

Bij het operationeel worden van PTT's "zeer voorlopige ELAN-assembler" en bij de voorbereidingen tot meer definitieve en volledige assemblers (PTT en ERC), lijkt het gewenst én zelfs noodzakelijk, een aantal punten tot klaarheid te brengen:

- a) in de praktijk is gebleken, dat sommige ~~BB~~ genomen beslissingen minder gelukkig waren,
- b) een belangrijk aspect van ELAN, - de MACROfaciliteit -, is tot nu toe grotendeels in de mist gebleven,
- c) een aantal details zijn nog steeds niet vastgelegd,
- d) over het samenspel tussen assembler(s) en postmortemprogramma's (desassemblers, tracers) alsmede handregisterfaciliteiten, is nog vrijwel niet gesproken.

Bijgaande nota bestrijkt een goed deel van ~~BBB~~ de materie sub a t/m d; de meeste punten zijn ingekleed als voorstellen. Deze nota is ontstaan uit gesprekken van ondergetekende met de heren:

Balder (EL),
Bruinenberg (EL),
Doetjes (PTT),
Kerbosch (TH-Eindhoven),
Kolff (EL),
vd Poel (PTT),
Silvester (ERC),
Veer (EL),
Witmans (PTT),
Zegeling (ERC).

Inhoud en formulering van een belangrijk deel van deze nota (globaal de ~~BBB~~ paragrafen 5 t/m 16) werd uitvoerig besproken op het PTT-laboratorium te Leidschendam, er werd een grote mate van overeenstemming bereikt.

Het lijkt van belang dat zo spoedig mogelijk beslissingen worden genomen over alle punten ~~BBB~~ die nog niet vastliggen, reden waarom deze nota met enige ~~minim~~ aandrang aan de Z8-commissie wordt voorgelegd.

S.G. van der Meulen
(ERC Utrecht)

INHOUD

1. ELAN en assemblage
- * 2. Indeling en gebruik ~~van~~ van de geheugenruimte
3. Toestand van de geheugenruimte na beëindiging van de assemblage
4. Interne representatie van de ELAN-basic-symbols
- * 5. Rusttoestand en instructionseparators
- (6) Assemblage-instructies
- (7) Programma-instructies
- * 8. Blok-structuur, declaratielijsten
- * 9. Blok-structuur (vervolg), heersende adresseringsvorm,
lokaliteit van namen
- 10.* Operanden en index-expressies
- 11.* Operatoren en voortekens
- 12.* Functienamen en voortekens, octale functienamen
13. Alfanumerieke strings
- (14) Constanten
- 15.* Constanten in opdrachten
- 16.* Macro's
- 17.* Varianten
- 18.* Varianten, notatie

Niet alle paragrafen zijn van gelijk gewicht. De met een * gemerkte bevatten de belangrijkste kwesties waarover een spoedige beslissing ~~van~~ gewenst is. De paragrafen met nummers tussen haakjes zijn alleen toegevoegd voor referenties en ook wel om en passant een nomenclatuur vast te leggen. De paragrafen zonder bijzondere kenmerken bevatten details die wel eens een keer vastgesteld moeten worden.

1. ELAN en assemblage

Uitgangspunt was, dat ELAN een door de Z8-commissie vastgestelde (en op sommige punten nog nader te preciseren) programmeertaal is, waarvan de uiteindelijke vorm (en betekenis) zo exact mogelijk in een rapport moet worden vastgelegd en niet wordt bepaald door wat enige assembler er toevallig wel of niet van implementeert.

ELAN is voorts vooral een machinetaal en daarom is het zelfs niet voldoende dat door verschillende assemblers vanuit eenzelfde tekst geassembleerde objectprogramma's dezelfde resultaten geven, maar dat bovendien deze objectprogramma's in de machine identiek zijn in hun binaire voorstelling.

Bij ELAN-opdrachten als:

```
A 'x' 100 000
A = S
GOTO ( :MT[250] )
etc.
```

~~IS~~ dit niet zonder meer gewaarborgd.

In onze gesprekken werd ~~maar~~ onderscheid gemaakt tussen verschillende ELAN-“niveau's” :

- ELAN 0 kent alleen de standaardnamen, eventueel een uitgebreide set standaardnamen: er is geen vrije naamkeuze.
- ELAN 1 kent vrije keuze van operandnamen, er kunnen labels worden gezet, er mag gebruik worden gemaakt van niet-declareerde namen en constanten buiten het interval 0 - 32767.
- ELAN 2 kent het macro-mechanisme, en daarmee de vrije keuze van functienamen.
- ELAN 3 verdere uitbreidingen van ELAN.

