else
    if region = m3p then bgn m3[x] := m; m3l := m cnt end else
    bgn bring in(region); goto rep end;
    if trace then report writing(addr, m)
end stm 1;

proc mem;
if addr > 56 ∧ addr < 63 then undef(304) else mem1;

proc mem1;
if addr > max addr then undef(26323) else
bgn int region; bool fa;
    if in monitor then goto skip adm1;
    if prot then
        bgn
        if dgp[addr : 512] > 0 then
            bgn
            if ¬ (addr < 15 ∨ addr = 63) then undef(27501)
            end
        end;
        end;
        ccs := ccs + 2; m cnt := mcnt + 1;
        fa := filled[addr];
        if cg < 2 ∧ ¬ fa then undef(25400);

```

```

skip adm1:
  region:= addr : bufs; x:= addr - region × bufs;
rep:
  if region = 0 then m:= m0[x] else
  if region = m1p then bgn m:= m1[x]; m1l:= m cnt end else
  if region = m2p then bgn m:= m2[x]; m2l:= m cnt end else
  if region = m3p then bgn m:= m3[x]; m3l:= m cnt end else
  bgn bring in (region); goto rep end;
  if trace then report reading(addr, m);
  if in monitor then goto skip adm2;
  if cg = 0 then else
  if cg = 1 then last par word:= m else
  bgn
    last par word:= if fa then m else -m;
    if cg = 3 then c:= if fa then 0 else 1
  end;
skip adm2:
end mem1;

proc init memory;
bgn
  y:= max addr : tp9;
  for x:= 0 step 1 until y do dgp[x]:= 2;
  co dp:= 'true', gp:= 'true';
  m cnt:= m1p:= m1l:= m2p:= m2l:= m3p:= m3l:= drum cnt:= 0;
  x:= 0;
rep:
  filled[x]:= nowrite[x]:= noexec[x]:= false;
  x:= x + 1; if x ≤ max addr then goto rep
end init memory;

int ar m0, m1, m2, m3[0:bufs - 1];
bool ar filled, nowrite, noexec[0 : max addr];
real m1p, m1l, m2p, m2l, m3p, m3l;
int ar dgp[0 : max addr : 512];
real m cnt, drum cnt;

co end memory simulation;

co simulatie charon;

real charon teller, ie0, ie1;
int af0, af1, if0, if1, lvif0, lvif1;
int afv0, afv1, ifv0, ifv1, lvifv0, lvifv1;
int ip lezer;

proc setq(l, q0, q1, v); val l; int l, q0, q1, v;
if l > 31 then q0:= set(v, 1 - 14, 1 - 14, q0) else
if l > 17 then q1:= set(v, 43 - 1, 43 - 1, q1) else
q0:= set(v, 17 - 1, 17 - 1, q0);

int proc readq(l, q0, q1); val l; int l, q0, q1;
readq:=

```



```

end lsnb;

proc ip;
begin real n, amount;

    real proc word;
    begin int n;

        real proc rehep1;
        bgn real n;
            n:= rehep1:= rehep;
            if n > 127 then undef(160)
            end rehep1;

        skip:
            n:= -(64 - rehep1); if n < 0 then goto skip;
            word:= ((n × 128 + rehep1) × 128 + rehep1) × 128 + rehep1
            end word;

    block:
        n:= word;
        amount:= n : tp18;
        addr:= n - amount × tp18; if addr = tp18m1 then addr:= -1;
        if amount ≠ 0 ∨ addr ≠ 0 then
            begin
                loop:
                    addr:= addr + 1; amount:= amount - 1;
                    m:= word; if m > tp26m1 then m:= m - tp27m1; stm;
                    goto if amount = 0 ∨ amount = -512 then block else loop
                end block
            end
        end ip;

    proc bi;
    bgn real a, s; int adr, cnt;

        proc rehep inf;
        bgn
            rep: s:= rehep;
            if s > 63 then
                bgn if s ≠ 127 then undef(172);
                for s:= rehep while s ≠ 127 do;
                goto rep
            end
        end rehep inf;

        proc re biword;
        bgn int n; real res; bool par;
            res:= s; par:= parlist[s] = 0;
            for n:= 1 step 1 until 4 do
                bgn
                    rehep inf; res:= res × 64 + s; par:= par = parlist[s] = 0
                end;
            end;

```

```

    if par then undef(173);
    a:= res div tp27; s:= -(a × tp27 - x);
    if s > tp26m1 then s:= s - tp27m1
end re biword;

```

```

read biblock:
  rehep inf; if s = 0 then goto read biblock;
  re biword; if a ≠ 3 then undef(174);
  adr:= and(s, tp18m1); cnt:= and(s div tp18, 511);
  if cnt = 0 then cnt:= 512;
  if adr > 0 ∨ cnt > 0 then
    bgn
      for cnt:= cnt step -1 until 1 do
        bgn rehep inf; re biword;
        if a ≠ 0 then undef(175);
        adr:= adr + 1;
        if adr < 24 ∨ adr > 56 ∧ adr < 63 then undef(176);
        addr:= adr; m:= s; stm1;
      end cnt;
      goto read biblock
    end biblock;
  end bi;

```

co einde simulatie charon;

co begin monitor procedures;

co ad hoc master- en trace-procedures, voor testdoeleinden;

```

proc master;
  bgn
    nlcr; printtext({simulatie beeindigd met foutnummer});
    absfixt(3, 0, error number);
    if ingreep type = 1 then printtext({in charon/sleutelingreep});
    if ingreep type = 2 then printtext({in foutingreep});
    if ingreep type = 3 then
      printtext({in foutingreep tijdens charon/sleutelingreep});
    print regs;
    print tlist;
    print dgp;
    print charon;
    dump(0, max addr);
    exit
  end master;

proc start monitor;
  bgn sva:= true; lsip; sva:= false;
  cg:= 0; addr:= 512; m:= 0; stm;
  bva
end start monitor;

proc terminate;
  bgn printtext({terminate}); master end;

```

```

proc report reading(addr, val); val addr, val; int addr, val;
bgn nlcr; printtext({read  });
  proct(addr, 9);
  proct(val, 9)
end;

```

```

proc report writing(addr, val); val addr, val; int addr, val;
bgn nlcr; printtext({write  });
  proct(addr, 9);
  proct(val, 9)
end;

```

```

proc report eoi;
bgn nlcr; printtext({e o inst}) end;

```

```

proc report eod(addr, instr); val addr, instr;
  int addr, instr;
bgn nlcr; printtext({e o do  });
  proct(addr, 9);
  proct(instr, 9)
end;

```

```

proc report interrupt(ir addr); val ir addr; int ir addr;
bgn nlcr; printtext({interr  });
  proct(ir addr, 9)
end;

```

```

proc report skip;
bgn nlcr; printtext({skip  }) end;

```

co einde ad hoc master- en trace-procedures;

```

int ir addr, error number;
bool sva, in monitor, trace, key;
real ccs, ccs1, ccs of, ccsmax, dcs, dcs max;
real jcnt, tent;
real ar tlist[0 : 31];

```

```

proc init monitor;
bgn
  ir addr:= 0; ccs:= ccs1:= ccs of:= dcs:= 0; error number:= 999;
  jcnt:= tent:= 0;
  ccsmax:= 1000000; dcsmax:= 100;
  trace:= sva:= key:= false; in monitor:= true;
  start monitor
end init monitor;

```

```

int proc undef(ern); val ern; int ern;
bgn undef:= 1;
  ir addr:= ern : 1000; error number:= ern - ir addr × 1000;
  if in monitor then terminate else
    if filled[ir addr] ∧ (ir addr= 0 ∨ ov=0) then
      bgn in monitor:= true;

```

```

    cg:= 0; master;
    in monitor:= false
  end else
  begin
    ingreep type:= ingreep type + 2;
    ov:= 0; goto ingreep
  end
end undef;

procedure proct(m,dig); val m,dig; int m, dig;
begin real n, x;
  space(1);
  if dig = 6 then x:= m / tp15 else
  bgn
    x:= (if 1/m < 0 then m + tp27m1 else m) / tp24;
    dig:= 9
  end;
rep:
  n:= entier(x); prsym(n);
  if dig= 7 then space(1);
  dig:= dig - 1;
  if dig= 0 then goto out;
  x:= (x-n) × 8;
  goto rep;
out:
  space(1)
end proct;

int proc reoct;
bgn int n;
skip:
  n:= resym; if n < 10 then undef(151);
  if n ≠ 120 then goto skip;
  x:= 0;
rep:
  n:= resym;
  if n = 93 then n:= resym;
  if n < 8 then
    bgn x:= x × 8 + n; goto rep end;
  if n ≠ 120 then undef(152);
  if x > tp27m1 then undef(153);
  if x > tp26m1 then x:= x - tp27m1;
  reoct:= x
end reoct;

int proc oct(x); val x; real x;
bgn int i, sgn, n; real res;
  if 1/x < 0 then bgn sgn:= -1; x:= -x end else sgn:= 1;
  x:= (x + .05)/109; res:= 0;
  for i:= 1 step 1 until 9 do
    bgn x:= x × 10; n:= entier(x); x:= x - n; res:= res × 8 + n
  end;
  oct:= (if res > tp26m1 then res - tp27m1 else res) × sgn

```

```

end oct;

proc tn(s, x); val x; real x; string s;
bgn int i, n;
  i:= 0;
  for n:= stringsymbol(i, s) while n  $\neq$  255,
    93 while i < 7 do
    bgn i:= i + 1; prsym(n) end;
    proct(x, 9); space(5)
  end tn;

procedure dump(from, to); val from, to; integer from, to;
begin real i, j, x;
  if from < 0  $\vee$  to > max addr then undef(161);
  if line number  $\neq$  1  $\vee$  print pos  $\neq$  0 then new page;
  printtext( $\{$ geheugendump van $\}$ ); proct(from, 6);
  printtext( $\{$ tot $\}$ ); proct(to,6); nlcr;
  addr:= from; goto heading;
rep:
  addr:= addr + 1;
repa:
  if addr > to then goto out;
  if 7 filled[addr] then goto rep;
  printtext( $\{$ loc $\}$ ); addr:= addr : 8  $\times$  8;
  proct(addr,6); prsym(65); proct(addr + 7,6);
  i:= 0;
repi:
  j:= 0; space(8);
repj:
  if filled[addr] then
    bgn mem1; proct(m, 9) end else space(12);
    addr:= addr + 1;
    if addr > to then goto out;
    j:= j + 1;
    if j < 4 then goto repj;
    i:= i + 1;
    if i < 2 then goto repi;
    nlcr;
    if line number = 1 then
heading:
  begin space(25);
  for i:= 0 step 1 until 7 do
    begin if i = 4 then space(8);
    printtext( $\{$  ..... $\}$ ); prsym(i)
    end;
    nlcr; nlcr;
  end;
  goto repa;
out:
  nlcr;
  for i:= 1 step 1 until 1 $\frac{1}{4}$  do prsym(66);
  nlcr
end dump;

```

```

proc print tlist;
  bgn int max, tent1, num, x;
  nclr; printtext({vorige sprongadressen}); nclr;
  max:= if jcnt > 32 then 32 else jcnt; tent1:= tent;
  for num:= 1 step 1 until max do
    bgn
      space(2); fixt(2, 0, -num);
      x:= tlist[tent1];
      if 1/x < 0 then proct(and(x, tp26m1), 9) else
        bgn space(4); proct(x, 6) end;
      tent1:= tent1 - 1;
      if tent1 = -1 then tent1:= 31
    end;
  nclr
end print tlist;

proc print dgp;
  bgn int i, j;
  nclr; printtext({protectiebits});
  for i:= 0 step 1 until aantal kasten - 1 do
    bgn nclr; printtext({kast}); absfixt(2, 0, i);
    printtext({ dp });
    for j:= 0 step 1 until 3 do
      prsym((even(dgp[32 × i + 8 × j]) + 1) / 2);
      printtext({ gp });
      for j:= 0 step 1 until 31 do
        prsym((dgp[32 × i + j] + 1) : 2)
      end;
    end;
  nclr
end print dgp;

co dp 0 1 0 1
   gp 0 0 1 1
   dgp -1 0 1 2;

proc print regs;
  bgn nclr; printtext({registers}); nclr;
  tn({tlink}, tlink ); tn({fh}, fh ); tn({ft}, ft );
  tn({a}, a ); tn({s}, s ); tn({b}, b );
  tn({or}, or ); tn({addr}, addr ); tn({m}, m );
  tn({ir addr}, ir addr);
  nclr; printtext({jcnt (dec)}); absfixt(10, 0, jcnt);
  space(10); printtext({ccs (dec)}); absfixt(10, 0, ccs );
  printtext({+}); absfixt(6, 0, ccs of);
  space(10); printtext({drum cnt (dec)}); absfixt(10, 0, drum cnt);
  nclr
end print regs;

proc print charon;
  bgn int i;
  nclr; nclr; printtext({charon}); nclr;
  printtext({apparaat af if lvif});

```

```

for i:= 0 step 1 until 39 do
  if read q(i, ifv0, ifv1) + read q(i, lvifv0, lvifv1) = 2 then
    bgn
      nlcr; absfixt(7, 0, i);
      absfixt(4, 0, readq(i, af0, af1));
      absfixt(4, 0, readq(i, if0, if1));
      absfixt(4, 0, readq(i, lvif0, lvif1))
    end;
  nlcr;
  for i:= 1 step 1 until 26 do prsym(66);
  nlcr
end print charon;

co einde monitor procedures;

comment herkennen van instructies;

real x, y, z, u, v, w, tp26, tp27, tp27m1;
real i, l, n;
real array tp[0:27];

int tp1, tp2, tp3, tp4, tp5, tp6, tp7, tp8, tp9, tp10,
      tp11, tp12, tp13, tp14, tp15, tp16, tp17, tp18, tp19, tp20,
      tp21, tp22, tp23, tp24, tp25;
int tp14m1, tp18m3, tp18m1, tp26m1;

comment hulp-velden uit or: ;
real b14t0, b16c15, b18c17, b20c19, b20,
      b14t12, b11t0, b11t5, b11t9, b8t6, b5t0,
      b4t0, b24, b21, b14t9, b8t0;

comment switches voor het herkennen van instructies: ;

switch sb26t22:=
  aplus,
  aeq,
  if b20c19 = 1 then aplusdyn else plusa,
  if b20c19 = 1 then aeqdyn else
    if b16c15 = 1 then aplusa else eqa,
  splus,
  seq,
  if b20c19 = 1 then splusdyn else pluss,
  if b20c19 = 1 then seqdyn else
    if b16c15 = 1 then spluss else eqs,
  mulas,
  muls,
  axor,
  aand,
  divas,
  diva,
  sxor,
  sand,
  bplus,

```

```

beq,
if b20c19 = 1 then bplusdyn else plusb,
if b20c19 = 1 then beqdyn else
  if b16c15 = 1 then bplusb else eqb,
if b18c17 > 1 then dsw[undef(202)] else jump,
if b18c17 > 1 then dsw[undef(204)] else
if b21 = 0 then goto else
  if b20c19 = 3 then gotodyn else dsw[undef(203)],
repn,
if b18c17 = 0 then subn else
if b18c17 = 2 then subn else
if b21 = 0 then subc else
  if b20c19 = 1 then subcdyn else do,
if b16c15 = 1 then
  (if b20c19 = 1 then dsw[undef(228)] else gplus)
  else fplus,
if b16c15 = 1 then
  (if b20c19 = 1 then dsw[undef(229)] else geq )
  else feq,
if b21 = 0 then
  (if b20c19 = 1 V b20c19 = 3 V b18c17 ≠ 0 then
    dsw[undef(232)] else operate)
  else dsw[undef(200)],
tabel6,
if b16c15 = 1 then
  (if b20c19 = 1 then dsw[undef(230)] else gtd )
  else ftd,
if b16c15 = 1 then
  (if b20c19 = 1 then dsw[undef(231)] else eqg )
  else if b20c19 = 1 then feqdyn else eqf,
dsw[undef(201)],
tabel7;

switch sb24c21c6c5:=
  lca, lua, lcas, luas,
  rca, rua, rcas, ruas,
  lcs, lus, lcsa, lusa,
  rcs, rus, rcsa, rusa;

comment hulp-procedures: ;

proc itp(x); real x;
bgn x:= tp[i]; i:= i + 1 end itp;

comment initialisatie: ;

tp[0]:= 1;
for i:= 1 step 1 until 27 do tp[i]:= tp[i - 1] × 2;
i:= 1;
itp(tp1); itp(tp2); itp(tp3); itp(tp4); itp(tp5); itp(tp6);
itp(tp7); itp(tp8); itp(tp9); itp(tp10); itp(tp11); itp(tp12);
itp(tp13); itp(tp14); itp(tp15); itp(tp16); itp(tp17); itp(tp18);
itp(tp19); itp(tp20); itp(tp21); itp(tp22); itp(tp23); itp(tp24);

```

```

itp(tp25); itp(tp26); itp(tp27);
tp14m1:= tp14 - 1; tp18m3:= tp18 - 3;
tp18m1:= tp18 - 1; tp26m1:= tp26 - 1; tp27m1:= tp27 - 1;

```

```

set random(date/622598 + time/7200);
init parlist;
init x8;
init memory;
init charon;
init monitor;

```

bval:

```

in monitor:= false;

```

basis cyclus 1:

```

if ot > tp18m3 then
  bgn co dynamische stop;
  if dyst stat < 2 then
    bgn dyst stat:= dyst stat + 1; goto skip end else
    if charon teller > 0 then
      bgn
        ccs:= ccs + charon teller; dyst stat:= 0; goto skip
      end else undef(1)
    end else dyst stat:= 0;

```

```

cg:= 0; addr:= ot; mem; or:= m;
if noexec[ot] then undef(4); ot:= ot + 1;
if ot > max addr then undef(26358);

```

basis cyclus 2:

```

if ccs > ccsmx then undef(2);
flag:= false;
y:= if 1/or < 0 then or + tp27m1 else or;
x:= y : tp15; b14t0 := y - x × tp15; y:= x;
x:= y : tp2; b16c15:= y - x × tp2; y:= x;
x:= y : tp2; b18c17:= y - x × tp2; y:= x;
x:= y : tp2; b20c19:= y - x × tp2; y:= x;
x:= y : tp1; b21 := y - x × tp1;
goto sb26t22[x + 1];

```

tabel6:

```

b24:= 0; goto tabel;

```

tabel7:

```

b24:= 1;

```

tabel:

```

b20: b20c19 : 2;
if b20 + b20 ≠ b20c19 then
  bgn
    if bitstring(26, 17, or) = 1004 ∧ b14t0 = 31488 then
      goto memprot;
      undef(206)
    end b19 = 1;
    b14t12:= b14t0 : tp12; b11t0:= b14t0 - b14t12 × tp12;

```

```

if b14t12 = 0 then
  bgn co shift, nora, nors, noras, rgt, ten;
  b11t5:= b11t0 : tp5; b4t0:= b11t0 - b11t5 × tp5;
  if b11t5 < 4 then goto shift else
  if b11t5 = 5 then
    bgn
    if b21 + b20 + b4t0 = 0 then
      goto if b24 = 0 then nora else nors
    end else
    if b11t5 = 7 then
      bgn
      if b24 + b21 + b20 + b4t0 = 0 then goto noras
      end else
      if b11t5 > 7 ∧ b11t5 < 11 then
        bgn
        if b24 + b20 = 0 ∧ b4t0 < 3 then goto rgt
        end else
        if b11t5 = 32 then
          bgn
          if b24 = 1 ∧ b21 + b20 = 0 ∧ b4t0 < 2 then goto ten
          end
        end else
        if b14t12 = 6 then
          bgn co clp, int, maak iv, maak ov, itvon;
          b11t9:= b11t0 : tp9; b8t0:= b11t0 - b11t9 × tp9;
          if b21 + b20 + b18c17 + b8t0 = 0 then
            bgn
            if b11t9 = 0 then goto maak iv;
            if b11t9 = 1 then goto maak ov;
            if b11t9 = 2 ∧ b24 = 1 then goto itvon;
            if b11t9 = 3 ∧ b24 = 0 then goto clp;
            if b11t9 = 4 ∧ b24 = 0 then goto int
            end
          end else
          if b14t12 = 7 then
            bgn co maak q, lees q;
            b11t9:= b11t0 : tp9; b8t0:= b11t0 - b11t9 × tp9;
            b 8t6:= b 8t0 : tp6; b5t0:= b 8t0 - b 8t6 × tp6;
            if b11t9 < 4 then
              bgn
              if b21 + b18c17 + b8t6 = 0 then goto maak q
              end else
              if b21 = 0 ∧ b8t6 < 2 ∧ b5t0 < 2 then goto lees q
            end tabel;
          under(205);

```