De wenselijkheid van een ~~concentrische~~ "concentrische assembler" (in "modulen") voor ELAN 0 - 1 - 2 (- 3) werd besproken. Een ELAN 0 - assembler zou nuttig kunnen zijn voor een gestroomlijnd handregister-programma.

Het lijkt van belang dat de Z8-commissie zich uitspreekt over de zin van deze ELAN-klassificatie en de mate waarin toekomstige assemblers zich ~~aan~~ hierbij aansluiten.

In het bijzonder het probleem van verdere uitbreidingen van ELAN verdient enige aandacht:

bij de afdeling operational research van de TH te Eindhoven werden ~~de~~ wensen geformuleerd, waarover tzt. mededelingen zullen volgen.

We stellen de volgende procedure voor:

(zie 5)

Wanneer een assembler in rusttoestand een onbekende woorddelimiter ontmoet, wordt naar een bepaald adres gesprongen waar een verdere uitbouw van de assembler kan worden aangehaakt. Daarnaast zal het op eenvoudige wijze mogelijk zijn de lijst van "bekende woorddelimiters" uit te breiden en aldus op een meer gestandaardiseerde wijze de assembler uit te breiden. Overeenkomstig voor nieuwe woorddelimiters in declaratielijsten (zie 8). ~~concentrische~~ In ad-hoc uitbreidingen voorziet ~~van~~ de "interlude".

Zowel de PTT (Leidschendam) als het ERC (Utrecht) zijn bezig met het ontwerp van een meer definitieve ELAN-assembler. De inzichten verschillen nogal, men is het echter in grote lijnen eens over de handvatten die er aan mogelijk zeer verschillende assemblers moeten zitten en liefst (evenals het beoogde objectprogramma) identiek moeten zijn, zodat geen gebruiker gedwongen zal zijn rekening te houden met de assembler die zijn tekst vertaalt.

2. Indeling en gebruik van de geheugenruimte

Onder "geheugenruimte" wordt hier verstaan: het stuk geheugen dat (door de coördinator) aan de assembler ter beschikking is gesteld. In deze ruimte wordt niet alleen het objectprogramma opgebouwd, maar moeten bovendien de naamstapel, de macrobuffers en de werkruimte worden ondergebracht.

We onderscheiden:

- | | |
|---------------------------------|--|
| 1) de <u>programmaruimte</u> , | hierin wordt het objectprogramma opgebouwd, |
| 2) de <u>assemblageruimte</u> , | naamstapel + werkruimte, |
| 3) het <u>MDmin-gebied</u> , | grootheden die de assembler snel moet kunnen raadplegen en voorts de grenspunten van de geheugenruimte-indeling,
deze grootheden zijn geadresseerd als:
$MD[-1]$, $MD[-2]$, enz. |

Het eerste adres van de geheugenruimte staat permanent in D ($=M[63]$).

Het eerste adres van de assemblageruimte ("BODAS"), de top van de assemblageruimte ("TOPAS") en de top van de geheugenruimte ("ENDAD") staan in het MDmin-gebied.

D en ENDAD worden door de coördinator bepaald en kunnen niet door de assembler worden gewijzigd. BODAS en TOPAS moet iedere programmeur vrij kunnen kiezen en ook wijzigen als hij daaraan behoeft heeft:

Bij het begin van ~~XXXXXXXXXX~~ de assemblage is het inzet-adres (een van de grootheden die in het MDmin-gebied gevonden wordt) gelijk aan ~~M~~ M[D], iedere programmatekst kan geacht worden te zijn begonnen met de inzet-aanwijzing:

M[D] :

Wil een programmeur ergens anders gaan inzetten (of een andere adresserings-vorm voor zijn programma, zie 9) dan doet hij dat door een andere inzet-aanwijzing te geven.

Bij het begin van de assemblage valt TOPAS samen met ENDAD terwijl BODAS bijv. halverwege D en ENDAD ligt.

Zolang de programmeur zwijgt over BODAS en TOPAS tracht de assembler bij conflict tussen programma- en assemblageruimte te bemiddelen door de assemblage ruimte te verkleinen, wanneer de assemblageruimte te klein blijkt wordt getracht enige ruimte van de programmaruimte over te nemen: er wordt steeds voor gezorgd dat: inzetadres < BODAS .

Dit is de ~~M~~ normale assemblage-modus.