```

aplusdyn:  b20c19:= 4;   goto aplus;
aeqdyn:    b20c19:= 4;   goto aeq;
splusdyn:  b20c19:= 4;   goto splus;
seqdyn:    b20c19:= 4;   goto seq;
bplusdyn:  b20c19:= 4;   goto bplus;
beqdyn:    b20c19:= 4;   goto beq;
gotodyn:

```

```

b20c19:= 4; b21:= 0; co abnormale betekenis van b21; goto goto;
subcdyn:   b20c19:= 4; goto subc;
feqdyn:    b20c19:= 4; goto feq;

```

```

aeq:
  if b16c15 > 1 then
    bgn if b16c15 ≠ c+2 then goto skip end;
    calc addr;
    if addr op then val:= addr else
      bgn cg:= itv + itv + 1; memreg; val:= m end;
      if mc addr then bdown1;
      if b21 = 1 then val:= -val;
      if b18c17 > 0 then cond;
      if b16c15 ≠ 1 then a:= val;
      goto time0;

```

```

seq:
  if b16c15 > 1 then
    bgn if b16c15 ≠ c+2 then goto skip end;
    calc addr;
    if addr op then val:= addr else
      bgn cg:= itv + itv + 1; memreg; val:= m end;
      if mc addr then bdown1;
      if b21 = 1 then val:= -val;
      if b18c17 > 0 then cond;
      if b16c15 ≠ 1 then s:= val;
      goto time0;

```

```

beq:
  if b16c15 > 1 then
    bgn if b16c15 ≠ c+2 then goto skip end;
    calc addr;
    if addr op then val:= addr else
      bgn cg:= itv + itv + 1; memreg; val:= m end;
      if mc addr then bdown1;
      if b21 = 1 then val:= -val;
      if b18c17 > 0 then cond;
      if b16c15 ≠ 1 then b:= val;
      goto time0;

```

```

aplus:
  if b16c15 > 1 then
    bgn if b16c15 ≠ c+2 then goto skip end;
    calc addr;
    if addr op then val:= addr else
      bgn cg:= itv + 1; memreg; val:= m end;
      if mc addr then bdown1;
      val:= if b21 = 0 then a + val else a - val;
      if abs(val) > tp26m1 then
        bgn of:= 1;
        val:= if val > 0 then val - tp27m1 else val + tp27m1
      end;

```

```

if b13c17 > 0 then cond;
if b16c15 ≠ 1 then a:= val;
goto time0;

```

splus:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if addr op then val:= addr else
  bgn cg:= itv + 1; memreg; val:= m end;
  if mc addr then bdown1;
  val:= if b21 = 0 then s + val else s - val;
  if abs(val) > tp26m1 then
  bgn of:= 1;
    val:= if val > 0 then val - tp27m1 else val + tp27m1
  end;
  if b18c17 > 0 then cond;
  if b16c15 ≠ 1 then s:= val;
  goto time0;

```

bplus:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if addr op then val:= addr else
  bgn cg:= itv + 1; memreg; val:= m end;
  if mc addr then bdown1;
  val:= if b21 = 0 then b + val else b - val;
  if abs(val) > tp26m1 then
  bgn of:= 1;
    val:= if val > 0 then val - tp27m1 else val + tp27m1
  end;
  if b18c17 > 0 then cond;
  if b16c15 ≠ 1 then b:= val;
  goto time0;

```

eqa:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if regop then undef(306);
  val:= if b21 = 0 then a else -a;
  if b18c17 > 0 then cond;
  m:= val; cg:= itv; stm1;
  if mc addr then b up1;
  goto time2;

```

eqs:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if regop then undef(307);
  val:= if b21 = 0 then s else -s;

```

```

if b13c17 > 0 then cond;
m:= val; cg:= itv; stm1;
if mc addr then b up1;
goto time2;

```

eqb:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if regop then undef(308);
  val:= if b21 = 0 then b else -b;
  if b18c17 > 0 then cond;
  m:= val; cg:= itv; stm1;
  if mc addr then b up1;
  goto time2;

```

aplusa:

```

flag:= true;

```

plusa:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if regop then undef(309);
  cg:= 1; mem1; val:= m;
  val:= if b21 = 0 then val + a else val - a;
  if abs(val) > tp26m1 then
    bgn of:= 1;
    val:= if val > 0 then val - tp27m1 else val + tp27m1
  end;
  if b18c17 > 0 then cond;
  if flag ∨ b16c15 ≠ 1 then
    bgn m:= val; cg:= itv; stm1 end;
  if flag then a:= m;
  if mc addr then b up1;
  goto time3;

```

spluss:

```

flag:= true;

```

pluss:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if regop then undef(310);
  cg:= 1; mem1; val:= m;
  val:= if b21 = 0 then val + s else val - s;
  if abs(val) > tp26m1 then
    bgn of:= 1;
    val:= if val > 0 then val - tp27m1 else val + tp27m1
  end;
  if b18c17 > 0 then cond;
  if flag ∨ b16c15 ≠ 1 then
    bgn m:= val; cg:= itv; stm1 end;
  if flag then s:= m;

```

```

if mc addr then b up1;
goto time3;

```

bplusb:

```

flag:= true;

```

plusb:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
calc addr;
if regop then undef(311);
cg:= 1; mem1; val:= m;
val:= if b21 = 0 then val + b else val - b;
if abs(val) > tp26m1 then
bgn of:= 1;
val:= if val > 0 then val - tp27m1 else val + tp27m1
end;
if b18c17 > 0 then cond;
if flag ∨ b16c15 ≠ 1 then
bgn m:= val; cg:= itv; stm1 end;
if flag then b:= m;
if mc addr then b up1;
goto time3;

```

axor:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
calc addr;
if addr op then val:= addr else
bgn cg:= 1; memreg; val:= m end;
if mc addr then bdown1;
val:= xor(a, if b21 = 0 then val else -val);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then a:= val;
goto time0;

```

sxor:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
calc addr;
if addr op then val:= addr else
bgn cg:= 1; memreg; val:= m end;
if mc addr then bdown1;
val:= xor(s, if b21 = 0 then val else -val);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then s:= val;
goto time0;

```

aand:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
calc addr;
if addr op then val:= addr else
bgn cg:= 1; memreg; val:= m end;

```

```

if mc addr then bdown1;
val:= and(a, if b21 = 0 then val else -val);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then a:= val;
goto time0;

```

sand:

```

if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if addr op then val:= addr else
  bgn cg:= 1; memreg; val:= m end;
  if mc addr then bdown1;
  val:= and(s, if b21 = 0 then val else -val);
  if b18c17 > 0 then cond;
  if b16c15 ≠ 1 then s:= val;
  goto time0;

```

mul:

```

if b16c15 = 1 then undef(223);
flag:= true;

```

mulas:

```

if b16c15 = 1 then undef(222);
if b16c15 > 1 then
  bgn if b16c15 ≠ c+2 then goto skip end;
  calc addr;
  if addr op then val:= addr else
  bgn cg:= 1; memreg; val:= m end;
  if mc addr then bdown1;
  if b21 = 1 then val:= - val;
  hulp1:= val; co voor ccs-metingen;
  if flag then a:= 0;
  y:= valxs + a;
  if y=0 then a:= s:= y else
  if abs(y)<tp26 then
  begin s:= y; a:= if y>0 then 0 else -0 end else
  begin y:= val : tp13; z:= y × tp13;
  v:= s : tp13;
  z:= (s - v×tp13) × z + (val - z) × s + a;
  w:= z : tp26;
  s:= z - w×tp26;
  a:= y × v + w;
  if s=0 then s:= sign(a)×0
end mulas;
if b18c17 > 0 then bgn val:= a; cond end;

```

```

x:= abs(hulp1)/tp26; y:= 26;

```

rep ccs mulas:

```

x:= x + x; y:= y - 1; ccs:= ccs + 1;
if x>1 then x:= x - 1 else
begin x:= x + x; y:= y - 1;
  if x > 1 then x:= x - 1
end;

```

```

if x = 0 then ccs:= ccs + (y+1) : 2 else goto rep ccs mulas;
goto time2;

```

diva:

```

if b16c15 = 1 then undef(224);
flag:= true;

```

divas:

```

if b16c15 = 1 then undef(225);
if b16c15 > 1 then
  bgn if b16c15  $\neq$  c+2 then goto skip end;
  calc addr;
  if addr op then val:= addr else
  bgn cg:= 1; memreg; val:= m end;
  if mc addr then bdown1;
  if b21 = 1 then val:= - val;
  if abs(val)  $\leq$  abs(a) then undef(103);
  if flag then
  s:= if a = 0 then a else if a > 0 then 0 else -0 else
  if  $1/a > 0 = 1/s > 0$  then undef(102);
  hulp1:= a;
  if abs(a)<tp13 then
  bgn
    y:= a  $\times$  tp26 + s;
    s:= y : val; a:= y - s  $\times$  val
  end else
  bgn
    u:= s : tp13; y:= a  $\times$  tp13 + u; v:= y : val;
    y:= (y - v  $\times$  val - u)  $\times$  tp13 + s; w:= y : val;
    a:= y - w  $\times$  val; s:= v  $\times$  tp13 + w
  end diva; divas;
  if a=0 then
  a:= if hulp1 = 0 then hulp1 else
    if hulp1 > 0 then 0 else -0;
  if b16c17 > 0 then bgn val:= a; cond end;
  ccs:= ccs + 28;
  goto time0;

```

ten:

```

if b16c15 = 1 then undef(211);
if b16c15 > 1 then
  bgn if b16c15  $\neq$  c+2 then goto skip end;
  x:= s  $\times$  10; val:= s:= tail of(x);
  if b4t0 = 0 then a:= head of(x);
  if b18c17 > 0 then cond;
  goto time1;

```

noras:

```

if b16c15 = 1 then undef(219);
flag:= true;

```

shift:

```

if b16c15 = 1 then undef(218);
if b16c15 > 1 then
  bgn if b16c15  $\neq$  c+2 then goto skip end;

```

```

if b20 = 0 then l := b4t0 else
  bgn l := b4t0 + b;
  if l < 0 ∨ l > 31 then undef(140)
  end;
  if b11t5 > 1 then
    bgn co ook voor noras;
    nega := 1/a < 0;
    negs := 1/s < 0;
  end;
  if flag then
    bgn co noras;
    b := if a = 0 then norm shift
          (if nega ∧ negs then s - tp26m1 else
           if negs ∧ nega then s + tp26m1 else s, x) + 26 else
          norm shift(a, x);
    l := n := b; goto lcas
  end;
  n := 1;
  goto sb24c21c6c5[(b24 + b24 + b21) × 4 + b11t5 + 1];

lca:
  a := circ shift(a, n);
  if b18c17 > 0 then bgn val := a; cond end;
  goto timel2;

lcs:
  s := circ shift(s, n);
  if b18c17 > 0 then bgn val := s; cond end;
  goto timel2;

lcas:
  if n > 26 then begin n := 53 - n; goto rcas2 end;
lcsa2:
  if n ≠ 0 then
    bgn
      if nega then a := a + tp27m1;
      if negs then s := s + tp26m1;
      hulp1 := tp[n]; hulp2 := tp26/hulp1; hulp3 := hulp2 + hulp2;
      ah := a : hulp3;
      sh := s : hulp2;
      a := -((ah × hulp3 - a) × hulp1 - sh);
      s := -((sh × hulp2 - s) × hulp1 - ah);
      if a > tp26 then a := a - tp27m1;
      if negs then s := s - tp26m1;
    end;
    if b18c17 > 0 then bgn val := a; cond end;
    goto timel2;

lcsa:
  if n > 27 then bgn n := 54 - n; goto rcsa2 end;
lcsa2:
  if n ≠ 0 then
    bgn

```

```

    if negs then s:= s + tp27m1;
    if nega then a:= a + tp27m1;
    hulp1:= tp[n]; hulp3:= tp27/hulp1;
    sh:= s : hulp3;
    ah:= a : hulp3;
    s:= -((sh × hulp3 - s) × hulp1 - ah);
    a:= -((ah × hulp3 - a) × hulp1 - sh);
    if s > tp26 then s:= s - tp27m1;
    if a > tp26 then a:= a - tp27m1;
    end;
    if b18c17 > 0 then bgn val:= s; cond end;
    goto timel2;

rca:
    a:= circ shift(a, -n);
    if b18c17 > 0 then bgn val:= a; cond end;
    goto timel2;

rcs:
    s:= circ shift(s, -n);
    if b18c17 > 0 then bgn val:= s; cond end;
    goto timel2;

rcas:
    if n>26 then begin n:= 53-n; goto lcas2 end;
rcas2:
    if n≠0 then
        bgn
        if nega then a:= a + tp27m1;
        if negs then s:= s + tp26m1;
        hulp1:= tp[n]; hulp2:= tp26/hulp1; hulp3:= hulp2 + hulp2;
        ah:= a : hulp1;
        sh:= s : hulp1;
        at:= ah × hulp1 - a;
        a:= -((sh × hulp1 - s) × hulp3 - ah);
        s:= -(at × hulp2 - sh);
        if a > tp26 then a:= a - tp27m1;
        if negs then s:= s - tp26m1;
        end;
        if b18c17 > 0 then bgn val:= a; cond end;
        goto timel2;

rcsa:
    if n>27 then bgn n:= 54 - n; goto lcsa2 end;
rcsa2:
    if n≠0 then
        bgn
        if negs then s:= s + tp27m1;
        if nega then a:= a + tp27m1;
        hulp1:= tp[n]; hulp3:= tp27/hulp1;
        sh:= s : hulp1;
        ah:= a : hulp1;
        st:= sh × hulp1 - s;

```

```

s:= -((ah × hulp1 - a) × hulp3 - sh);
a:= -(st × hulp3 - ah);
if s > tp26 then s:= s - tp27m1;
if a > tp26 then a:= a - tp27m1;
end;
if b18c17 > 0 then bgn val:= s; cond end;
goto timel2;

```

lua:

```

if n > 26 then a:= sign(a) × 0 else
if a ≠ 0 ∧ n ≠ 0 then
bgn
hulp2:= tp[26 - n]; at:= a - a : hulp2 × hulp2;
a:= if at = 0 then sign(a) × 0 else tp26 / hulp2 × at
end;
if b18c17 > 0 then bgn val:= a; cond end;
goto timel;

```

lus:

```

if n > 26 then s:= sign(s) × 0 else
if s ≠ 0 ∧ n ≠ 0 then
bgn
hulp2:= tp[26 - n]; st:= s - s : hulp2 × hulp2;
s:= if st = 0 then sign(s) × 0 else tp26 / hulp2 × st
end;
if b18c17 > 0 then bgn val:= s; cond end;
goto timel;

```

luas:

```

if n > 26 then
bgn
if negs then s:= s + tp26m1;
hulp1:= tp[n - 26]; hulp2:= tp26/hulp1;
a:= -(s : hulp2 × hulp2 - s) × hulp1;
s:= if negs then -tp26m1 else 0;
if nega then
bgn
a:= a - tp26m1 + hulp1 - 1;
s:= s + tp26m1
end;
end else
if n ≠ 0 then
bgn
if nega then a:= a + tp26m1;
if negs then s:= s + tp26m1;
hulp1:= tp[n]; hulp2:= tp26/hulp1;
sh:= s : hulp2;
a:= -((a : hulp2 × hulp2 - a) × hulp1 - sh);
s:= -(sh × hulp2 - s) × hulp1;
if negs then s:= s - tp26m1;
if nega then
bgn a:= a - tp26m1; s:= s + hulp1 - 1 end;
end;

```

```

if b18c17 > 0 then bgn val:= a; cond end;
goto timel2;

```

lusa:

```

if n>26 then
  bgn
    if nega then a:= a + tp27m1;
    hulp1:= tp[n - 27]; hulp2:= tp26/hulp1;
    s:= -(a : hulp2 × hulp2 - a) × hulp1;
    a:= 0;
    if negs then
      bgn s:= s - tp26m1 + hulp1 - 1; a:= -0 end;
    end else
      if n ≠ 0 then
        bgn
          if nega then a:= a + tp27m1;
          if negs then s:= s + tp26m1;
          hulp1:= tp[n]; hulp2:= tp26/hulp1; hulp3:= hulp2 + hulp2;
          ah:= a : hulp3;
          a:= -(ah × hulp3 - a) × hulp1;
          s:= -((s : hulp2 × hulp2 - s) × hulp1 - ah);
          if a > tp26 then a:= a - tp27m1;
          if negs then
            bgn s:= s - tp26m1; a:= a + hulp1 - 1 end;
          end;
        if b18c17 > 0 then bgn val:= s; cond end;
        goto timel2;

```

rua:

```

a:= if n > 26 then sign(a) × 0 else a : tp[n];
if b15c17 > 0 then bgn val:= a; cond end;
goto timel;

```

rus:

```

s:= if n > 26 then sign(s) × 0 else s : tp[n];
if b18c17 > 0 then bgn val:= s; cond end;
goto timel;

```

ruas:

```

if n>26 then
  begin s:= a:tp[n-26]; a:= if nega then -0 else 0;
    if nega ∧ ¬negs then s:= s + tp26m1 else
      if negs ∧ ¬negs then s:= s - tp26m1
    end else if n≠0 then
      bgn
        hulp1:= tp[n]; hulp2:= tp26/hulp1;
        ah:= a : hulp1;
        if nega then a:= a + tp26m1;
        if negs then s:= s + tp26m1;
        s:= -((a : hulp1 × hulp1 - a) × hulp2 - s : hulp1);
        a:= ah;
        if negs then s:= s - tp26m1;
      end;

```

```

if b18c17 > 0 then bgn val:= a; cond end;
goto timel2;

```

rusa:

```

if n>26 then
begin a:= s:tp[n-27]; s:= if negs then -0 else 0 end
else if n ≠ 0 then
bgn
  hulp1:= tp[n]; hulp3:= tp27/hulp1;
  sh:= s : hulp1;
  if negs then s:= s + tp26m1;
  if nega then a:= a + tp27m1;
  a:= -((s : hulp1 × hulp1 - s) × hulp3 - a : hulp1);
  s:= sh;
  if a ≥ tp26 then a:= a - tp27m1
end;
if b18c17 > 0 then bgn val:= s; cond end;
goto timel2;

```

nora:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
b:= l:= norm shift(a, val);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then a:= val; goto timel;

```

nors:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
b:= l:= norm shift(s, val);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then s:= val; goto timel;

```

feq:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
calc addr;
if addrop then bgn valh:= 0; valt:= addr end else
if addr = 57 then
bgn valh:= fh; valt:= ft end else
if addr = 59 then
bgn valh:= a; valt:= s end else
bgn cg:= 0; mem; valh:= m; addr:= addr + 1; mem; valt:= m end;
if mc addr then bdown2;
if signinc then of:= 1;
if b21 = 1 then bgn valh:= -valh; valt:= -valt end;
if b18c17 > 0 then condf;
if abs(valh) > tp14m1 then nint:= 1;
fh:= valh; ft:= valt;
goto time0;

```

eqf:

```

if b16c15 > 1 then

```

```

bgn if b16c15  $\neq$  c+2 then goto skip end;
calc addr;
if b21 = 0 then bgn valh:= fh; valt:= ft end else
bgn valh:= -fh; valt:= -ft end;
if b18c17 > 0 then condf;
if abs(valh) > tp14m1 then nint:= 1;
cg:= 0; m:= valh; stm; addr:= addr + 1; m:= valt; stm;
if mc addr then b up2;
goto time0;

```

geq:

```

calc addr;
if addr = 57 then undef(341);
cg:= 0; memreg; valt:= m; valh:= sign(m)  $\times$  0;
if mc addr then bdown1;
if b21 = 1 then bgn valh:= -valh; valt:= -valt end;
if b18c17 > 0 then condf;
fh:= valh; ft:= valt;
goto time0;

```

eqg:

```

calc addr;
if regop then undef(312);
if b21 = 0 then bgn valh:= fh; valt:= ft end else
bgn valh:= -fh; valt:= -ft end;
if b18c17 > 0 then condf;
if  $7(\text{valh} = 0 \wedge \text{sign}(1/\text{valh}) = \text{sign}(1/\text{valt}))$  then nint:= 1;
cg:= 0; m:= valt; stm1;
if mc addr then b up1;
goto time0;

```

ftd:

flag:= true;

fplus:

```

if b16c15 > 1 then
bgn if b16c15  $\neq$  c+2 then goto skip end;
calc addr;
if addrop then bgn valh:= 0; valt:= addr end else
if addr = 57 then
bgn valh:= fh; valt:= ft end else
if addr = 59 then
bgn valh:= a; valt:= s end else
bgn cg:= 0; mem; valh:= m; addr:= addr + 1; mem; valt:= m end;
if mc addr then bdown2;
if signinc then of:= 1;
x:= compf(fh, ft); y:= compf(valh, valt);
if flag then
bgn
if b21 = 0 then
bgn x:= x  $\times$  y; ccs:= ccs + 4; ccsof:= ccsof + 45 end else
bgn
if x = 0  $\wedge$  y  $\neq$  0 then undef(120);
x:= x / y; ccs:= ccs + 44; ccsof:= ccsof + 5

```

```

    end
  end else
    bgn
      x:= if b21 =0 then x + y else x - y;
      if b20c19 = 1 then ccs:= ccs + 1;
      ccsof:= ccsof + 9
    end;
    valh:= head of(x); valt:= tail of(x);
    if b18c17 > 0 then condf;
    if abs(valh) > tp14m1 then nint:= 1;
    fh:= valh; ft:= valt;
    goto time0;

gtd:
  flag:= true;
gplus:
  calc addr;
  if addr = 57 then undef(342);
  cg:= 0; memreg; valt:= m; valh:= sign(m) × 0;
  x:= compf(fh, ft); y:= compose(valh, valt);
  if flag then
    bgn
      if b21 = 0 then
        bgn x:= x × y; ccs:= ccs + 5; ccsof:= ccsof + 43 end else
        bgn
          if x = 0 ∧ y =0 then undef(121);
          x:= x / y; ccs:= ccs + 45; ccsof:= ccsof + 5
        end
      end else
        bgn
          x:= if b21 =0 then x + y else x - y;
          ccsof:= ccsof + 9
        end;
        valh:= head of(x); valt:= tail of(x);
        if b18c17 > 0 then condf;
        if abs(valh) > tp14m1 then nint:= 1;
        fh:= valh; ft:= valt;
        goto time0;

jump:
  flag:= true;
goto:
  if b16c15 > 1 then
    bgn if b16c15 ≠ c+2 then goto skip end else
    if b16c15 = 1 then
      bgn if of = 0 then goto skip end;
      calc addr;
      if mc addr then bdown1;
      if b16c15 = 1 then of:= 0;
      if addrop then val:= addr else
      if addr = 62 then undef(343) else
      bgn cg:= 0; memreg; val:= m end;
      if b21 = 1 then val:= -val;

```

```

hulp1:= val;
if abs(val) > tp18m1 then val:= star(val);
if flag then val:= -(-ot - val);
if 1/val < 0 then undef(26350) else
if val > tp18m1 then val:= star(val) else
if val > tp18m3 then else
if val > max addr then undef(26351);
jcnt:= jcnt + 1; tcnt:= tcnt + 1;
if tcnt = 32 then tcnt:= 0;
tlist[tcnt]:= ot;
ot:=val;
if b18c17 = 0 then goto time0;
y:= (if 1/hulp1 < 0 then hulp1 + tp26m1 else hulp1) : tp18;
x:= y : 2; c:= y - x - x;
y:= x : 2; iv:= x - y - y;
x:= y : 2; lt:= y - x - x;
y:= x : 2; of:= x - y - y;
x:= y : 2; last par word:= -(y - x - x - 1);
y:= x : 2; nint:= x - y - y;
x:= y : 2; ov:= y - x - x;
if bt = 0 V x ≠ 0 then bgn iv:= ov:= 1; bt:= 0 end;
goto time0;

```

repn:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end else
if b16c15 = 1 then
bgn if of = 0 then goto skip end;
if b16c15 = 1 then of:= 0;
hulp1:= b21 + b21 + b21 + b21 + b20c19;
if 7 filled[hulp1] then undef(25410);
x:= m0[hulp1] -1;
if trace then report reading(hulp1, x + 1);
if x < -tp26m1 then x:= tp26m1;
m0[hulp1]:= x;
if trace then report writing(hulp1, x);
if if b18c17 = 0 then true else
if b18c17 = 1 then x > 1 else
if b18c17 = 2 then x = 0 else x > 0 then
bgn
if b14t0 > max addr then undef(26354);
jcnt:= jcnt + 1; tcnt:= tcnt + 1;
if tcnt = 32 then tcnt:= 0;
tlist[tcnt]:= ot;
ot:= b14t0
end;
goto time0;

```

subn:

```

if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end else
if b16c15 = 1 then
bgn if of = 0 then goto skip end;

```

```

if b16c15 = 1 then of:= 0;
hulp2:= if ingreep type = 2 then tlink + tp24 else tlink;
x:= b18c17 + b21;
hulp1:= x + x + x + x + b20c19 + 8;
if prot then
  bgn
  if hulp1 > 15  $\wedge$  dgp[0]  $\geq$  0 then undef(27503)
  end;
filled[hulp1]:= true;
m0[hulp1]:= hulp2;
if trace then report writing(hulp1, hulp2);
if b14t0 > max addr then undef(26355);
jcnt:= jcnt + 1; tcnt:= tcnt + 1;
if tcnt = 32 then tcnt:= 0;
tlist[tcnt]:= hulp2 - tp26m1;
ot:= b14t0;
goto time0;

```

subc:

```

if b16c15 > 1 then
  bgn if b16c15 = c+2 then goto skip end else
  if b16c15 = 1 then
    bgn if of = 0 then goto skip end;
  calc addr;
  subcd:= bt = 1  $\wedge$  b18c17 = 3;
  gate:= bt = 0  $\wedge$  addr > 256  $\wedge$  addr < 320;
  if mc addr then bdown1;
  if b16c15 = 1 then of:= 0;
  if gate then prot:= false;
  if addrop then val:= addr else
  if addr = 62 then undef(344) else
  bgn cg:= 0; memreg; val:= m end;
  if gate then prot:= true;
  hulp2:= if ingreep type = 2 then tlink + tp24 else tlink;
  addr:= if b > tp18m1 then star(b) else b;
  if 1/addr < 0 then undef(26362);
  if 1/addr op  $\wedge$  b = 0 then undef(26363);
  if gate  $\wedge$  addr < 256 then undef(27502);
  cg:= 0; m:= hulp2; stm;
  b:= b + 1;
  if abs(val) > tp18m1 then val:= star(val);
  if 1/val < 0 then undef(26352) else
  if val > tp18m3 then else
  if val > max addr then undef(26353);
  jcnt:= jcnt + 1; tcnt:= tcnt + 1;
  if tcnt = 32 then tcnt:= 0;
  tlist[tcnt]:= hulp2 - tp26m1;
  ot:=val;
  if gate then bgn bt:= 1; iv:= 0 end;
  if subcd then iv:= 0;
  goto time2;

```

do:

```

if b16c15 > 1 then
  bgn if b16c15  $\neq$  c+2 then goto skip end else
  if b16c15 = 1 then
    bgn if of = 0 then goto skip end;
    calc addr;
    if mc addr then bdown1;
    if b16c15 = 1 then of:= 0;
    if b18c17 = 3 then s:= addr;
  if addr = 62 then undef(345) else
    bgn cg:= 0; memreg; val:= m end;
    or:= val;
    dcs:= dcs + 1; if dcs > dcs max then undef(3);
    if key then
      bgn ingreep type:= 1; ir addr:= 28; goto ingreep end;
      if trace then report eod(addr, val);
      goto basis cyclus 2;

```

```

clp:
  if b16c15 = 1 then undef(212);
  if b16c15 > 1 then
    bgn if b16c15  $\neq$  c+2 then goto skip end;
    c:= parbit(last par word);
    goto time0;

```

```

int:
  if b16c15 > 1 then
    bgn if b16c15  $\neq$  c+2 then goto skip end;
    val:= nint; nint:= 0;
    if b16c15  $\neq$  1 then c:= val;
    goto time0;

```

```

rgt:
  if b16c15 > 1 then
    bgn if b16c15  $\neq$  c+2 then goto skip end;
    val:= if b11t5 = 8 then a else
           if b11t5 = 9 then s else b;
    if b21 = 1 then val:= - val;
    if b18c17 > 0 then cond;
    if b16c15  $\neq$  1 then
      bgn if b4t0 = 0 then a:= val else
           if b4t0 = 1 then s:= val else b:= val
      end;
      goto time0;

```

```

lees q:
  if b16c15 > 1 then
    bgn if b16c15  $\neq$  c+2 then goto skip end;
    if b20 = 0 then l:= b5t0 else
      bgn
        l:= b5t0 + b;
        if l < 0  $\vee$  l > 1 then undef(141)
      end;
    if b11t9 = 5 then val:= if l = 0 then if0 else if1 else

```

```

if b11t9 = 6 then val := if l = 0 then lvif0 else lvif1 else
undef(145);
if b8t6 = 1 then
val := and( val, if b24 = 0 then a else s);
if b18c17 > 0 then cond;
if b16c15 ≠ 1 then
bgn if b24 = 0 then a := val else s := val end;
goto time1;

```

```

maak q:
if b16c15 = 1 then undef(213);
if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
if bt = 0 then undef(130);
if b20 = 0 then l := b5t0 else
bgn
if b < 0 then undef(142);
l := b5t0 + b
end;
if l > 39 then undef(143);
if b11t9 = 0 then
bgn
if read q(l, afv0, afv1) = 0 then undef(147);
set q(l, af0, af1, b24); attendee charon af(l)
end else
if b11t9 = 1 then
bgn
if read q(l, ifv0, ifv1) = 0 then undef(148);
set q(l, if0, if1, b24)
end else
if b11t9 = 2 then
bgn
if read q(l, lvifv0, lvifv1) = 0 then undef(149);
set q(l, lvif0, lvif1, b24)
end else
undef(146);
goto time1;

```

```

maak iv:
if b16c15 = 1 then undef(214);
if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;
if bt = 0 then undef(131);
iv := b24;
goto time0;

```

```

maak ov:
if b16c15 = 1 then undef(215);
if b16c15 > 1 then
bgn if b16c15 ≠ c+2 then goto skip end;

if bt = 0 then undef(132);
ov := b24;

```

```

    goto time0;

itvon:
    if b16c15 = 1 then undef(216);
    if b16c15 > 1 then
        bgn if b16c15  $\neq$  c+2 then goto skip end;

    if bt = 0 then undef(133);
    itv:= 1;
    goto basis cyclus1;

memprot:
    if b16c15 = 1 then undef(217);
    if b16c15 > 1 then
        bgn if b16c15  $\neq$  c+2 then goto skip end;

    if bt = 0 then undef(134);
    b20c19:= 2; b14t0:= tp14; calc addr;
    cg:= 0; mem reg; val:= m;
    hulp1:= bitstring(17,13,val)  $\times$  16;
    if hulp1 > max addr : tp9 then undef(26361);
    val:= circ shift(val, 3);
    for hulp3:= 0 step 1 until 15 do
        dgp[hulp1 + hulp3]:=
            bit(hulp3, val)  $\times$  2 + bit(hulp3 : 8 + 25, val) - 1;

    b up1;
    goto time2;

operate:
    if b16c15 = 1 then undef(226);
    if b16c15 > 1 then
        bgn if b16c15  $\neq$  c+2 then goto skip end;

    if b20c19  $\neq$  2 then l:= b14t0 else l:= b14t0 + b;
    if l < 0  $\vee$  l > 15 then undef(154);
    bgn
        switch act:= 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,16;
        goto act[l + 1];
1:  newpage;      goto time0;
2:  s:= rehep;   goto time0;
3:  puhep(s);    goto time0;
4:  col(s);      goto time0;
5:  prsym(s);    goto time0;
6:  s:= resym;   goto time0;
7:  pusym(s);    goto time0;
8:  csym(s);     goto time0;
9:  proct(s, 9); goto time0;
10: s:= reoct;   goto time0;
11: undef(155);  goto time0;
12: undef(156);  goto time0;
13: print(s);    goto time0;
14: s:= read;    goto time0;
15: punch(s);    goto time0;

```

```

16: cpunch(s);   goto time0;
end operate;

time1:
  ccs:= ccs + (if 1 < 16 then 1 else 2);
  goto time0;

time12:
  ccs:= ccs + 1 + (1 - 1) : 2;
  goto time0;

skip:
  if trace then report skip; goto time0;

time3:
  ccs:= ccs + 1;
time2:
  ccs:= ccs + 1;
time1:
  ccs:= ccs + 1;
time0:
  if ingreep type > 0 then bgn bt:= 1; iv:= 0 end;

  ingreep type:= itv:= 0; prot:= bt = 0;
  if sva then undef(0);

  if charon teller > 0 then
    bgn
      charon teller:= charon teller - ccs + ccs1;
      if charon teller ≤ 0 then wek charon
    end;

  ccs1:= ccs; dcs:= 0;
  if trace then report eoi;
  if key then
    bgn ingreep type:= 1; ir addr:= 28; goto ingreep end;

  if iv = 0 then x:= y:= 0 else
    bgn
      x:= lvif0; y:= lvif1;
      if if0 + if1 > 0 then
        bgn
          if and(and(ie0, lvif0), if0) +
            and(and(ie1, lvif1), if1) > 0 then
            bgn co charon ingreep;
              ingreep type:= 1; ir addr:= 24; goto ingreep
            end charon ingreep
          end minstens een if aan
        end horend;

      ie0:= x; ie1:= y; co bewaren voor volgende opdracht;

```

```
goto basis cyclus 1;  
  
ingreep:  
  if trace then report interrupt(ir addr);  
  if itv = 1 then undef(135);  
  prot:= false;  
  cg:= 0; addr:= ir addr; mem1; or:= m;  
  goto basis cyclus 2;  
  
end  
end
```

9. Het testprogramma.

9.1. Het is buitengewoon moeilijk een omvangrijk programma als de X8-simulator uitputtend te testen. Gedeeltelijk werd geprobeerd door een enigszins orthogonale opbouw van het programma sommige delen afzonderlijk testbaar te maken; zo is het adresseringsgedeelte afzonderlijk geprogrammeerd, zodat uit het correct werken van b.v. de opdrachten A + DYN en GOTO(STAT) geconcludeerd kan worden dat ook GOTO(DYN) correct zal werken. Hierbij moet wel aangetekend worden dat de EL X8 niet volgens dit principe werkt; soms zijn van een opdracht vele adresvarianten als aparte pulsenrijen uitgeprogrammeerd.

Verder werd gepoogd het aantal fouten laag te houden door zoveel mogelijk gebruik te maken van rigoureuze correcte redeneringen en zo weinig mogelijk toe te geven aan de verleiding bepaalde stukken "maar wat slimmer" te programmeren of om aan te nemen dat wat voor drie voorbeelden goed gaat wel altijd goed zal gaan. (Zie 5.7.)

Tenslotte werd, om schrijffouten en eventueel andere fouten te ontdekken, een testprogramma geschreven dat zoveel mogelijk elke ALGOL 60-opdracht in de X8-simulator minstens eenmaal doet uitvoeren. Van elke geteste situatie worden resultaten geponst; door nu het testprogramma zowel op de EL X8 als via de simulator te verwerken en de aldus verkregen banden te vergelijken, kunnen de werking van de EL X8 en die van de simulator met elkaar vergeleken worden.

Bij het op deze wijze testen van de X8-simulator kwamen fouten aan het licht die in de volgende drie klassen in te delen zijn:

- a. programmeerfouten, voornamelijk vergeten initialisaties,
- b. fouten in de EL X8 (zie 10.3.3., 10.7. en 10.16.),
en de grootste klasse,
- c. misvattingen over de precieze werking van de EL X8.

In de logische opzet werden geen fouten gevonden.

Het testprogramma test vrijwel het gehele opdrachtenarsenaal van de EL X8, ook de meer bizarre varianten, en is van uitgebreid commentaar voorzien; gezien ook de slechte toegankelijkheid van de testprogramma's van Electrologica en hun geringe leesbaarheid, volgt daarom hieronder de tekst van het testprogramma. De lezer moet zich daarbij wel realiseren dat het programma een programma test en geen machine; hierdoor ontstaan de volgende opvallende verschillen met een machinetest:

- a. Een opdracht die een keer goed gaat, gaat de volgende keer onder dezelfde omstandigheden weer goed; dit is bij een elektronische schakeling niet noodzakelijk. Het programma doet elke test dus slechts eenmaal.
- b. Aangezien de processen in de X8-simulator en op de EL X8 op verschillende wijze geïmplementeerd zijn, schuilen de gevaren soms in verschillende hoeken. Zo is het verstandig bij de simulator te testen of +0 wel correct geanalyseerd wordt als zijnde de opdracht A + M[0], terwijl zulks op de EL X8 vanzelf spreekt.

9.2. Tekst van het testprogramma.

Hierna volgt de tekst van het testprogramma, zoals vertaald door de MC ELAN-assembler (zie MR132/72), gevolgd door de regeldrukkeroutput welke de X8-simulator leverde bij het verwerken van het testprogramma.