Door te schrijven: 'START' :STACK ; nieuwe BODAS ; nieuwe TOPAS ; definieert de programmeur nieuwe grenzen voor zijn assemblageruimte: de (re-locateerbare) informatie in de oude assemblageruimte wordt verhuisd naar de nieuwe waarin verder wordt gewerkt. Controles op conflict tussen assemblage-ruimte en programmaruimte blijven nu achterwege: de programmeur die ~~KENMERK~~ 'START' :STACK heeft geschreven heeft het zelf in handen genomen en de gevolgen zijn voor zijn rekening:

de vrije assemblage-modus.

(in principe)

Op deze wijze is bereikt, dat de geheugenruimte ~~geheel~~ ter beschikking staat van de programmeur, maar dat anderzijds de programmeur die simpel te werk wil gaan, zich niet hoeft te verdiepen in zaken die hem wellicht minder interesseren.

Alle grootheden, zoals BODAS, TOPAS, ENDAD, inzet-adres etc. zijn relatief t.o.v. D.

Er wordt vanuitgegaan, dat de programmaruimte en de assemblageruimte geen van beide omvangrijker hoeven te worden dan 32768 X8-woorden, zodat de maximaal bespeelbare geheugenruimte zou kunnen zijn: 32K voor het programma, waarachter nog eens 32K voor de assemblage.

3. Toestand van de geheugenruimte na beëindiging van de assemblage

Bij alle assemblers moet de vulling van de programmaruimte (het objectprogramma) na beëindiging van de assemblage identiek zijn (zie 1).

Moeilijker ligt dit met de toestand van de assemblageruimte. Verschillende assemblers zullen hier op verschillende wijze mee om willen springen. Aan de andere kant kan men, i.h.b. voor de "rest-naamlijst" bepaalde wensen hebben voor post-mortem programma's (ic. des-assemblers, tracers etc.).

We hebben alleen het probleem van de rest-naamlijst bekeken:

Bij het passeren van de laatste 'END' (zie 9) krijgen de constanten en superglobalen een plaats volgens het dan geldend inzet-adres (dat wil in de meeste gevallen dus zeggen: achter het zojuist gerassembleerde programma).

Men kan dan de rest-naamlijst (waarin de super-globalen en de in het buitenste blok gedeclareerde namen) "aansluiten" door te schrijven:

'START' :STACK ; :LLL ; :LLL[100]

waarin "LLL" ~~nu~~ de "slot-label" van het programma is (een label voor de laatste 'END') en men bijvoorbeeld 50 namen wil meenemen.

In deze rest-naamcellen moeten de volgende informatie-eenheden worden geborgen:

- 1) de geconverteerde naam, (32 bits indien 37-tallig geconverteerd)
- 2) het adres van deze naam, (15 bits)
- 3) nog twee bits voor meededelingen inzake de adresseringsvorm van de naam

Over de wijze waarop deze informatie in twee woorden gepakt zou moeten worden, is nog geen overeenkomst bereikt. Een of andere vorm van standaardisering die uitgaat van de behoeften der post-mortemprogramma's lijkt gewenst. Hierover moet wel een beslissing worden genomen.

4. Interne representatie van de ELAN-basicsymbols

< wordt tegelijk met het definitieve rapport over ELAN gepubliceerd >

5. In rusttoestand is de assembler in staat, hetzij een assemblage-instructie, hetzij een programma-instructie (zie 6 en 7) te accepteren.

De rusttoestand treedt in na het lezen van een instruction-separator, (voortaan "insep") als zodanig kan optreden:

- a) een ":" na een al of niet geïndiceerde naam in een kantlijn-instructie (zie 6),

OPM een ":" als "adres-operator" voor een naam,
een ":" achter een naam in een MACRO-declaratie worden niet als insep geïnterpreteerd.

- b) een ";" resp "newline" (deze zijn geheel identiek in ELAN) alsmede een aanhalingssteken resp dubbel-accent gevuld door te skippen comment tot ";" resp "newline",

OPM een ";" resp "newline" onmiddellijk voorafgegaan door een komma, ~~wordt niet als insep geïnterpreteerd~~ heeft dezelfde

vb: betekenis als een komma

```
AA[128]@②
BLOKNAAM@②
'BEGIN' L1, L2,
'MT' K1, K2,    '' COMMENTAAR
           K3,      " COMMENTAAR ; K4,
'MACRO' NN :
'BEGIN'
          " MACROBODY
'END' NN " EINDE DECLARATIELIJST ②
GOTO ( :PP )@
```

De omrande puntkomma's en dubbelepunten zijn insep's evenals de met pijltjes aangegeven newlines, alleen de aldus gemarkeerde inseps ~~worden~~ brengen de assembler in rusttoestand.