[illegible]

PROTECTIEPOORTEN

47
48
49
50 '00J404': GATE LIST(4): " PROTECTIEPOORT
PUNCH GATE: " INDICATIE
51 '00J404': '022025252' A = '25 252'
52 '00J405': '577001001' SUB15(:PUSTAT) " IN BESTUURSTOESTAND
53 '00J406': '526475377' GOTOR(MC(-1)) " EXIT, RESTORE

54
55 '00J407': DUMMY GATE:
56 '00J407': '526475377' GOTOR(MC(-1)) " DUMMY
57
58 '00J410': OPEN GATE: " PROTECTIEPOORT
59 '00J410': '022025252' A = '52 525' " INDICATIE
60 '00J411': '577001001' SUB15(:PUSTAT)
61 '00J412': '526075377' GOTO(MC(-1)) " EXIT, BT BLIJFT 1

62
63
64 '00J474': GATE LIST(60):
65 '00J474': BAD ADDR:
66 '00J475': '000000000' +0
67 '00J475': DUMMY GATE ADDR:
68 '00J475': '000000407' :DUMMY GATE
69 '00J476': PUNCH GATE ADDR:
70 '00J476': '000000404' :PUNCH GATE

```

***** HULPROUTINES *****
75 /
76 //
77 //
78 M[512]:
79 '001000': IN SIM:
80 '001000': +1
81
82
83 PUSTAT:
84
85 2
86
87 '001001': '060001063'
88 '001002': '160001064'
89 '001003': '460001065'
90 '001004': '422000003'
91 '001005': '460001067'
92 '001006':
93 '001006': '574441067'
94 '001007': '432000004'
95 '001010': '770000125'
96 '001011': '362000077'
97 '001012':
98 '001012': '160001066'
99 '001013': '120001066'
100 '001014': '660063000'
101 '001015': '023000200'
102 '001016': '121101000'
103 '001017': '522301044'
104 '001020': '640000002'
105 '001021': '760071022'
106 '001022':
107 '001022': '403000001'
108 '001023': '123000000'
109 '001024': '760300107'
110 '001025': '522301012'
111 '001026': '422000001'
112 '001027': '471101067'
113 '001030': '522301006'
114 '001031': '020001063'
115 '001032': '120001064'
116 '001033': '420001065'
117 '001034': '520400027'
118
119 TERM:
120 '001035': '120002734'
121 '001036': '122000001'
122 '001037': '160000027'
123 '001040': '122000177'
124 '001041': '432000001'
125 '001042': '470001067'
126 '001043': '522001012'
127
128 TRANSBIN:
129 '001044': '160001062'
130 '001045': '122001060'
131 '001046': '160000210'
132 '001047': '160000211'
133 '001050': '120001061'
134 '001051': '160000212'
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
```

```

135 '001052': '760070022' AFON(PUNCH) " START DE BANDPONSER
136 '001053': '20000211' S = PAR(1) " IFT
137 '001054': '771000062' RUS(18), Z " TELLINGDEEL 0?
138 '001055': '512200003' Y, JUMP(-3)
139 '001056': '522001022' Y, GOTO(:RET)
140
141 '001057': PSS: " STARTSCHAKEL
142 '001057': '000000000' +0 " SLOTWOORD
143 '001060': '000001060' :PSS(1) " GEEN VOLGENDE SCHAKEL
144 '001061': '001001061' ('1000000' + :BIN(-1))" TEL := 1, ADR := BIN
145 '001062': BIN:
146 '001062': 'SKIP' 1
147
148 '001063': SAVE A: 'SKIP' 1
149 '001064': SAVE S: 'SKIP' 1
150 '001065': SAVE B: 'SKIP' 1
151
152 '001066': GEH: 'SKIP' 1
153 '001067': CE: 'SKIP' 1
154
155 '001070': INSTRU:
156 '001070': S = LINK(5)
157 '001071': '120001064' S = SAVE S
158 '001072': '120001063' S = SAVE A
159 '001073': INSTR1:
160
161 2 'END' " PUSTAT, TERM
162
163
164 '001073': CRASH1:
165
166 " SUB(:CRASH1),
167 " STOPT HET PROGRAMMA, EN WEL ALS VOLGT:
168 " OP DE X8: BT = 0: IN ADRESINGREEP, A1
169 " BT = 1: IN LOOP
170 " IN DE SIMULATOR!
171 " BT = 0: IN ADRESINGREEP
172 " BT = 1: OP DE OPDRACHT OPOFF
173
174 3 'BEGIN'
175 '001073': '600061000' OPOFF
176 '001074': '020002722' A = D25
177 '001075': '261000076' A := T, Z
178 '001076': '520202723' Y, GOTO(SGN BIT)
179 '001077': '512000001' JUMP(-1)
180 'END' CRASH1
181
182 3
183
184 '001100': SUBC TEST:
185
186 " SUB2(:SUBC TEST)
187 " DE STAPELENDE SUBROUTINESPRONG DIE IN HET
188 " S-REGISTER WORDT MEEGEGEVEN, WORDT UITGEVOERD MET
189 " EEN AANTAL VERSCHILLENDE WAARDEN VAN B, TERWIJL
190 " DE KLEINE REGISTERS GEZET WORDEN VOLGENS
191 " BIT25 - BIT18 VAN A.
192 'BEGIN' NEXTB, BLISTU, BLISTU, RETURN, PROF, ADDR, INT WORDS, REG MASK,
193 REGISTERS, SAVE WA, INSTR
194
195 '001100': '160001157' INSTR = S
196 '001101': '260001136' A := REG MASK
197 '001102': '060001160' REGISTERS = A
198 '001103': '620000032' F = WA
199 '001104': '720001161' SAVE WA = F

```

```

195 '001105': '620001134' F = INT WORDS
196 '001106': '720000032' WA = F
197 '001107': '022000020' A = (:BLIST1=:BLIST0) " FOUTADRES- EN PROTECTIE INGREPEN
198 '001110': '060000007' COUNT = A " LENGTE VAN DE LIJST
199 '001111': '622001136' G = :BLIST0(-1) " G LOOPT OVER DE LIJST
200 '001112': NEXT B:
201 '001112': G + 1
202 '001113': '426072400' B = MG " LOPEDE WAARDE VAN B
203 '001114': '022000777' A = '777' " INDICATIE
204 '001115': '120001160' S = REGISTERS " ZET EENBIT-REGISTERS
205 '001116': '500400074' JUMPR(S) " ZOALS VERLANGD
206 '001117': '570401157' DO(INSTR) " INGREEP (FOUTADRES- OF PROTECTIE-) OF
207 " NORMAAL VERLOOP
208 '001120': '032000777' A = - '777' " INDIEN NORMAAL UITGEVOERD (EN TERUSGEKEERD),
209 '001121': '526076377' GOTO(MT[-1]) " FORCEER DAN EEN ADRESINGREEP
210 '001122': RETURN:
211 '001122': REPP(:NEXT B) " VOLGENDE
212 '001123': '620001161' F = SAVE WA
213 '001124': '720000032' WA = F
214 '001125': '520400012' GOTOR(LINK2) " HERSTEL (WA, PR)
215
216 '001126': PROT:
217 '001126': '260002725' A 'LPU " INDICATIE
218 '001127': '502000001' JUMP(1)
219 '001130': ADDR:
220 '001130': '260002724' A 'LPU " INDICATIE
221 '001131': '120000075' S = B
222 '001132': '577001001' SUB15(:PSTAT) " PONS STATUS
223 " NA SOMMIGE INGREPEN BIJ EEN SUBC-OPDRACHT IS DE
224 " WAARDE VAN T ONGEDEFINIËRD EN DE LINK DUS
225 " ONBRUIKBAAR, WE SPRINGEN DAAROM RECHTSTREEKS TERUG
226
227 '001133': '522001122' GOTO(:RETURN)
228
229 '001134': INT WORDS:
230 '001134': '522001130' GOTO(:ADDR)
231 '001135': '522001126' GOTO(:PROT)
232 '001136': REG MASK:
233 '001136': '377000000' '377 000 000'
234
235 '001137': BLIST0:
236 '001137': '000000000' '0'
237 '001140': '000000001' '1'
238 '001141': '000000050' '50'
239 '001142': '000000077' :D
240 '001143': '000000377' :GATE LIST(-1)
241 '001144': '000000400' :GATE LIST
242 '001145': '000000477' :GATE LIST(63)
243 '001146': '000000500' :GATE LIST(64)
244 '001147': '000100000' (MEM LENGTH / 2)
245 '001150': '000177777' (MEM LENGTH - 1)
246 '001151': '000200000' (MEM LENGTH)
247 '001152': '001000000' '1 000 000'
248 '001153': '100000000' '100 000 000'
249 '001154': '377177777' ('377 000 000' + MEMLENGTH - 1)
250 " GROOTST MOGELIJKE VEILIGE WAARDE VAN B
251
252 '001155': '400000000' '400 000 000'
253 '001156': '777777777' -0
254 '001157': BLIST1:
255 INSTR:
256 '001157': 'SKIP'

```

```

255 '001160': REGISTERS:
256 '001160': 'SKIP' 1
257 '001161': SAVE WAI
258 '001161': 'SKIP' 2
259 '001161': 'END' SUBC TEST
260
261 '001163': DO FF:
262 " SUB2(100 FF), VOERT EEN IN S MEEGEGEVEN TRANSPUT-
263 " BESTUURSOPDRACHT VOOR EEN AANTAL APPARATEN UIT
264
265 5
266 '001163': '0220000003' 'BEGIN' NEXT, LIST0, LIST1
267 '001164': '0600000007' A = (:LIST1 - :LIST0) " LENGTE VAN DE LIJST
268 '001165': '6220001174' COUNT = A " IN 'COUNT'
269 '001166': '4220000023' G = :LIST0(-1) " G LOOPT OVER DE LIJST
270 '001167': '6020000001' B = TEST DEV 2
271 '001167': '0200000074' NEXT:
272 '001170': '006072400' G + 1 " COPIE VAN DE OPDRACHT
273 '001171': '570400073' A + MG " VORM OPDRACHT,
274 '001172': '596401167' DO(A) " EN VOER HEM UIT
275 '001173': '520400012' REPR(:NEXT)
276 '001174': LIST0:
277 " (TEST DEV 1) " EERSTE WOORD
278 " (B), TWEEDE WOORD
279 " (14 000 000) + TEST DEV 3 - TEST DEV 2)
280 " (B + N), ADMINISTRATIEF APPARAAT
281 " LIST1:
282 " 'END' DO FF
283
284 6
285 '001200': DYST S:
286 " SUBC(10YST S)
287
288 6
289 '001200': '100002737' 'BEGIN'
290 '001201': '520400074' S + DYST NBT " VORM DYNAMISCH STOPADRES
291 " GOTOR(S) " ZET HET IN T
292 " 'END'
293
294 7
295 '001202': SET PROT:
296 " SUB8(1SET PROT), ZET DE PROTECTIEBITS VAN ALLE
297 " HALVE KASTEN ZOALS AANGEGEVEN IN S
298
299 '001202': '0300001211' 'BEGIN' LOOP, MAX ADDR, HALF MODULE = '20 000'
300 '001203': A = -MAX ADDR
301 '001203': B = IS
302 '001204': '4220000074' MEMPROT
303 '001205': '766075400' S + HALF MODULE " VORM VOLGEND PROTECTIEWOORD
304 '001206': '102020000' A + HALF MODULE, Z " BESCHRIJFT S DE EERSTE NIET-AANWEZIGE KAST?
305 '001207': '003020000' N, GOTOR(1LOOP)
306 '001210': '522301203' GOTOR(LINK8)
307
308 '001211': MAX ADDR:
309 '001211': '0002000000' (MEM LENGTH)
310 'END' SET PROT
311
312 7
313 '001212': DUMMY:
314 " SUBC(1DUMMY)

```

12/2-107

D 2220V.0 GRUNE

9

17

315 '001212': '526475377' GOTOR(MC[-1])
316 '001213': DUMMY ADDR:
317 '001213': '000001212' :DUMMY

```

***** TESTS *****
323 '001214': ENTRY:
324 '001214': LV:OFF(PUNCH) " LEG DE BANDPONSER HET ZWIJGEN OP
325 '001214': '600072022'
326
327 " TESTPROGRAMMA VOOR ADRESSERINGSVORMEN
328 '001215': '502400000' " ALLE EENBIT-REGISTERS WORDEN 0
329 '001216': '022000000' " :STAT
330 '001217': '132000000' " :STAT
331 '001220': '577001001' " PONS STATUS
332 '001221': '120001772' " STAT
333 '001222': '030001772' " STAT
334 '001223': '577001001' " PONS STATUS
335 '001224': '422001773' " BEGINWAARDE
336 '001225': '124040000' " STAT
337 '001226': '034037776' " - STATB
338 '001227': '577001001' " PONS STATUS
339 '001230': '622001772' " DYN
340 '001231': '126072000' " DYN
341 '001232': '026072777' " PONS STATUS
342 '001233': '577001001' " PONS STATUS
343 '001234': '022001774' " DYN
344 '001235': '126073405' " PONS STATUS
345 '001236': '577001001' " PONS STATUS
346 '001237': '122001400' " DYN
347 '001240': '026074777' " DYN
348 '001241': '577001001' " PONS STATUS
349 '001242': '126075400' " DYN, B-MOD
350 '001243': '026075400' " DYN, B-MOD
351 '001244': '577001001' " PONS STATUS
352 '001245': '026076400' " PONS STATUS
353 '001246': '126076400' " DYN, T-MOD
354 '001247': '577001001' " PONS STATUS
355 '001250': '460000077' " D WIJST NU MIDDEN IN DE INSTRUCTIES
356 '001251': '026077400' " DYN
357 '001252': '126077400' " DYN
358 '001253': '577001001' " PONS STATUS
359 '001254': '422002766' " PONS STATUS
360 '001255': '460000077' " D WIJST NAAR EEN LIJST VAN 58 WOORDEN
361 '001256': '022003060' " D WIJST NAAR EEN LIJST VAN 58 WOORDEN
362 '001257': '064040000' " MD[0] WIJST NAAR 'REST'
363 '001260': '466073400' " MD[0] GEVULD ZONDER GEBRUIK VAN MP[Q]-ADRESSERING
364 '001261': '002000001' " :REST[1]
365 '001262': '064040071' " MD[57] = :REST[1]
366 '001263': '726173412' " MD[10] = G
367 '001264': '026000400' " MD[10] = G
368 '001265': '126071412' " S = M57[+10]
369 '001266': '577001001' " S = M57[+10]
370 '001267': '022000012' " SUB15(:PUSAT)
371 '001270': '066077401' " A = 10
372 '001271': '466073366' " MD[1] = A
373 '001272': '126001366' " MA[-10] = B
374 '001273': '577001001' " S = M1[-10]
375 '001274': '422003060' " SUB15(:PUSAT)
376 '001275': '076075400' " B = :REST
377 '001276': '166075400' " MC = -A
378 '001277': '476075400' " MC = S
379 '001300': '120003060' " MC = -B
380 '001301': '020003061' " S = REST
381 '001302': '577001001' " A = REST[1]
    SUB15(:PUSAT) " PONS STATUS
    " BEKIJK RESULTAAT
    " PONS STATUS

```

12:12/2-10/ D 2220V.0 GRUNE

```

382 '001303': '120003062' S = REST(2) " IS B JUIST?
383 '001304': '020000075' A = B " PONS STATUS
384 '001305': '577001001' SUB15(:PUSTAT)
385 '001306': '422000076' B = !T
386 '001307': '024000077' A = M(B) " STATB MAG DE REGISTERS ADRESSEREN
387 '001308': '024040000' S = M(B + (:G - :T)) " MOET BIJ G UITKOMEN
388 '001309': '124037774' SUB15(:PUSTAT) " PONS STATUS
389 '001310': '577001001' " AANGEZIEN VOOR DYN-ADRESSERING HET :DYN-ADRESSERINGSMECHANISME
390 " GEBRUIKT WORDT, BEHOEFT DIT NIET AFZONDERLIJK GETEST TE WORDEN,
391 " BEHALVE VOOR DE CONSTRUCTIE :MC
392 '001312': '422002766' B = :STACK
393 '001313': '162075000' S = :MC(-256) " IDYN
394 '001314': '020000075' A = B " PONS STATUS
395 '001315': '577001001' SUB15(:PUSTAT) " IDYN
396 '001316': '02075777' A = :MC(255)
397 '001317': '120000075' S = B " PONS STATUS
398 '001320': '577001001' SUB15(:PUSTAT)
399
400 " TEST BEPALING PARITEITSBIT, EN CONDITIE-ZETTING OP INTEGERS
401 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
402 '001321': '502400000' S = LP1
403 '001322': '120002725' SUB15(:PUSTAT) " PONS STATUS
404 '001323': '577001001' S = LP0
405 '001324': '120002724' SUB15(:PUSTAT) " PONS STATUS
406 '001325': '577001001' S = -LP1
407 '001326': '130002725' SUB15(:PUSTAT) " PONS STATUS
408 '001327': '577001001' S = -LP0 " MAAKT NIKS UIT
409 '001328': '130002724' SUB15(:PUSTAT) " PONS STATUS
410 '001329': '577001001' S = 0, P " C: 0, LT: 0
411 '001330': '122400000' SUB15(:PUSTAT) " PONS STATUS
412 '001331': '577001001' S = -0, P " C: 1, LT: 1
413 '001332': '132400000' SUB15(:PUSTAT) " PONS STATUS
414 '001333': '577001001' S = 0, Z " C: 0, LT: 0
415 '001334': '123000000' SUB15(:PUSTAT) " PONS STATUS
416 '001335': '577001001' S = -0, Z " C: 0, LT: 1
417 '001336': '133000000' SUB15(:PUSTAT) " PONS STATUS
418 '001337': '577001001' S = SGN BIT, Z " C: 1, LT: 1
419 '001338': '121002723' SUB15(:PUSTAT) " PONS STATUS
420 '001339': '577001001' S = -0, E " C: 0, LT: 1
421 '001340': '133400000' SUB15(:PUSTAT) " PONS STATUS
422 '001341': '577001001' S = 0, E " C: 1, LT: 0
423 '001342': '123400000' SUB15(:PUSTAT) " PONS STATUS
424
425 " TEST OPDRACHTHERKENNING
426 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
427 '001350': '502400000' A = 5
428 '001351': '022000005' M(0) = A
429 '001352': '060000000' A = -4
430 '001353': '032000004' A + M(0) " 0 ALS OPDRACHT
431 '001354': '000000000' B = -3 " -MAX INT ALS OPDRACHT
432 '001355': '432000003' B + M(0)
433 '001356': '400000000' S = B " PONS STATUS
434 '001357': '120000075' SUB15(:PUSTAT)
435 '001358': '577001001' S = :REST(-255)
436 '001359': '122002461' D = S " BITPATROON
437 '001360': '160000077' S = LP0 " IN REST
438 '001361': '120002724' REST = S " ANDER BITPATROON, C: 1
439 '001362': '160003060' S = DYST NBT, Z " MAX INT ALS OPDRACHT
440 '001363': '121002737' N.S. (*, -MD(255)), E " PONS STATUS
441 '001364': '377777777' SUB15(:PUSTAT)
442 '001365': '577001001'

```

```

442 " -U IS GEEN OPDRACHT
443
444 " TEST ARITMETISCHE BEWERKINGEN IN A-, S- OF B-REGISTER
445 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
446 S = LP0, Z " C:= 1
447 S + LP1, Z " C:= 0, LT:= 1, S:= -0
448 S - LP1, Z " C:= 1, LT:= 1, S:= -0
449 A = SGN BIT " OF:= 1, A:= 1
450 A = SGN BIT " PONS STATUS
451 SUB15(:PUSTAT) " OF:= 0
452 U, JUMP(0) " OF:= 1, S:= -1
453 S = SGN BIT " PONS STATUS
454 S - SGN BIT " OF:= 1, S:= -1
455 SUB15(:PUSTAT) " PONS STATUS
456 U, JUMP(0) " S:= 2*26 - 1
457 S = -SGN BIT " B > 0
458 B = :REST " B BIJ M[B] GETELD, OVERFLOW, B VERHOOGD
459 M[B] = S
460 MC + B
461 S = B
462 A = REST " PONS STATUS
463 SUB15(:PUSTAT) " S + IDYN
464 B = :STACK " S + IDYN
465 S + :MC " PONS STATUS
466 A = B " B = -U
467 SUB15(:PUSTAT) " A = -U
468 B = :MC " PONS STATUS
469 S = B " PONS STATUS
470 SUB15(:PUSTAT) " STACK := :STACK
471 B = :STACK " STACK:= (:STACK * 2), B:= (:STACK * 2) + 1, C:= 1
472 M[B] = B
473 PLUSB(MC(0)), Z
474 A = STACK
475 S = B
476 SUB15(:PUSTAT) " PONS STATUS
477
478 " TEST LOGISCHE OPERATIES OP A EN S
479 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
480 S = LP0 " SU
481 A = LP1 " A0
482 S + A " S:= SU + A
483 A + A " A:= AU + A
484 S + A " S:= SU + A
485 SUB15(:PUSTAT) " PONS STATUS
486 A = -LP0 " A = -LP0
487 A + LP1 " A = -LP0 + LP1 = -0
488 S = A " S = LP0 + LP1 = -0
489 SUB15(:PUSTAT) " PONS STATUS
490
491 " TEST VERMENIGVULDIGING EN DELING
492 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
493 S = 10 000, P " C = 0
494 M[LP0], Z " C = 1
495 SUB15(:PUSTAT) " PONS STATUS
496 S = LP0 " S > 0
497 A = LP1 " A < 0
498 G = 1
499 MULAS(G), P " S * 1 + A = -0
500 SUB15(:PUSTAT) " PONS STATUS
501 A = 0

```

```

202 '001455': '132000000' S = -0
203 '001456': '632000001' G = -1
204 '001457': '200000072' MULAS(G)
205 '001460': '577001001' SUB15(:PUSTAT) " S = -1 + A = +0
206 '001461': '022000000' A = 0 " PONS STATUS
207 '001462': '122000000' S = 0
208 '001463': '303000001' DIVAS(1), Z " A = S = C = 0
209 '001464': '577001001' SUB15(:PUSTAT) " PONS STATUS
210 '001465': '313000001' DIVAS(-1), Z " A = S = -0, C = 0
211 '001466': '577001001' SUB15(:PUSTAT) " PONS STATUS
212 " ZIE VERDER DE AFZONDERLIJKE TESTPROGRAMMA'S VOOR MULS, MULAS, DIVA, DIVAS
213
214 '001467': '502400000' JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
215 '001470': '133000001' S = -1, Z " C = 1
216 '001471': '760000071' LUS(25) " S = -2**25
217 '001472': '761002000' TENAS, Z " A = -D, S = -0, C = 0
218 '001473': '577001001' SUB15(:PUSTAT) " PONS STATUS
219
220 " TESTS VOOR SHIFT-INSTRUCTIES
221 '001474': '502400000' JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
222 '001475': '020002724' A = LP0
223 '001476': '120002725' S = LP1
224 '001477': '422000011' B = 9
225 '001500': '664400000' LCA(B), P
226 '001501': '577001001' SUB15(:PUSTAT) " PONS STATUS
227 '001502': '765400001' LCS(B + 1), E
228 '001503': '577001001' SUB15(:PUSTAT) " PONS STATUS
229 '001504': '664400102' LCAS(B + 2), P
230 '001505': '577001001' SUB15(:PUSTAT) " PONS STATUS
231 '001506': '765400103' LCSA(B + 3), E
232 '001507': '577001001' SUB15(:PUSTAT) " PONS STATUS
233 '001510': '675400004' RCA(B + 4), E
234 '001511': '577001001' SUB15(:PUSTAT) " PONS STATUS
235 '001512': '774400005' RCS(B + 5), P
236 '001513': '577001001' SUB15(:PUSTAT) " PONS STATUS
237 '001514': '675400106' RCAS(B + 6), E
238 '001515': '577001001' SUB15(:PUSTAT) " PONS STATUS
239 '001516': '774400107' RCSA(B + 7), P
240 '001517': '577001001' SUB15(:PUSTAT) " PONS STATUS
241 '001520': '422000001' B = 1
242 '001521': '665400046' LUA(B + 6), E
243 '001522': '577001001' SUB15(:PUSTAT) " PONS STATUS
244 '001523': '764400045' LUS(B + 5), P
245 '001524': '577001001' SUB15(:PUSTAT) " PONS STATUS
246 '001525': '664400144' LUAS(B + 4), P
247 '001526': '577001001' SUB15(:PUSTAT) " PONS STATUS
248 '001527': '765400143' LUSA(B + 3), E
249 '001530': '577001001' SUB15(:PUSTAT) " PONS STATUS
250 '001531': '020002724' A = LP0
251 '001532': '120002725' S = LP1
252 '001533': '674400046' RUA(B + 6), P
253 '001534': '577001001' SUB15(:PUSTAT) " PONS STATUS
254 '001535': '774400041' RUS(B + 1), P
255 '001536': '577001001' SUB15(:PUSTAT) " PONS STATUS
256 '001537': '675400144' RUAS(B + 4), E
257 '001540': '577001001' SUB15(:PUSTAT) " PONS STATUS
258 '001541': '775400140' RUSA(B + 0), E
259 '001542': '577001001' SUB15(:PUSTAT) " PONS STATUS
260 " ZIE VERDER HET AFZONDERLIJKE TESTPROGRAMMA VOOR SHIFTS
261

```

```

202 " TEST NORMEER-INSTRUCTIES
203 " JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
204 A = 0
205 NORA, Z " A := 0, B := 26
206 S = B
207 SUB15(:PUSTAT) " PONS STATUS
208 A = -B " A := -26
209 NORA, Z
210 S = B
211 SUB15(:PUSTAT) " PONS STATUS
212 A = 0
213 S = SGN BIT, Z " C = 1
214 NORA, Z " C = 0, SCHUIFT OM TEKENBIT VAN S MEEN
215 SUB15(:PUSTAT) " PONS STATUS
216 S = B " B = 52
217 SUB15(:PUSTAT) " PONS STATUS
218 A = -0
219 S = -SGN BIT
220 NORA, P " C = 1, SCHUIFT OM TEKENBIT MEEN
221 SUB15(:PUSTAT) " PONS STATUS
222 S = B " B = 52
223 SUB15(:PUSTAT) " PONS STATUS
224
225 " TEST F-TRANSPORT
226 " JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
227 B = :LPO
228 F = MC, P
229 REST = -F
230 A = REST
231 S = REST[1]
232 SUB15(:PUSTAT) " PONS STATUS
233 S = B
234 SUB15(:PUSTAT) " PONS STATUS
235
236 " TEST CONDITIE-ZETTING, OF, NINT OP F-OPDRACHTEN
237 " JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
238 A = 0
239 B = -0
240 F = A, P " C = 0, OF = 1, NINT = 0
241 SUB15(:PUSTAT) " PONS STATUS
242 A = -0
243 S = 0
244 F = A, P " C = 1, OF = 1, NINT = 0
245 SUB15(:PUSTAT) " PONS STATUS
246 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
247 A = 0
248 S = 0
249 F = -A, Z " C = 0, OF = 0, NINT = 0
250 SUB15(:PUSTAT) " PONS STATUS
251 A = 0
252 S = -0
253 F = A, Z " C := 1, OF = 0
254 SUB15(:PUSTAT) " PONS STATUS
255 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
256 A = 0
257 S = SGN BIT
258 F = A, Z
259 SUB15(:PUSTAT) " PONS STATUS
260 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
261 A = 0
262 S = SGN BIT
263 F = A, Z
264 SUB15(:PUSTAT) " PONS STATUS
265 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
266 A = D1310

```

```

022 '001632': '122000000' S = 0 " NINT = 0
023 '001633': '620400073' F = A, P " PONS STATUS
024 '001634': '577001001' SUB15(:PUSTAT) " ALLE EENBIT-REGISTERS WORDEN 0
025 '001635': '502400000' JUMPR(0)
026 '001636': '030000073' A = -A " NINT = 0
027 '001637': '620400073' F = A, P " PONS STATUS
028 '001640': '577001001' SUB15(:PUSTAT) " ALLE EENBIT-REGISTERS WORDEN 0
029 '001641': '502400000' JUMPR(0) " '77777'
030 '001642': '020002733' A = D14TU " NINT = 1
031 '001643': '620400073' F = A, P " PONS STATUS
032 '001644': '577001001' SUB15(:PUSTAT)
033
034 " TEST VERDERE F- EN G- OPDRACHTEN
035 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
036 '001645': '502400000' B = :REST " X = F
037 '001646': '422003060' MC = F
038 '001647': '726075400' A = REST
039 '001650': '020003060' S = REST(1) " PONS STATUS
040 '001651': '120003061' SUB15(:PUSTAT) " PONS STATUS
041 '001652': '577001001' S = B
042 '001653': '120000075' SUB15(:PUSTAT)
043 '001654': '577001001' G = -1 " -0
044 '001655': '632000001' A = F " -1
045 '001656': '020000071' S = G " PONS STATUS
046 '001657': '120000072' SUB15(:PUSTAT) " ALLE EENBIT-REGISTERS WORDEN 0
047 '001660': '577001001' JUMPR(0)
048 '001661': '502400000' A = 1
049 '001662': '022000001' S = 1 " PH = 1, FT = 1
050 '001663': '122000001' F = A " NINT = 1
051 '001664': '620000073' B = :REST
052 '001665': '422003060' MC = G
053 '001666': '726175400' S = B " PONS STATUS
054 '001667': '120008075' SUB15(:PUSTAT)
055 '001670': '577001001' A = -0
056 '001671': '032000000' F = A " ALLE EENBIT-REGISTERS WORDEN 0
057 '001672': '620000073' JUMPR(0) " NINT = 1, WEGENS TEKENINCONSISTENTIE
058 '001673': '502400000' REST = G " PONS STATUS
059 '001674': '720103060' SUB15(:PUSTAT) " F = :DYN
060 '001675': '577001001' F = :MC " BEKIJK
061 '001676': '7220075400' A = F " RESULTAAT
062 '001677': '020000071' S = G " PONS STATUS
063 '001701': '577001001' SUB15(:PUSTAT) " IS B NIET VERANDERD ?
064 '001702': '120000075' S = B " PONS STATUS
065 '001703': '577001001' SUB15(:PUSTAT)
066 '001704': '620002726' F = FP0
067 '001705': '601402730' F = FP2, E "
068 '001706': '020000071' A = F " BEKIJK
069 '001707': '120000072' S = G " RESULTAAT
070 '001710': '577001001' SUB15(:PUSTAT) " PONS STATUS
071 '001711': '620002726' F = FP0
072 '001712': '611402730' F = FP2, E "
073 '001713': '020000071' A = F " BEKIJK
074 '001714': '120000072' S = G " RESULTAAT
075 '001715': '577001001' SUB15(:PUSTAT) " PONS STATUS
076 '001716': '620002726' F = FP0
077 '001717': '701402730' F = FP2, E "
078 '001720': '020000071' A = F " BEKIJK
079 '001721': '120000072' S = G " RESULTAAT
080 '001722': '577001001' SUB15(:PUSTAT) " PONS STATUS
081 '001723': '620002726' F = FP0

```

```

002 '001724': '711402730' F / FP2, E " / BEKIJK
003 '001725': '020000071' A = F " RESULTAAT
004 '001726': '220000072' S = G " PONS STATUS
005 '001727': '577001001' SUB15(:PUSTAT) " ALLE EENBIT-REGISTERS WORDEN 0
006 '001730': '502400000' JUMPR(0)
007 '001731': '620102724' G = LP0
008 '001732': '601502725' G + LP1, E
009 '001733': '020000071' A = F
010 '001734': '120000072' S = G
011 '001735': '577001001' SUB15(:PUSTAT) " PONS STATUS
012 '001736': '620102724' G = LP0
013 '001737': '611502725' G - LP1, E
014 '001740': '020000071' A = F
015 '001741': '120000072' S = G
016 '001742': '577001001' SUB15(:PUSTAT) " PONS STATUS
017 '001743': '620102724' G = LP0
018 '001744': '701502725' G + LP1, E
019 '001745': '020000071' A = F
020 '001746': '120000072' S = G
021 '001747': '577001001' SUB15(:PUSTAT) " PONS STATUS
022 '001750': '620102724' G = LP0
023 '001751': '711502725' G / LP1, E
024 '001752': '020000071' A = F
025 '001753': '120000072' S = G
026 '001754': '577001001' SUB15(:PUSTAT) " PONS STATUS
027 '001755': '502400000' " TEST SPRONG-OPDRACHTEN
028 '001756': '120002747' JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
029 '001757': '160000000' S = GOTOL1 " SPRONG-OPDRACHT
030 '001760': '220000001' M(0) = S " IN M(0)
031 '001761': '700000071' S = 1 " S = 2**25 + 0
032 '001762': '520000074' LUS(25) " S-STER = 0, -> OT:= 0, EN SPRINGT VANDAAR TERUG
033 '001763': '560001073' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
034 '001764': '577001001' L1:
035 '001765': '120000000' SUB15(:PUSTAT) " PONS STATUS
036 '001766': '500000074' JUMP(S) " DOET NIETS
037 '001767': '020002723' A = SGN BIT " OF:= 1
038 '001770': '000000073' A + A
039 '001771': '502100001' U, JUMP(1) " WORDT UITGEVOERD
040 '001772': '560001073' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
041 '001773': '577001001' SUB15(:PUSTAT) " PONS STATUS
042 '001774': '502100001' U, JUMP(1) " SPRINGT NIET
043 '001775': '112000001' S - 1 " WORDT UITGEVOERD
044 '001776': '577001001' SUB15(:PUSTAT) " PONS STATUS
045 '001777': '462076402' B = :MTL1 " B = :MT(Q)
046 '002000': '400002734' B + D18
047 '002001': '536075401' GOTOL(MC(1)) " GOTO(:IDYN), NAAR MTL1(1)
048 '002002': '560001073' MTL1:
049 '002003': '120000075' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
050 '002004': '577001001' S = B
051 '002005': '122000012' SUB15(:PUSTAT) " PONS STATUS
052 '002006': '160000005' " TEST REPETERENDE SPRONGEN
053 '002007': '020002723' S = 10
054 '002010': '000000073' COUNT5 = S
055 '002011': '552102013' A = SGN BIT " STARTWAARDE
056 '002012': '577001001' A + A " OF:= 1
057 '002013': '577001001' U, REP5(:L2) " WORDT UITGEVOERD
058 '002014': '577001001' "

```

12/2-10/ D 2200.0 GRUNE

```

742 '002012:' '560001073' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
743 '002013:' L2: " IS NU 9
744 '002014:' S = COUNT5 " PONS STATUS
745 '002015:' SUB15(:PUSTAT)
746 '002016:' COUNT3 = S
747 '002017:' A = 0
748 '002018:' L3: " TE HERHALEN ACTIE
749 '002019:' A + 1
750 '002020:' REP3P(:L3)
751 '002021:' S = COUNT3 " COUNT3 = -0
752 '002022:' SUB15(:PUSTAT) " PONS STATUS
753 '002023:' S = 12
754 '002024:' COUNT6 = S
755 '002025:' A = 0
756 '002026:' L4:
757 '002027:' A + 1
758 '002028:' REP6E(:L4)
759 '002029:' S = COUNT6 " COUNT6 = -1
760 '002030:' SUB15(:PUSTAT) " PONS STATUS
761 '002031:' COUNT0 = A
762 '002032:' S = 0
763 '002033:' L5:
764 '002034:' REP0Z(:L6) " ALLEEN ALS COUNT0 = 0
765 '002035:' S + 1
766 '002036:' GOTO(:L5)
767 '002037:' L6:
768 '002038:' A = COUNT0 " PONS STATUS
769 '002039:' SUB15(:PUSTAT)
770 '002040:' " TEST SUBN
771 " JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
772 '002041:' S = 0
773 '002042:' SUB(:L7)
774 '002043:' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
775 '002044:' L7:
776 '002045:' A = LINK
777 '002046:' SUB15(:PUSTAT) " PONS STATUS
778 '002047:' S = 0, Z " C = 0
779 '002048:' N, SUB3(:L8) " NIET DOEN
780 '002049:' S + 1 " WEL DOEN
781 '002050:' L8:
782 '002051:' SUB15(:PUSTAT) " PONS STATUS
783 '002052:' Y, SUB5(:L9) " WEL DOEN
784 '002053:' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
785 '002054:' L9:
786 '002055:' A = LINK5
787 '002056:' SUB15(:PUSTAT) " PONS STATUS
788 '002057:' A = SGN BIT
789 '002058:' A + A " OF: = 1
790 '002059:' U, SUB8(:L10) " DOEN
791 '002060:' SUB(:CRASH1) " MAG NIET UITGEVOERD WORDEN
792 '002061:' L10:
793 '002062:' A = LINK8
794 '002063:' SUB15(:PUSTAT) " PONS STATUS
795 '002064:' " OM GOTOR, JUMPR EN SUBC TE KUNNEN TESTEN,
796 " MOETEN WE EERST DE GEDRAGINGEN VAN IV, OV EN BT TESTEN
797 " TEST OV, IV, BT
798 " JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
799
800
801 '002065:' '502400000'

```

```

802 '002066': '760061000' OVON
803 '002067': '577001001' SUB15(:PUSAT) " PONS STATUS
804 '002070': '660061000' OVOFF
805 '002071': '577001001' SUB15(:PUSAT) " PONS STATUS
806 '002072': '120002746' S = CRASH
807 '002073': '160000030' CH = S
808 '002074': '760060000' IVON
809 '002075': '660060000' IVOFF
810 '002076': '120002745' S = SUB15PUSAT
811 '002077': '160000030' CH = S
812 '002100': '760060000' IVON
813 '002101': '102000001' S + 1
814 " INGREEP KOMT HIER
815 '002102': '660071005' IFOFF(IP)
816 '002103': '760072022' LVIFON(PUNCH)
817 '002104': '660072022' LVIFOFF(PUNCH)
818 '002105': '420002740' B = ZERO
819 '002106': '766075400' MEMPROT
820 "
821 " TEST SUBC
822 '002107': '502400000' JUMPR(0)
823 '002110': '420002766' B = STACK
824 '002111': '572476400' SUBC(:MT)
825 '002112': '120000075' S = B
826 '002113': '026075377' A = MC(-1)
827 '002114': '577001001' SUB15(:PUSAT)
828 '002115': '022001212' A = DUMMY
829 '002116': '066075400' MC = A
830 '002117': '566475377' SUBC(MC[-1])
831 "
832 "
833 "
834 "
835 '002120': '020002766' A = STACK
836 '002121': '120000075' S = B
837 '002122': '577001001' SUB15(:PUSAT)
838 "
839 '002123': '020002721' A = OV BIT
840 '002124': '120002753' S = SUBC DUMMY
841 '002125': '564001100' SUB2(:SUBC TEST)
842 '002126': '020002721' A = OV BIT
843 '002127': '120002754' S = SUBC DUMMY ADDR
844 '002130': '564001100' SUB2(:SUBC TEST)
845 '002131': '020002721' A = OV BIT
846 '002132': '120002755' S = SUBC GATE
847 '002133': '564001100' SUB2(:SUBC TEST)
848 '002134': '020002721' A = OV BIT
849 '002135': '120002756' S = SUBC GATE ADDR
850 '002136': '564001100' SUB2(:SUBC TEST)
851 "
852 " GOTOR IS VOOR HET GROOTSTE DEEL AL GETEST
853 '002137': '120000076' S = T
854 '002140': '370002723' S = -SGN BIT
855 '002141': '102000004' S + 4
856 '002142': '520400074' GOTOR(S)
857 '002143': '560001073' SUB(:CRASH1)
858 "
859 '002144': '577001001' SUB15(:PUSAT)
860 "
861 " TEST BT, HET VERLATEN VAN DE BESTUURSTOESTAND