Tussen stringquotes (zie 13) bestaan geen inseps, in een MACRO-body (zie 16) alleen bij evaluatie na ~~;~~ aanvraag.

6. Assemblage-instructies

Dit zijn instructies aan de assembler ("directieven"), als zodanig dragen ze slechts indirect bij tot het objectprogramma: door een assemblage-instructie wordt niet in het objectprogramma geschreven. We onderscheiden:

- a) kantlijn-instructies: identificatie: Er wordt een naam "bekend" gemaakt (als label uitgegeven)
inzet-aanwijzing: Het inzet-adres krijgt een nieuwe waarde, er wordt een adresseringsvorm gedefinieerd
Kantlijn-instructies worden afgesloten door de insep ":" → locale
- b) blok-'BEGIN': Opent een nieuw blok, de assembler accepteert de toevoeging van ~~namen~~ namen aan de naamlijst en nieuwe macro-buffers (voor declaraties zie 9).
- c) blok-'END': Sluit een blok af, de gedeclareerde namen verliezen hun locale betekenis. Een eventuele naam achter 'END' wordt op adres en adresseringsvorm vergeleken met de blok-'BEGIN', bij discrepantie wordt alarm geslagen.
- d) 'SKIP': Het inzet-adres wordt verhoogd resp. verlaagd met het achter 'SKIP' genoteerde bedrag (constante of replica).
- e) 'START': Brengt de assembler in rusttoestand en veroorzaakt vervolgens een directe sprong naar het achter 'START' gedefinieerde adres.

7. Programma-instructies

Deze worden na conversie in het objectprogramma ingezet. De normale op-hoging van het inzet-adres kan alleen worden doorbroken door een 'SKIP' - instructie of een inzet-aanwijzing. We onderscheiden:

- a) opdrachten: genoteerd als functie: zonder parameter
genoteerd met operator (zie 11)
met 1 parameter

b) getallen: FIXT-getallen enkellengte] (zie 14)
FLOT-getallen dubbellengte]

c) strings: alfanumerieke strings (zie 13)
pictures aangekondigd door de woorddelimiteur 'PICT'

d) MACRO-aanvragen genoteerd als functie met geen, een of meer parameters (zie 16)

8. Blok-structuur

Een blok is een (ELAN-)tekst beginnend met 'BEGIN' en eindigend met 'END'. De blokstructuur kan worden gebruikt voor:

- a) het geven van een lokale betekenis aan namen (zie hieronder: declaration-listen)
 - b) het afdwingen van een bepaalde adresseringsvorm voor binnen dat blok uitgegeven labels (zie 9)
 - c) voor de **KKK** begin-end-markering van een MACRO-declaratie. (zie 16)

Achter de 'BEGIN' mag een declaratielijst worden geopend, deze kan bestaan uit drie delen:

- 1) gewone lokale operandnaamlijst: namen gescheiden door komma's, de betekenis van deze namen is lokaal (voor hun adressering zie 9)
 - 2) lokale 'MT'-operand naamlijst: namen gescheiden door komma's maar voorafgegaan door de woorddelimiteur 'MT', de betekenis van deze namen is lokaal, ze worden MT-gadresseerd
 - 3) lokale 'MACRO'(functie)naamlijst: een lijst van 'MACRO'-declaraties gescheiden door komma's (zie 16)

De volgorde van de drie bestanddelen van een declaratielijst is de hierboven gegevene; elk der drie bestanddelen mag echter leeg zijn.

Een gelabeld blok is een blok voorafgegaan door een of meer kantlijn-instructies. al of niet

Een EIAN programma is een gelabeld blok.

Een "macro-body" is een ongelabeld blok (men zou de macronaam met de formele parameters als "macroblok-label" kunnen aanzien, een macronaam is echter essentieel een functienaam zodat de verhoudingen hier beslist anders liggen als bij een gelabeld blok), 'BEGIN' en 'END' markeren hier het begin en einde van de MACRO-buffer (zie 16).

Een declaratielijst eindigt bij de eerste ";" resp. "newline" (eventueel voorafgegaan door comment) die niet wordt voorafgegaan door een komma.

Een declaratielijst moet beginnen op dezelfde regel als 'BEGIN'. Een blok-'BEGIN' gevolgd door ";" of "newline" geeft een lege declaratielijst.