```

```

802 " VOOR HET PONSSEN VAN RESULTATEN WORDT EEN FOUT-ADRES-INGREEP GEBRUIKT
803 JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
804 S = SUB15PUSTAT
805 WA = S
806 " FOUT-ADRES-INGREEP PLAATS
807 S = D25
808 " BT = U, IV = OV = 1
809 JUMPR(S)
810 S = T
811 " BEKIJK RESULTAAT
812 GOTO(MT[-1])
813 JUMPR(0)
814 " INGREEP, OM STATUS TE PONSSEN
815 GOTO(MT[-1])
816 B = :STACK
817 S = T
818 " INGREEP, OM STATUS TE PONSSEN
819 SUBC(:PUNCH GATE)
820 S = T
821 " BEKIJK TOESTAND IN DE WERELD
822 " PROTECTIEPOORT, :STAT, KOMT TERUG MET BT = 0
823 " BEKIJK TOESTAND IN DE WERELD
824 SUBC(:PUNCH GATE ADDR)
825 S = T
826 " PROTECTIEPOORT, :STAT, KOMT TERUG MET BT = 0
827 " BEKIJK TOESTAND IN DE WERELD
828 GOTO(MT[-1])
829 SUBCD(:MT)
830 " INGREEP, OM STATUS TE PONSSEN
831 S = MCI-1]
832 " LINK
833 GOTO(MT[-1])
834 " INGREEP, OM STATUS TE PONSSEN
835 " BT = 1, IV = 0, OV = 1
836 SUBC(:OPEN GATE)
837 SUB15(:PUSTAT)
838 IVON
839 " PONS STATUS
840 " IV = 1
841 SUB15(:PUSTAT)
842 " PONS STATUS
843 SUBCD(:MT)
844 " IV = U.
845 S = T
846 " MEETEEN KIJKEN
847 A = MCI-1]
848 " PONS STATUS
849 SUB15(:PUSTAT)
850
851 " TEST OV TIJDENS FOUTINGREEP EN
852 " TEKENCONSISTENTIE VAN T (D26 = D18) TIJDENS SUBC
853 OVON
854 S = READ T
855 " S = T
856 WA = S
857 " CI = 1
858 A = -1, P
859 SUBC(MA)
860 " ADRESINGREEP, IV = 0, OV = 0
861 SUB15(:PUSTAT)
862 " PONS STATUS
863
864 " TEST DO, DOS
865 JUMPR(0)
866 " ALLE EENBIT-REGISTERS WORDEN 0
867 DO(AMT255)
868 " A = INT(255)
869 DO(SMT)
870 " S = MT
871 SUB15(:PUSTAT)
872 " PONS STATUS
873 B = :STACK
874 " INSTRUCTIE OP STAPEL
875 S = SUB15PUSTAT
876 MC = S
877 " OF = 1
878 A = SGN BIT
879 A + A
880 U,DO(MCI-1])
881 A = B
882 " PONS STATUS
883 SUB15(:PUSTAT)
884 " A = -60, C = 1
885 A = -IS, P
886 " ALTIJD GOED
887 S = LP0
888 " S = :S, DO(S), A+S, A := -0
889 N,DOS(S)
890 " PONS STATUS
891 SUB15(:PUSTAT)
892
893 " TEST CLP, INT
894 JUMPR(0)
895 " ALLE EENBIT-REGISTERS WORDEN 0
896 A = LP1
897 " LP = 1
898 CLP
899 " CI = 1
900
901 "002206:" '502400000'
902 "002207:" '570402751'
903 "002208:" '570402752'
904 "002209:" '570402752'
905 "002210:" '570402752'
906 "002211:" '570402752'
907 "002212:" '570402752'
908 "002213:" '570402752'
909 "002214:" '570402752'
910 "002215:" '570402752'
911 "002216:" '570402752'
912 "002217:" '570402752'
913 "002218:" '570402752'
914 "002219:" '570402752'
915 "002220:" '570402752'
916 "002221:" '570402752'
917 "002222:" '570402752'
918 "002223:" '570402752'
919 "002224:" '570402752'
920 "002225:" '570402752'
921 "002226:" '570402752'
922 "002227:" '570402752'
923 "002228:" '570402752'
924 "002229:" '570402752'
925 "002230:" '570402752'
926 "002231:" '570402752'
927 "002232:" '570402752'
928 "002233:" '570402752'
929 "002234:" '570402752'
930 "002235:" '570402752'
931 "002236:" '570402752'
932 "002237:" '570402752'
933 "002238:" '570402752'
934 "002239:" '570402752'
935 "002240:" '570402752'
936 "002241:" '570402752'
937 "002242:" '570402752'
938 "002243:" '570402752'
939 "002244:" '570402752'
940 "002245:" '570402752'
941 "002246:" '570402752'
942 "002247:" '570402752'
943 "002248:" '570402752'
944 "002249:" '570402752'
945 "002250:" '570402752'
946 "002251:" '570402752'
947 "002252:" '570402752'
948 "002253:" '570402752'
949 "002254:" '570402752'
950 "002255:" '570402752'
951 "002256:" '570402752'
952 "002257:" '570402752'
953 "002258:" '570402752'
954 "002259:" '570402752'
955 "002260:" '570402752'
956 "002261:" '570402752'
957 "002262:" '570402752'
958 "002263:" '570402752'
959 "002264:" '570402752'
960 "002265:" '570402752'
961 "002266:" '570402752'
962 "002267:" '570402752'
963 "002268:" '570402752'
964 "002269:" '570402752'
965 "002270:" '570402752'
966 "002271:" '570402752'
967 "002272:" '570402752'
968 "002273:" '570402752'
969 "002274:" '570402752'
970 "002275:" '570402752'
971 "002276:" '570402752'
972 "002277:" '570402752'
973 "002278:" '570402752'
974 "002279:" '570402752'
975 "002280:" '570402752'
976 "002281:" '570402752'
977 "002282:" '570402752'
978 "002283:" '570402752'
979 "002284:" '570402752'
980 "002285:" '570402752'
981 "002286:" '570402752'
982 "002287:" '570402752'
983 "002288:" '570402752'
984 "002289:" '570402752'
985 "002290:" '570402752'
986 "002291:" '570402752'
987 "002292:" '570402752'
988 "002293:" '570402752'
989 "002294:" '570402752'
990 "002295:" '570402752'
991 "002296:" '570402752'
992 "002297:" '570402752'
993 "002298:" '570402752'
994 "002299:" '570402752'
995 "002300:" '570402752'
996 "002301:" '570402752'
997 "002302:" '570402752'
998 "002303:" '570402752'
999 "002304:" '570402752'
1000 "002305:" '570402752'

```

```

922 '002231' '577001001' SUB15(:PUSTAT) " PONS STATUS
923 '002232' '120002724' S = LP0 " LP:= 0
924 '002233' '600063000' CLP " C:= 0
925 '002234' '577001001' SUB15(:PUSTAT) " PONS STATUS
926 '002235' '622000001' G = 1
927 '002236' '712000002' F / 2
928 '002237' '726175400' MC = G " NINT:= 1
929 '002240' '577001001' SUB15(:PUSTAT) " PONS STATUS
930 '002241' '660064000' INT " NINT:= 0, C:= 1
931 '002242' '577001001' SUB15(:PUSTAT) " PONS STATUS
932 '002243' '660164000' U, INT " NINT:= 0, C:= 1
933 '002244' '577001001' SUB15(:PUSTAT) " PONS STATUS
934
935 " TEST SNELLE REGISTERTRANSPORTEN
936 '002245' '022000000' A = 0
937 '002246' '120002725' S = LP1
938 '002247' '660000440' '660 000 440' " TRAS
939 '002250' '577001001' SUB15(:PUSTAT) " PONS STATUS
940 '002251' '022000000' A = 0
941 '002252' '660100440' '660 100 440' " U, TRAS
942 '002253' '577001001' SUB15(:PUSTAT) " PONS STATUS
943 '002254' '033000001' A = -1, Z " C = 1
944 '002255' '670500402' '670 500 402' " U,TRBNA,P , C:= 0
945 '002256' '120000075' S = B
946 '002257' '577001001' SUB15(:PUSTAT) " PONS STATUS
947 '002260' '671400501' '671 400 501' " TRSNB,E , C:= 1
948 '002261' '577001001' SUB15(:PUSTAT) " PONS STATUS
949

```

" TEST VAN DE TRANSPUT BESTUUR- EN LEESOPDRACHTEN
 " DE OPDRACHTEN AFON EN AFOFF WORDEN NIET GETEST

```

950
951
952 '002262' '660060000' IVOFF
953 '002263' '120002757' S = IFONU " WIJZIG FLIPFLOPS
954 '002264' '564001163' SUB2(:DO FF)
955 '002265' '422000000' B = 0
956 '002266' '660075000' IFA(0)
957 '002267' '764075001' IFA(B + 1)
958 '002270' '577001001' SUB15(:PUSTAT) " PONS STATUS
959 '002271' '120002740' S = IFOFF0
960 '002272' '564001163' SUB2(:DO FF) " WIJZIG FLIPFLOPS
961 '002273' '422000000' B = 0
962 '002274' '664075000' IFA(B)
963 '002275' '760075001' IFA(1)
964 '002276' '577001001' SUB15(:PUSTAT) " PONS STATUS
965 '002277' '120002762' S = LVIFOFF0 " WIJZIG FLIPFLOPS
966 '002300' '564001163' SUB2(:DO FF)
967 '002301' '422000000' B = 0
968 '002302' '660076000' LVIFA(0)
969 '002303' '764076001' LVIFA(B + 1)
970 '002304' '577001001' SUB15(:PUSTAT) " PONS STATUS
971 '002305' '120002761' S = LVIFON0
972 '002306' '564001163' SUB2(:DO FF) " WIJZIG FLIPFLOPS
973 '002307' '422000000' B = 0
974 '002310' '664076000' LVIFA(B)
975 '002311' '760076001' LVIFA(1)
976 '002312' '577001001' SUB15(:PUSTAT) " PONS STATUS
977
978 '002313' '120002757' S = IFONU
979 '002314' '564001163' SUB2(:DO FF) " WIJZIG FLIPFLOPS
980 '002315' '422000000' B = 0
981 '002316' '020002724' A = LP0 " D25 = 1, D12 = 0

```

```

982 '002317': '660075100' IFAC(0)
983 '002320': '120002725' S = LP1
984 '002321': '764075101' IFSC(B + 1)
985 '002322': '577001001' SUB15(:PUSTAT)
986 '002323': '120002760' S = IFOFF0
987 '002324': '564001163' SUB2(:DO FF)
988 '002325': '422000000' B = 0
989 '002326': '020002724' A = LP0
990 '002327': '664075100' IFAC(B)
991 '002330': '120002725' S = LP1
992 '002331': '764075101' IFSC(1)
993 '002332': '577001001' SUB15(:PUSTAT)
994 '002333': '120002762' S = LVIFOFF0
995 '002334': '564001163' SUB2(:DO FF)
996 '002335': '422000000' B = 0
997 '002336': '020002724' A = LP0
998 '002337': '660075100' LVIFAC(0)
999 '002340': '120002725' S = LP1
1000 '002341': '764075101' LVIFSC(B + 1)
1001 '002342': '577001001' SUB15(:PUSTAT)
1002 '002343': '120002761' S = LVIFON0
1003 '002344': '564001163' SUB2(:DO FF)
1004 '002345': '422000000' B = 0
1005 '002346': '020002724' A = LP0
1006 '002347': '664075100' LVIFAC(B)
1007 '002350': '120002725' S = LP1
1008 '002351': '760075101' LVIFSC(1)
1009 '002352': '577001001' SUB15(:PUSTAT)
1010
1011 '002353': '023000000' A = 0, Z
1012 '002354': '661175001' U, IFA(1), Z
1013 '002355': '577001001' SUB15(:PUSTAT)
1014 '002356': '131002723' S = -SGN BIT, Z
1015 '002357': '761175100' U, IFSC(U), Z
1016 '002360': '577001001' SUB15(:PUSTAT)
1017 '002361': '131000074' S = -S, Z
1018 '002362': '761175101' U, LVIFSC(1), Z
1019 '002363': '577001001' SUB15(:PUSTAT)
1020 '002364': '120002745' S = SUB15PUSTAT
1021 '002365': '1600000030' CH = S
1022 '002366': '760060000' IVON
1023 '002367': '102000001' S + 1
1024 '002370': '760071044' IFON(A DEV)
1025
1026 '002371': '660071044' IFOFF(A DEV)
1027
1028
1029 '002372': '502400000' " TEST DYNAMISCHE STUPTOESTANDEN
1030 '002373': '120002763' JUMPR(0)
1031 '002374': '1600000030' S = SUB14
1032 '002375': '760071005' CH = S
1033 '002376': '422002766' IFON(IP)
1034 '002377': '122000000' B = ISTACK
1035 '002400': '562401200' S = 0
1036
1037 '002401': " L14:
1038 '002401': '020002766' A = STACK
1039 '002402': '1200000010' S = LINK
1040 '002403': '577001001' SUB15(:PUSTAT)
1041 '002404': '120002764' S = SUB15

```

```

1042 '002405': '160000030' CH = S
1043 '002406': '120002737' S = DYST NBT
1044 '002407': '112002412' S = :L15[1]
1045 '002410': '500400074' JUMPR(S)
1046
1047 '002411':
1048 '002411': '020000010' A = LINK
1049 '002412': '577001001' SUB15(:PUSTAT)
1050 '002413': '660071005' IFOFF(IP)
1051
1052
1053 '002414': '502400000' " TEST ITVON EN PARITEITSINGREEP
1054 '002415': '120002746' JUMPR(0)
1055 '002416': '160000030' S = CRASH
1056 '002417': '160000032' CH = S
1057 '002420': '120002745' WA = S
1058 '002421': '160000031' S = SUB15PUSTAT
1059 '002422': '760061000' WP = S
1060 '002423': '422002766' OVON
1061 '002424': '760062000' B = :STACK
1062 '002425': '166075400' ITVON
1063 '002426': '760062000' MC = S
1064 '002427': '166075377' ITVON
1065 '002430': '020000075' S = -MC[-1]
1066 '002431': '577001001' A = B
1067 '002432': '166075400' SUB15(:PUSTAT)
1068 '002433': '760062000' MC = S
1069 '002434': '166075377' ITVON
1070 '002435': '020000075' S = -MC[-1]
1071 '002436': '577001001' A = B
1072 '002437': '760062000' SUB15(:PUSTAT)
1073 '002440': '166075400' ITVON
1074 '002441': '760062000' MC = S
1075 '002442': '166075377' S = -MC[-1]
1076 '002443': '020000075' A = B
1077 '002444': '577001001' SUB15(:PUSTAT)
1078 '002445': '020000001' S + 1
1079 '002446': '166075400' MC = S
1080 '002447': '760062000' ITVON
1081 '002450': '166075377' S + MC[-1]
1082 '002451': '020000075' A = B
1083 '002452': '577001001' SUB15(:PUSTAT)
1084 '002453': '120002744' S = MCS
1085 '002454': '577001001' DO(S)
1086 '002455': '760062000' ITVON
1087 '002456': '576475377' DO(MC[-1])
1088
1089 '002457': '576475377' DO(MC[-1])
1090 '002460': '020000075' A = B
1091 '002461': '577001001' SUB15(:PUSTAT)
1092
1093 '002462': '160003060' REST = S
1094 '002463': '760062000' ITVON
1095 '002464': '160103060' PLUS(R)
1096 '002465': '620103060' G = REST
1097 '002466': '120002724' S = LPO
1098 '002467': '760062000' ITVON
1099 '002470': '160003060' REST = S
1100 '002471': '120402740' S = ZERO, P
1101 '002472': '760062000' ITVON

```

```
1102 '002473:' '120003060' S = REST LP:= 1, C:= 1
1103 '002474:' '577001001' SUB15(:PUSAT) " PONS STATUS
1104 '002475:' '120402720' S = ONE, P " LP:= 0, C:= 0
1105 '002476:' '760062000' ITVON
1106 '002477:' '120003060' S = REST LP:= 1, C:= 1
1107 '002478:' '577001001' SUB15(:PUSAT) " PONS STATUS
1108 '002479:' '120402740' S = ZERO, P " LP:= 1, C:= 0
1109 '002480:' '760062000' ITVON
1110 '002481:' '100003060' S + REST " LP:= 1, C:= 0
1111 '002482:' '577001001' SUB15(:PUSAT) " PONS STATUS
1112
1113
1114
1115 '002483:' '122000407' S = :DUMMY GATE
1116 '002484:' '760062000' ITVON
1117 '002485:' '160000474' BAD ADDR = S
1118 '002486:' '512400000' JUMPR(-0)
1119 '002487:' '422002766' B = :STACK " BT:=0
1120 '002488:' '560400474' SUBC(BAD ADDR)
1121
1122
1123 '002489:' '562400410' SUBC(:OPEN GATE)
1124
1125 '002490:' '122000000' S = 0
1126 '002491:' '760062000' ITVON
1127 '002492:' '160000030' CH = S
1128 '002493:' '760071005' IFON(IP)
1129 '002494:' '760060000' IVON
1130 '002495:' '102000001' S + 1
1131 " CHARONINGREEP, GEEFT PARITEITSFOOT, PARITEITSINGREEP MET OV = 0
1132 '002496:' '660071005' IFOFF(IP) " KAN NOG NET
1133
1134
1135 " TEST PROTECTIEMECHANISME
1136 '002497:' '502400000' JUMPR(0) " ALLE EENBIT-REGISTERS WORDEN 0
1137 '002498:' '422002741' B = :PROTW " DP = 000000011111111,
1138 '002499:' '766075400' MEMPROT " GP = 1000010001000000, VOOR EERSTE HELFT M = 0
1139 '002500:' '120000075' S = B
1140 '002501:' '577001001' SUB15(:PUSAT) " PONS STATUS
1141 '002502:' '120002746' S = CRASH
1142 '002503:' '160000032' CH = S
1143 '002504:' '160000031' WA = S
1144 '002505:' '120002745' WP = S
1145 '002506:' '160000033' S = SUB15PUSAT
1146 '002507:' '120002724' PR = S
1147 '002508:' '160004777' S = LP0
1148 '002509:' '102000001' M(: 4 777) = S
1149 '002510:' '160005000' S + 1
1150 '002511:' '102000001' M(: 5 000) = S
1151 '002512:' '120001000' S + 1
1152 '002513:' '102000001' M(:10 000) = S
1153 '002514:' '160010777' S + 1
1154 '002515:' '102000001' M(:10 777) = S
1155 '002516:' '160011000' S + 1
1156 '002517:' '102000001' M(:11 000) = S
1157 '002518:' '160011777' S + 1
1158 '002519:' '102000001' M(:11 777) = S
1159 '002520:' '160012000' S + 1
1160 '002521:' '500402722' M(:12 000) = S
1161 '002522:' '120001000' JUMPR(D25) " BT:=40, IV:= OV:= 1
1162 " S = M(:10000)
1163
```

12:12/2-10/ 0 2220U.0 GRUNE

```

1162 '002556:' '120004777' S = M('14777') " INGREEP
1163 '002557:' '120005000' S = M('5000')
1164 '002560:' '160007777' M('7777') = S
1165 '002561:' '160010000' M('10000') = S
1166 '002562:' '120010000' S = M('10000')
1167 '002563:' '120010777' S = M('10777')
1168 '002564:' '120011000' S = M('11000')
1169 '002565:' '120011777' S = M('11777')
1170 '002566:' '120012000' S = M('12000')
1171 '002567:' '160000077' D = S
1172 '002570:' '122000001' S = 1
1173 '002571:' '160000007' COUNT = S
1174 '002572:' L12:
1175 '002572:' REPP('L12) " COUNT-PLAATSEN ZIJN VRIJ
1176 '002573:' SUB7('L13) " LINK0 - LINK7
1177 '002574:' SUB('CRASH1) " MAG NIET UITGEVOERD WORDEN
1178 '002575:' L13:
1179 '002575:' SUB8('CRASH1) " MAG NIET GOED GAAN
1180
1181 '002576:' B = :STACK
1182 '002577:' SUBC(PUNCH GATE ADDR) " PROTECTIEPOORT
1183 '002600:' B = '5000'
1184 '002601:' SUBC('CRASH1) " DP = 0, GP = 1
1185 '002602:' B = '10000'
1186 '002603:' SUBC('CRASH1) " DP = 1, GP = 0
1187 '002604:' B = :STACK
1188 '002605:' SUBC('OPEN GATE) " BT: 1
1189
1190 '002606:' S = 0
1191 '002607:' SUB8('SET PROT) " ALLE DP'S EN GP'S WORDEN 0
1192 '002610:' A = D25 " IN NIET-BESTUURTOESTAND
1193 '002611:' S = SUBC DUMMY " SUBC, DIRECT
1194 '002612:' SUB2('SUBC TEST)
1195 '002613:' A = D25 " IN NIET-BESTUURTOESTAND
1196 '002614:' S = SUBC DUMMY ADDR " SUBC, INHOUDELIJK
1197 '002615:' SUB2('SUBC TEST)
1198 '002616:' A = D25 " IN NIET-BESTUURTOESTAND
1199 '002617:' S = SUBC GATE " PROTECTIEPOORT, DIRECT
1200 '002620:' SUB2('SUBC TEST)
1201 '002621:' A = D25 " IN NIET-BESTUURTOESTAND
1202 '002622:' S = SUBC GATE ADDR " PROTECTIEPOORT, INHOUDELIJK
1203 '002623:' SUB2('SUBC TEST)
1204
1205 '002624:' S = DP ON
1206 '002625:' SUB8('SET PROT) " ALLE DP'S WORDEN 1, ALLE GP'S 0
1207 '002626:' A = D25 " IN NIET-BESTUURTOESTAND
1208 '002627:' S = SUBC DUMMY " SUBC, DIRECT
1209 '002630:' SUB2('SUBC TEST)
1210 '002631:' A = D25 " IN NIET-BESTUURTOESTAND
1211 '002632:' S = SUBC DUMMY ADDR " SUBC, INHOUDELIJK
1212 '002633:' SUB2('SUBC TEST)
1213 '002634:' A = D25 " IN NIET-BESTUURTOESTAND
1214 '002635:' S = SUBC GATE " PROTECTIEPOORT, DIRECT
1215 '002636:' SUB2('SUBC TEST)
1216 '002637:' A = D25 " IN NIET-BESTUURTOESTAND
1217 '002640:' S = SUBC GATE ADDR " PROTECTIEPOORT, INHOUDELIJK
1218 '002641:' SUB2('SUBC TEST)
1219
1220 " TEST ADRESINGREEP
1221 '002642:' '502400000' " ALLE EENBIT-REGISTERS WORDEN 0
JUMPR(0)

```

15-2/2-10/ D 2220U.0 GRUNE

25

```

1222 '002643' '120002746' S = CRASH
1223 '002644' '160000030' CH = S
1224 '002645' '160000031' WP = S
1225 '002646' '160000033' PR = S
1226 '002647' '120002745' S = SUB15PUSTAT
1227 '002650' '160000032' WA = S
1228 '002651' '032000001' A = -1
1229 '002652' '760061000' OVON
1230 '002653' '062073400' A = :MA
1231 '002654' '062073401' A = :MA[1]
1232 '002655' '577001001' SUB15(:PUSTAT)
1233 '002656' '020002737' A = DYST NBT
1234 '002657' '062073400' A = :MA
1235 '002660' '577001001' SUB15(:PUSTAT)
1236 '002661' '062073401' A = :MA[1]
1237
1238 '002662' '020002736' A = EOM1
1239 '002663' '062073400' A = :MA
1240 '002664' '577001001' SUB15(:PUSTAT)
1241 '002665' '026073400' A = :MA
1242 '002666' '520002723' GOTO(SGN BIT)
1243 '002667' '520002736' GOTO(EOM1)
1244 '002670' '020002736' A = EOM1
1245 '002671' '066073400' MA = A
1246 '002672' '422000000' B = 0
1247 '002673' '572476400' SUBC(:MT)
1248 '002674' '120000075' S = B
1249 '002675' '026075377' A = MC(-1)
1250 '002676' '577001001' SUB15(:PUSTAT)
1251 '002677' '120002737' S = DYST NBT
1252 '002700' '112000002' S = 2
1253 '002701' '520000074' GOTO(S)
1254
1255 '002702' '422002743' B = :WA PROT W
1256 '002703' '766075400' MEMPROT
1257 '002704' '432002740' B = -ZERO
1258 '002705' '766075400' MEMPROT
1259 '002706' '120002765' S = SUB16
1260 '002707' '160000032' WA = S
1261 '002710' '120002750' S = READ T
1262 '002711' '020002735' A = EOM
1263 '002712' '166073377' MA[-1] = S
1264 '002713' '166073400' MA = S
1265 '002714' '536073377' GOTO(:MA[-1])
1266 '002715' L16:
1267 '002715' A = LINK
1268 '002716' SUB15(:PUSTAT)
1269
1270 '002717' '577001035' SUB15(:TERM)
1271

```

" GEEN CHARONINGREEP"

" GEEN PARITEITSINGREEP"

" GEEN PROTECTIEINGREEP"

" WAS 0, WORDT 1

" METEEN PROBEREN

" MAG, A:= +0

" PONS STATUS

" A:= A-STER = '777 777', BESTAAND

" PONS STATUS

" A-STER + 1 = '1.000.000', NIET BESTAAND,

" DUS INGREEP

" WEL BESTAAND, MAG DUS

" PONS STATUS

" NIET AANWEZIG

" NIET AANWEZIG

" MA = A

" B = 0

" SUBC(:MT)

" S = B

" A = MC(-1)

" SUB15(:PUSTAT)

" S = DYST NBT

" S = 2

" GOTO(S)

" MET BT

" VERWIJST NAAR EERSTE NIET-AANWEZIGE GEHEUGENKAST

" STAAT NIET IN 'T BOEK, GEEFT WEL INGREEP

" IDEN

" EEN NA LAATSTE EN

" LAATSTE GEHEUGENPLAATS BEVATTEN 'S = T'

" OT := EOM - 1

" OT TIJDENS DE INGREEP

" PONS STATUS

" EINDE X8 TEST

15/2/2-10/ D 2220U.0 GRUNE

" CONSTANTEN

12/5
12/6

12/7 '002720' ONE;
12/8 '000000001' +1
12/9 '002721' OV BIT;
1280 '002721' '100 000 000'
1281 '002722' D25;
1282 '002722' '200 000 000'
1283 '002723' SGN BIT;
1284 '002723' '400 000 000'
1285 '002724' LP0;
1286 '002724' '252 525 252'
1287 '002725' LP1;
1288 '002725' '525 252 525'
1289 '002726' FP0;
1290 '002726' +.6363636363636
1291 '002727' '773105056' " 7/11
1292 '002730' '213505643' " -11/7
1293 '002731' FP2;
1294 '002732' -1.571428571429
1295 '002733' '733333332'
1296 '002733' D13T0;
1297 '002734' '37 777'
1298 '002734' D14T0;
1299 '002735' '77 777'
1300 '002735' D10;
1301 '002736' '1 000 000'
1302 '002736' EOM;
1303 '002737' (MEM LENGTH - 1)
1304 '002737' EOM1;
1305 '002737' (MEM LENGTH)
1306 '002737' DYST NBT;
1307 '002737' '200 777 777'

" PROTECTIEWOORDEN

1308 '002740' ZERO: +0
1309 '002741' PROTW;
1310 '002741' '140 000 104'
1311 '002742' DP ON;
1312 '002742' '060 000 000'
1313 '002743' WA PROTW;
1314 '002743' (MEM LENGTH)
1315 '002743' '002000000'

" INSTRUCTIES

1316 '002744' MCS;
1317 '002744' MC = S
1318 '002744' SUB15PSTAT;
1319 '002745' SUB15(:PSTAT)
1320 '002745' CRASH;
1321 '002746' SUB(:CRASH1)
1322 '002746' GOTOL1;
1323 '002747' GOTOL1;
1324 '002747' GOTOL1;
1325 '002747' READ T;
1326 '002750' S = T
1327 '002750' AMT255;
1328 '002751' A = :MT[255]
1329 '002751' SMT;
1330 '002752' S = MT
1331 '002752' SUBC DUMMY;
1332 '002753'

" MAG NOOIT VOORKOMEN

[illegible]

1212/2-10/

0 2220.0 GRUNE

31

17

1
03902
16034
5786

220
11686944
44370
96322

17.1.72- 93 D 2220U.53 DICKGRUNE

GLHEU-GEBOUW VAN 00000 TOT 17777

	0	1	2	3	4	5	6	7
LOC 000000 - 000007	100 002674	302 001120		777 777777		000 000011	777 777776	777 777777
LOC 000010 - 000017	120 200000		100 002642			000 002054		322 002574
LOC 000020 - 000027	120 002626							000 777777
LOC 000030 - 000037	560 001073	560 001073	560 002715	560 001073	560 001073			
LOC 000040 - 000047	302 001120							
LOC 000050 - 000057								
LOC 000060 - 000067								
LOC 000070 - 000077								
LOC 000080 - 000087								
LOC 000090 - 000097								
LOC 000100 - 000107	302 001120	577 001001	526 075377		022 025252	577 001001	526 475377	526 475377
LOC 000110 - 000117	022 052525					000 000407	000 000404	302 001120
LOC 000120 - 000127	302 001120							
LOC 000130 - 000137	000 000000	060 001063	160 001064	460 001065	422 000003	460 001067	574 441067	432 000004
LOC 000140 - 000147	770 000125	362 000077	160 001066	120 001066	660 063000	102 300200	121 101000	522 301044
LOC 000150 - 000157	640 000002	760 071022	403 000001	122 300000	760 300107	522 300102	422 000001	471 101067
LOC 000160 - 000167	522 301006	020 001063	120 001064	420 001065	520 400027	120 002734	112 000001	160 000027
LOC 000170 - 000177	122 000177	432 000001	470 001067	522 001012	160 001062	122 001060	160 000210	160 000211
LOC 000180 - 000187	120 001061	160 000212	760 070022	120 000211	771 000062	512 200003	522 001022	000 000000
LOC 000190 - 000197	000 001060	001 001061		120 200000	120 177777	777 775037	000 000177	777 777777
LOC 000200 - 000207	120 000027	120 001064		660 061000	020 002722	261 000076	520 202723	512 000001
LOC 000210 - 000217	160 001157	260 001136	060 001160	620 000032	720 001161	620 001134	720 000032	022 000020
LOC 000220 - 000227	060 000007	622 001136	602 000001	426 072400	022 000777	120 001160	500 400074	570 401157
LOC 000230 - 000237	032 000777	526 076377	556 401112	620 001161	720 000032	520 400012	260 002725	502 000001
LOC 000240 - 000247	260 002724	120 000075	577 001001	522 001122	522 001130	522 001126	377 000000	000 000000
LOC 000250 - 000257	000 000001	000 000000	000 000077	000 000377	000 000400	000 000477	000 000500	000 100000
LOC 000260 - 000267	000 177777	000 200000	001 000000	100 000000	377 177777	400 000000	777 777777	560 400475
LOC 000270 - 000277	200 000000	560 001073	577 001001	022 000003	060 000007	622 001174	422 000023	602 000001
LOC 000280 - 000287	020 000074	006 072400	570 400073	556 401167	020 000012	000 000003	004 000000	004 000023
LOC 000290 - 000297	100 002737	520 400074	030 001211	422 000074	766 075400	102 020000	003 020000	522 301203
LOC 000300 - 000307	520 400020	000 200000	526 475377	000 001212	660 072022	502 000000	022 000000	132 000000
LOC 000310 - 000317	577 001001	120 001772	030 001772	577 001001	422 001773	124 040000	034 037776	577 001001
LOC 000320 - 000327	622 001772	126 072007	026 072777	577 001001	022 001774	126 073405	577 001001	122 001400
LOC 000330 - 000337	026 074777	577 001001	126 075400	026 075400	577 001001	026 076400	126 076400	577 001001
LOC 000340 - 000347	460 000077	026 077400	126 077400	577 001001	422 002766	460 000077	022 003060	064 040000
LOC 000350 - 000357	466 073400	002 000001	064 040071	726 173412	026 000400	126 071412	577 001001	022 000012
LOC 000360 - 000367	066 077401	466 073366	126 001366	577 001001	422 003060	076 075400	166 075400	476 075400
LOC 000370 - 000377	121 063060	020 003061	577 001001	120 003062	020 000075	577 001001	422 000076	024 040000
LOC 000380 - 000387	124 037774	577 001001	422 002766	162 075000	020 000075	577 001001	072 075777	120 000075
LOC 000390 - 000397	577 001001	502 400000	120 002725	577 001001	120 002724	577 001001	130 002725	577 001001
LOC 000400 - 000407	130 002724	577 001001	122 400000	577 001001	132 400000	577 001001	123 000000	577 001001
LOC 000410 - 000417	133 000000	577 001001	121 002723	577 001001	133 400000	577 001001	123 000000	577 001001
LOC 000420 - 000427	502 400000	022 000005	060 000000	032 000004	000 000000	432 000003	400 000000	120 000075
LOC 000430 - 000437	502 400000	122 002461	160 000077	120 002724	160 003060	121 002737	577 777777	577 001001
LOC 000440 - 000447	502 400000	121 002724	101 002725	020 002723	000 002723	577 001001	502 100000	130 002723
LOC 000450 - 000457	110 002723	577 001001	502 100000	130 002723	422 003060	164 040000	446 075400	120 000075
LOC 000460 - 000467	020 003060	577 001001	422 002766	142 075400	020 000075	577 001001	452 075400	062 075400
LOC 000470 - 000477	120 000075	577 001001	422 002766	464 040000	467 175400	020 002766	120 000075	577 001001
LOC 000480 - 000487	502 400000	120 002724	020 002725	340 000073	240 000074	340 000073	577 001001	030 000724
LOC 000490 - 000497	270 002725	130 000073	577 001001	502 400000	132 423400	577 001001	632 000001	120 000724
LOC 000500 - 000507	020 002725	622 000001	200 400072	577 001001	022 000000	132 000000	632 000001	200 000072
LOC 000510 - 000517	577 001001	022 000000	122 000000	303 001001	577 001001	313 000001	577 001001	502 400000
LOC 000520 - 000527	133 000001	760 000071	761 002000	577 001001	502 400000	020 002724	120 002725	422 000011
LOC 000530 - 000537	664 400000	577 001001	765 400001	577 001001	664 401002	577 001001	765 400103	577 001001
LOC 000540 - 000547	675 400004	577 001001	774 400005	577 001001	675 400106	577 001001	774 400107	577 001001
LOC 000550 - 000557	422 000001	665 400046	577 001001	764 400045	577 001001	664 400144	577 001001	765 400143
LOC 000560 - 000567	577 001001	020 002724	120 002725	674 400046	577 001001	774 400041	577 001001	675 400144
LOC 000570 - 000577	577 001001	775 400140	577 001001	502 400000	032 000000	661 000240	120 000075	577 001001
LOC 000580 - 000587	030 000075	661 000241	120 000075	577 001001	022 000000	121 002723	661 000340	577 001001

	0	1	2	3	4	5	6	7
L0C 001560 - 001567	120 000075	577 001001	032 000000	130 002723	660 400340	577 001001	120 000075	577 001001
L0C 001570 - 001577	502 400000	422 002724	626 475400	730 003060	020 003060	120 003061	577 001001	120 000075
L0C 001600 - 001607	577 001001	502 400000	022 000000	132 000000	620 400073	577 000001	032 000000	122 000000
L0C 001610 - 001617	620 400073	577 001001	502 400000	022 000000	122 000000	631 000073	577 001001	022 000000
L0C 001620 - 001627	132 000000	621 000073	577 001001	502 400000	022 000000	120 002723	621 000073	577 001001
L0C 001630 - 001637	502 400000	020 002732	122 000000	620 400073	577 001001	502 400000	030 000073	620 400073
L0C 001640 - 001647	577 001001	502 400000	020 000000	620 400073	577 001001	502 400000	422 003060	726 075400
L0C 001650 - 001657	020 003060	120 003061	577 001001	120 000075	577 001001	632 000001	020 000071	120 000072
L0C 001660 - 001667	577 001001	502 400000	022 000001	122 000001	620 000073	422 003060	726 175400	120 000075
L0C 001670 - 001677	577 001001	032 000000	620 000073	502 400000	720 103060	577 000001	722 075400	020 000071
L0C 001700 - 001707	120 000072	577 001001	120 000075	577 001001	620 002726	601 402730	020 000071	120 000072
L0C 001710 - 001717	577 001001	620 002726	611 402730	020 000711	120 000072	577 001001	620 002726	701 402730
L0C 001720 - 001727	020 000071	120 000072	577 001001	620 002726	711 402730	020 000071	120 000072	577 001001
L0C 001730 - 001737	502 400000	620 402724	601 502725	020 000071	120 000072	577 001001	620 102724	611 502725
L0C 001740 - 001747	020 000071	120 000072	577 001001	620 102724	701 502725	020 000071	120 000072	577 001001
L0C 001750 - 001757	620 102724	711 502725	020 000071	120 000072	577 001001	502 400000	120 002747	160 000000
L0C 001760 - 001767	122 000001	760 000071	520 000074	560 001073	577 001001	122 000000	500 000074	020 002723
L0C 001770 - 001777	000 000073	502 100001	560 001073	577 001001	502 100001	112 000001	577 001001	462 076402
L0C 002000 - 002007	400 002734	536 075401	560 001073	120 000075	577 001001	112 000001	160 000005	020 002723
L0C 002010 - 002017	000 000073	552 102013	560 001073	120 000005	577 001001	160 000003	022 000000	002 000001
L0C 002020 - 002027	546 402017	120 000003	577 001001	122 000014	160 000006	022 000000	002 000001	555 402026
L0C 002030 - 002037	120 000006	577 001001	060 000000	122 000000	541 002037	102 000004	522 002034	020 000000
L0C 002040 - 002047	577 001001	502 400000	122 000000	560 002045	560 001073	020 000010	577 001001	123 000000
L0C 002050 - 002057	566 302052	102 000001	577 001001	572 202055	560 001073	020 000015	577 001001	020 002723
L0C 002060 - 002067	000 000073	561 102063	560 001073	020 000020	577 001001	502 400000	760 061000	577 001001
L0C 002070 - 002077	660 061000	577 001001	120 002746	160 000030	760 060000	660 060000	120 002745	160 000030
L0C 002100 - 002107	760 060000	502 000001	660 070005	760 072022	660 072022	660 072022	766 075400	502 400000
L0C 002110 - 002117	422 002766	102 476400	120 000075	026 075377	577 001001	022 000001	066 075400	565 475377
L0C 002120 - 002127	020 002766	120 000075	577 001001	020 002721	120 002753	564 000000	020 002721	120 002754
L0C 002130 - 002137	564 000000	020 000000	120 002755	564 001001	020 002721	564 000000	120 000076	120 000076
L0C 002140 - 002147	370 002723	102 000004	520 400074	560 001073	577 001001	502 400000	120 002745	160 000032
L0C 002150 - 002157	120 002722	500 400074	120 000076	526 076377	502 400000	526 076377	422 002766	120 000076
L0C 002160 - 002167	562 400410	120 000076	560 400476	120 000076	526 076377	573 476400	126 075377	526 076377
L0C 002170 - 002177	760 061000	577 001001	760 060000	577 001001	573 476400	120 000076	026 075377	577 001001
L0C 002200 - 002207	560 061000	120 002750	160 000032	032 400001	566 473400	577 001001	502 400000	570 402751
L0C 002210 - 002217	570 402752	577 001001	422 002766	120 002745	166 075400	030 002723	000 000073	576 575377
L0C 002220 - 002227	020 000075	577 001001	032 400074	120 002724	166 075400	577 000074	502 400000	020 002725
L0C 002230 - 002237	660 063000	577 001001	120 002724	660 063000	577 001001	622 000001	712 000002	726 175400
L0C 002240 - 002247	577 001001	660 064000	577 001001	660 164000	577 001001	022 000010	120 002725	660 000440
L0C 002250 - 002257	577 001001	022 000000	660 100440	577 001001	033 000001	022 000010	120 000075	577 001001
L0C 002260 - 002267	671 400501	577 001001	660 060000	120 002757	564 001163	422 000000	660 075000	764 075001
L0C 002270 - 002277	577 001001	120 002760	564 001163	422 000000	664 075000	760 075001	577 001001	120 002762
L0C 002300 - 002307	564 001163	422 000000	660 076000	764 076001	577 001001	120 002761	564 001163	422 000000
L0C 002310 - 002317	664 076000	760 076001	577 001001	120 002757	564 001163	422 000000	020 002724	660 075100
L0C 002320 - 002327	120 002725	764 075101	577 001001	120 002760	564 001163	422 000000	020 002724	660 075100
L0C 002330 - 002337	120 002725	760 075101	577 001001	120 002762	564 001163	422 000000	020 002724	660 075100
L0C 002340 - 002347	120 002725	764 076101	577 001001	120 002761	564 001163	422 000000	020 002724	660 075100
L0C 002350 - 002357	120 002725	760 076101	577 001001	023 000000	661 175001	577 000001	131 002723	761 175100
L0C 002360 - 002367	577 001001	131 000074	761 176101	577 001001	120 002745	160 000030	760 060000	102 000001
L0C 002370 - 002377	760 071044	660 071044	502 400000	120 002763	160 000030	760 070075	422 002766	122 000000
L0C 002400 - 002407	562 401200	020 002766	120 000010	577 001001	120 002764	160 000030	120 002737	112 002412
L0C 002410 - 002417	500 400074	020 000010	577 001001	660 071005	502 400000	120 002746	160 000030	160 000032
L0C 002420 - 002427	120 002745	160 000031	760 061000	422 002766	760 062000	166 075400	760 062000	136 075377
L0C 002430 - 002437	020 000075	577 001001	166 075400	760 062000	136 075377	020 000075	577 001001	760 062000
L0C 002440 - 002447	166 075400	760 062000	116 075377	020 000075	577 001001	102 000001	166 075400	760 062000
L0C 002450 - 002457	106 075377	020 000075	577 001001	120 002744	570 400074	760 062000	576 475377	576 475377
L0C 002460 - 002467	020 000075	577 001001	160 003060	760 062000	160 103060	620 103060	120 002724	760 062000
L0C 002470 - 002477	160 003060	120 402740	760 062000	120 003060	577 001001	120 402720	760 062000	120 003060

1/11/2- 93

D 2220U.53 DICKGRUNE

37

19

57

178

178

23808244

24004730

12903

33137

4/52

444743

10. Detailtests op de EL X8.

Voor het schrijven van de X8-simulator bleek het nodig een aantal kwesties aangaande de werking van de EL X8 op te helderen. Soms ontstonden de vragen door onduidelijke uitleg of ontbrekende informatie in het Reference Manual, vaak ook rezen de vragen doordat bij het draaien van het testprogramma (zie hoofdstuk 9) de EL X8 anders reageerde dan verwacht.

Hieronder volgt een lijst, in willekeurige volgorde, van tests die gedaan werden om deze vragen te beantwoorden.

- 10.1. Volgens Reference Manual A4.9.1. zijn B-gemodificeerde schuifopdrachten over meer dan 31 plaatsen ongedefinieerd. De uitgebreide beschrijving (Reference Manual A4.9.2.) houdt in dat botweg over meer dan 31 plaatsen geschoven wordt. Het gerucht wilde dat het CRO erdoor in een "loop" raakt en de opdracht niet afmaakt. Bij tests bleek dat telkens alleen de laatste 5 bits van de schuifafstand beschouwd werden; geen van beide bovenstaande effecten werd waargenomen.
- 10.2. De tekenbit van het T-register bevat een kopie van C. De link die tijdens een subroutinesprong weggeschreven wordt, bevat echter een nul als tekenbit. Nu is het bekend, dat, wanneer de EL X8 stopt op een subroutinesprong (b.v. ten gevolge van een adresfout) het T-register, zoals dat uit de lampjes blijkt, als tekenbit een nul bevat en geen kopie van C, m.a.w. het is de link. De vraag rijst nu of in de machine tijdens een subroutinesprong T inderdaad de link bevat. De enige manier om T tijdens de sprong te zien te krijgen, is de volgende. In M[26] (de adresfoutplaats) wordt de opdracht $S = T$ geplaatst; vervolgens wordt een subroutinesprong met adresfout uitgevoerd en wel wegens B negatief. De sprong wordt dan afgebroken nog voor de link weggeschreven wordt, en inplaats ervan wordt $S = T$ gedaan. De aldus verkregen waarde van T bleek echter wel degelijk in de tekenbit een kopie van C te bevatten.
- 10.3. Het samenspel tussen LP, C en de opdracht ITVDN was onduidelijk (zie Reference Manual A6.3.6.). Het volgende blijkt het geval te zijn.
- 10.3.1. De volgende opdrachten zijn de enige die LP beïnvloeden; LP wordt gelijk aan de pariteitsbit (bij oneven pariteit) van de uit het geheugen gehaalde operand.

$R = + x$	A 'X' x	MULS(x)
$R = \overline{x}$	A '+' x	MULAS(x)
$x + R$	S 'X' x	DIVA(x)
PLUSR(x)	S '+' x	DIVAS(x)
MINR(x)		

- 10.3.2. De volgende opdrachten zijn de enige die door ITV beïnvloed worden. De werking wordt hier gegeven voor het geval dat ITV true is.

$R = + x$

Als de operand van goede pariteit is, wordt C 0, en bij een foute operand 1. In beide gevallen wordt het bitpatroon van de operand in het register gehaald, en LP gelijk gemaakt aan de pariteitsbit zoals dat uit het geheugen gekomen is.

$R + x$

Onafhankelijk van de pariteit van de operand wordt het bitpatroon van de operand bij het register opgeteld (resp. van het register afgetrokken), en LP gelijk gemaakt aan de pariteitsbit zoals dat uit het geheugen gekomen is.

$x = + R$

Het bitpatroon uit het register wordt in het geheugen geschreven, voorzien van een foute pariteitsbit.

$x + R$, PLUSR(x), MINR(x)

Als de geheugenoperand foute pariteit heeft, volgt een pariteitsingreep als OV true is, anders stopt de machine. Als de geheugenoperand goede pariteit heeft, wordt LP gelijk gemaakt aan de (goede) pariteitsbit, het rekenwerk gedaan en het resultaat naar het geheugen geschreven, voorzien van een foute pariteitsbit.

- 10.3.3. Tijdens het testen bleek dat de EL X8 bij een opdracht MULS(x) met operand van foute pariteit in een "loop" raakt; de opdracht werd niet afgemaakt. Deze hardware fout is inmiddels door Electrologica hersteld.

- 10.4. Zoals in 5.10. al werd opgemerkt, kunnen niet alle IF's true gemaakt worden, noch kunnen alle LVIF's false gemaakt worden. Hieronder volgt een lijst van alle 40 IF's en LVIF's in de volgorde waarin ze door het CRO uitgelezen kunnen worden. Een x geeft aan dat het onderhavige register variabel is (d.w.z. zowel true als false gemaakt kan worden), een - dat de waarde vast is (false voor IF, true voor LVIF),

app.no.	IF	LVIF	apparaat
39	x	x	klok
38	x	x	reset
37	x	x	foutdetector
36	x	x	
35	x	-	
34	x	-	
33	-	-	
32	-	-	
0	x	x	handlezer 1
1	x	x	telex
2	x	x	keyboard
3	x	x	commandotelex

4	x	x	keyboard commandotelex
5	x	x	bandlezer 2, IP-lezer
6	x	x	telex
7	x	x	keyboard
8	x	x	telex
9	x	x	keyboard
10	x	x	bandlezer 3
11	x	x	regeldrukker
12	x	x	PX-kanaal
13	x	x	XP-kanaal
14	x	-	
15	x	-	
16	x	x	keyboard
17	x	x	telex
18	x	x	bandponser 1
19	x	x	bandponser 2
20	x	x	bandponser 3
21	x	x	plotter
22	-	-	
23	-	-	
24	x	x	schijven
25	x	x	kaartlezer
26	x	x	kaartponser
27	x	x	trommel
28	x	x	magneetband eenheden
29	x	x	
30	-	-	
31	-	-	

Uit bovenstaande lijst blijkt dat nummers 30 en 29 de enige niet-aanwezige apparaten zijn waarvan IF en LVIF naar behoren werken.

- 10.5. In het Reference Manual A6.3.6. staat dat er "totdat de opdracht volgend op een ITV-zetting is uitgevoerd, niet op ingrepen gereageerd wordt". Dit is maar betrekkelijk waar; op CHARON-ingrepen wordt inderdaad niet gereageerd, terwijl protectie-ingrepen niet op kunnen treden aangezien de uitvoering van ITVON eist dat BI = true. Pariteits- en adres-ingrepen kunnen echter wel degelijk optreden. Hier rijst dan de vraag of de ten gevolge van de ingreep uitgevoerde opdracht nog wordt uitgevoerd met ITV = true. Bij experimenten bleek dit bij pariteitsingrepen telkens niet het geval te zijn, terwijl bij adresingrepen het gedrag grillig was; soms werd dan de opdracht in M[26] wel met ITV = true uitgevoerd. Zie verder 5.9.
- 10.6. De EL X8 heeft drie opdrachten die impliciet B ophogen, t.w. de uit-opdrachten met MC-adressering, de stapelende subroutinesprong en de opdracht MEMPROT. Bij positieve B is de werking duidelijk, bij negatieve B is de werking volgens het Reference Manual ongedefinieerd.

Experimenteel blijkt het volgende:

- a. Bij uit-opdrachten met MC-adressering (b.v. $B = -10$, $MC[11] = A$) wordt ook een negatieve B met 1 opgehoogd, tenzij $B = -0$, welke dan overgaat in +0.
- b. Bij de SUBC-opdracht met $B \leq -0$ volgt een adresingreep.
- c. Bij de opdracht MEMPROT geeft $B < -0$ een adresingreep terwijl voor $B = -0$ het woord $M[0]$ gebruikt wordt en B de waarde +0 krijgt. De ophoging van B geschiedt dus kennelijk zonder 'end-around-carry'.

10.7. Een pariteits-, adres- of protectie-ingreep kan alleen optreden als OV true is. In de link die tengevolge van de subroutinesprong weggeschreven wordt, is bit24 dus altijd true en draagt als zodanig geen informatie. de bit kan daarom gebruikt worden om antwoord te geven op een vraag die in het ingreep-programma gesteld moet worden, namelijk: "Is deze ingreep ontstaan door een fout van het lopende programma of doordat het lopende programma door een CHARON-ingreep onderbroken werd en de opdracht in $M[24]$ de foutingreep veroorzaakte?". Uit de beschrijving in het Reference Manual A6.3.3. blijkt inderdaad dat in de weggeschreven link de notitie $OV = \text{false}$ wordt gemaakt indien "de foutingreep is opgetreden tijdens de honorering van een CHARON- of operateurs-ingreep". Dit gebeurt echter niet. Deze fout is niet door Electrologica hersteld.

10.8. Bij het lezen van de beschrijving van de opdracht DOS(x) rijst de vraag, wat DOS(S) doet. Het antwoord is conform de beschrijving: het adres van de operand (d.w.z. 60, zijnde het adres van S) wordt in S gehaald, waarna de operand, S, wordt uitgevoerd. Het getal 60 wordt dus als opdracht, d.w.z. als $A + M[60]$ uitgevoerd. $M[60]$ is echter weer S en bevat 60; het totaaleffect is dus:

$$\begin{aligned} S &= 60 \\ A &+ 60 \end{aligned}$$

Uit de tijdsdiagrammen die bij de EL X8 behoren blijkt dat de stappen inderdaad met zekerheid in de hierboven genoemde volgorde uitgevoerd worden.

10.9. De opdracht U,GOTO(x) heeft als eigenschap dat tijdens de sprong OF false gemaakt wordt. Als x een woord van foute pariteit is, wordt de sprong ergens middenin afgebroken en volgt een ingreep. Volgens het Reference Manual A5.2.3. is het "undefined" tot hoever de opdracht heeft plaats gehad. Inderdaad blijkt dat bij de opdracht U,GOTO(woord van foute pariteit) OF soms wel en soms niet "gecleard" wordt. Iets dergelijks zou ook moeten gelden voor de in-opdrachten met MC-adressering waarbij de geheugenoperand van foute pariteit is; hierbij zou B ook soms wel en soms niet afgelaagd moeten zijn. Dit effect werd niet waargenomen; B behield altijd zijn oude waarde.

- 10.10. De werking van ITV beperkt zich tot de volgende opdracht. De verwachting is, dat wanneer dit een DO-opdracht is, ITV ook zal gelden voor de hierdoor uitgevoerde opdracht, enzovoort. Dit blijkt inderdaad zo te zijn.
- 10.11. Van een aantal opdrachten wordt in het Reference Manual niet aangegeven hoe lang ze duren. De ontbrekende tijden zijn in de volgende lijst te vinden.

snelle registertransporten	2.5 mmsec
IFA, IFS, IFAC, IFSC, etc.	3.75 mmsec
IFON(n) etc.	3.75 mmsec
OVON, OVOFF, IVON, IVOFF	2.5 mmsec
ITVON	2.5 mmsec
MEMPROT	7.5 mmsec

- 10.12. Uit de beschrijving van de afhandeling van een foutingreep (zie Reference Manual A6.3.3.) blijkt dat OV false gemaakt wordt voordat de ingreepopdracht uitgevoerd wordt, terwijl de beïnvloeding van BT en IV erna komt. Dit houdt in dat, wanneer de ingreepopdracht b.v. $S = T$ is, uit de inhoud van S moet blijken dat OV al een nieuwe waarde heeft terwijl IV en BT nog de oude waarden hebben. Dit werd experimenteel bevestigd.
- Door uit te gaan van de niet-bestuurstoestand (dus BT = false, OV = IV = true) kunnen we zo de machine in een (pathologische) toestand krijgen waarbij BT = false niet impliceert dat OV = IV = true (hetgeen normaal altijd wel het geval is).
- 10.13. Conform de beschrijving (Reference Manual A5.2.2.c) treedt een protectie-ingreep op als in de niet-bestuurstoestand een sprong door een protectiepoort wordt uitgevoerd terwijl $+0 < B < 256$. Hoewel dit duidelijk met opzet gedaan is, blijkt de zin hiervan moeilijk te doorgronden. De twee meest voor de hand liggende redenen zijn onjuist.
- Het zou gedaan zijn om de apparaatregisters te beschermen. Echter, elk zinnig protectiesysteem werkt toch al met gedeeltelijke of gehele protectie op de eerste 512 geheugenplaatsen.
 - Het zou gedaan zijn omdat tijdens de sprong de geheugenprotectie wordt opgeheven. Dit geldt echter beslist niet voor het wegschrijven van de link.

Drie andere redenen klinken plausibeler:

- Stel B zou kleiner zijn dan 16, dan kan de link wel weggeschreven worden en naar alle waarschijnlijkheid over belangrijke informatie. Het is nu echter niet duidelijk waarom de test $B < 256$ luidt en waarom de gebruiker de plaatsen 0 tot 15 die tot het gebruikersprogramma behoren, niet mag gebruiken zoals hem dat goeddunkt.
- Stel dat B tussen 57 en 62 zou liggen, dan zou de link verloren gaan, terwijl bij de terugsprong uit het systeem naar het gebruikersprogramma de kleine registers en OT met onvoorspelbare waarden gevuld zouden worden wat duidelijk ongewenst is.

- e. Ook het systeem heeft voor het afhandelen van de protectiepoortroutines een stack nodig. Verder zullen een aantal protectiepoortroutines op hun beurt weer andere protectiepoortroutines aanroepen; bij een aanroep door het gebruikersprogramma zullen meegegeven parameters nauwkeurig op toelaatbaarheid getest moeten worden, terwijl bij een aanroep vanuit een andere protectiepoortroutine deze test achterwege kan blijven. Wordt de systeemstack nu in een van de trajecten M[29] tot M[56] of M[224] tot M[255] gekozen, dan blijkt uit de waarde van B na de aanroep onmiddellijk of de aanroep gedaan werd door het gebruikersprogramma ($B \geq 256$) of door een ander protectiepoortprogramma ($B < 256$). Het is bij deze verklaring echter niet duidelijk waarom een protectiepoortroutine die een andere wil aanroepen dit niet kan doen op een punt waar alle tests al gedaan zijn.

- 10.14. Uit de beschrijving van de opdracht MEMPROT (Reference Manual A5.2.5.) blijkt dat de operand met een M[B]-adressering gehaald wordt (hoewel de bitconfiguratie van de opdracht een MC-operand aanduidt). Hieruit zou volgen dat de operand een van de registers mag zijn. Dit werd experimenteel bevestigd.

- 10.15.1. Uit de beschrijving van de sprong door een protectiepoort (Reference Manual A5.2.4.) zou blijken dat gedurende het halen van de operand BT true zou zijn. Treedt er dus bij het halen van de operand een pariteitsingreep op, dan wordt in de link BT = true genoteerd waaruit ten onrechte de conclusie getrokken zou kunnen worden dat de instructie die aanleiding was tot de protectie-ingreep, d.w.z. de subroutinesprong, in de bestuurstoestand uitgevoerd was. Bovenstaand effect werd experimenteel niet gevonden; het blijkt dat tijdens het halen van de operand de geheugenprotectie uitgeschakeld wordt zonder BT te beïnvloeden.

- 10.15.2. Uit dezelfde beschrijving blijkt dat de hoofdfunctie is, OT, BT en IV nieuwe waarden te geven. Het nieuwe adres dat in OT gezet moet worden kan echter niet-aanwezig of niet-bestaand zijn, in welk geval er een adresingreep optreedt. Het blijkt dan experimenteel dat het "undefined" is of de wijzigingen in BT en IV al plaats gehad hebben. Misschien is het zo zelfs mogelijk de machine in de (pathologische) toestand BT = IV = false te krijgen, hoewel dit met een beperkt aantal malen proberen niet lukte.

10.15.3.

Uit het voorgaande blijkt dat het protectiepoortentraject geen woorden mag bevatten, die, als adres opgevat, een fout adres zouden zijn; de gebruiker zou dan namelijk een adresingreep kunnen veroorzaken met een onontwarbare link. Het traject moet dus gevuld worden met de adressen van de routines en alle sprongen door een protectiepoort moeten inhoudelijke sprongen (STAT, STATB of DYN) zijn. Niets weerhoudt de gebruiker echter om met een directe sprong (:STAT of :DYN) door een protectiepoort te springen; de opeenvolgende adressen in het traject worden dan als opdrachten uitgevoerd (met BT = true) en wel als A + M[adres]-opdrachten. Dit kan geen kwaad, wel moet M[320] dan een sprong naar een foutmeldingsroutine bevatten.

10.16. Het bleek experimenteel dat een inhoudelijke stapelende subroutinesprong (d.w.z. SUBC(STAT), SUBC(STATB) of SUBC(DYN)) met B = +0 een adresingreep veroorzaakt; dit effect treedt niet op bij een directe stapelende subroutinesprong (d.w.z. SUBC(:STAT) of SUBC(:DYN)) met B = +0. Zulks wordt bevestigd door de bij de EL X8 behorende tijdsdiagrammen; de reden hiervan is onbekend.

10.17. De besturing van de EL X8 heeft 4 statusvariabelen, de besturingstoestand, de ingreepvergunning, de onderzoekvergunning en de CRO-toestand (DYST1/2 versus lopend); de laatste is geïmplementeerd als een bijzondere bitconfiguratie in OT, de andere zijn elk een bit. Met deze 4 variabelen (en even afziende van het verschil tussen DYST1 en DYST2) kunnen we 16 combinaties maken.

BT IV OV DYST

1	1	1	ja	het systeem wacht
1	1	1	nee	het systeem loopt
1	1	0	ja	begintoestand (IP, NB)
1	1	0	nee	systeeminitialisatie
1	0	1	ja	???
1	0	1	nee	het systeem loopt, doof
1	0	0	ja	???
1	0	0	nee	na inlezen IP-band, enz.
0	1	1	ja	???
0	1	1	nee	een gebruikersprogramma loopt
0	1	0	ja	—
0	1	0	nee	tijdens foutingreep in gebruikersprogramma
0	0	1	ja	—
0	0	1	nee	—
0	0	0	ja	—
0	0	0	nee	—

Vijf van deze combinaties kunnen niet optreden, een komt slechts in een bijzonder geval kortstondig voor, en drie werpen vragen op. Twee daarvan behelzen de toestand waarbij de machine in doofheid in DYST geraakt; in principe is de machine daardoor in een statische stop geraakt: alleen de operator kan nog iets doen. De derde twijfelachtige toestand is die waarbij een gebruikersprogramma (in de niet-besturingstoestand) al dan niet opzettelijk in DYST geraakt; hierdoor ontstaat een voor de operator misleidende en voor de gebruikers onprofijtelijke situatie.