8. (vervolg)

vb:

AA[128]:
 BLOKNAAM: " het blok is gelabeld
 'BEGIN' L1, L2, L3, " gewone lokale operandnaamlijst
 'MT' K1, K2, K3, " lokale 'MT' ~~is~~ operandnaamlijst
 'MACRO' NN1 (J1, J2) : " begin macro-declaratielijst
 'BEGIN'
 "
 " " macro-buffer NN1, formele namen J1 en J2
 "
 " " " de declaratielijst wordt voortgezet
 'END' NN1,
 'MACRO' NN2 :
 'BEGIN'
 "
 " " " macro-buffer NN2, geen formele namen
 "
 " " " " de declaratielijst wordt afgesloten
 "
 "
 " " PROGRAMMA
 "
 " " "
 'END' BLOKNAAM " einde blok-voorbeeld

9. Blok-structuur (vervolg)

Heersende adresseringsvorm

Alle 'MT'-gedeclareerde namen worden ~~xx~~^{MT}-geadresseerd, de MT-index kan door de assembler worden berekend zodra de naam bekend wordt. Een MT-label moet liggen in de dynamische omgeving van de opdrachten waarin hij wordt gebruikt (uitzondering GOTO(:)), die bij overschrijding wordt ~~om~~gezet in een JUMP()).

Alle namen, voor zover ze niet in een lokale 'MT'-lijst voorkomen, die binnen eenzelfde blok als label verschijnen volgen dezelfde adresseringsvorm: de in dat blok heersende adresseringsvorm.

De in een blok heersende adresseringsvorm is bepaald door:

- a) hetzij ~~aan~~^{de} de blok-'BEGIN' voorafgaande ~~zzzzzzzzzzzzzzzzzzzz~~ kantlijn-instructie
- b) hetzij (bij afwezigheid van zo'n ~~zzzzzzzzzzzzzzzzzzzz~~) de in het omvattende blok heersende adresseringsvorm.

Wanneer de heersende adresseringsvorm "Mp" is en de index-grens wordt overschreden, schakelt de assembler automatisch over op "p+1" met overeenkomstige invulling van het corresponderend display-element MD[p+1].

Wanneer de eerste 'BEGIN' van een ELAN-programma niet wordt voorafgegaan door een inzet-aanwijzing of Mp-label, is de heersende adresseringsvorm de statische.

vb:

M3: " de heersende adresseringsvorm is "M3"
 'BEGIN' P,Q,R
 " bij bekendwording in een identificatie worden P,Q
 " en R geadresseerd als "M3[q]", wanneer "q" groter
 " is dan 256, wordt dit automatisch "M4[q-512]" met
 "bijwerking van MD[4]
 'END' M3

9. (vervolg)

Lokaliteit van namen

Iedere naam moet bekend worden gemaakt (als label worden uitgegeven) binnen het blok waarin hij is gedeclareerd.

OPM Het is onverstandig om automatisch gedeelde plaatsen te gaan uitdelen aan namen die lokaal zijn in parallelle blokken en onbekend zijn gebleven. Immers: elke operandnaam in ELAN is altijd de naam van een "array" van a priori onbekende lengte.

Namen die gedeclareerd (lokaal) zijn in een blok, heten globaal in een binnenblok voor zover zij niet voorkomen in de declaratielijst van dat binnenblok. Ook wanneer een naam bekend wordt binnen een blok waarin hij globaal is, volgt hij de in dat blok heersende adresseringsvorm (tenzij uiteraard deze naam 'MT'-gedeclareerd was).

In ELAN mogen ook namen worden gebruikt die ~~gebruikt~~ nergens zijn gedeclareerd, dergelijke namen heten superglobaal. Ze worden behandeld als te zijn gedeclareerd op gelijk niveau met de standaard-namen (in het "alles-omvattend buitenste blok"), in tegenstelling tot dezen zijn ze bij het eerste 'BEGIN' van de assemblage nog niet bekend.

De assembler bevat een routine "LABEL" die alle lokaal gedeclareerde namen die nog onbekend zijn, behandelt alsof ze op dat moment in successie bekend worden gemaakt. Het inzet-adres wordt met 1 resp. 2 verhoogd na elke automatische labeling, al naar gelang er niet of wel een F-opdracht voorkomt in de bijwerkketen. De aldus automatisch gelabelde namen worden door de assembler uitgeprint. De assembler-routine "LABEL" wordt aangeroepen met:

'START' :LABEL

Wanneer bij het passeren van een 'END' een lokale naam onbekend is gebleven (niet op een of andere wijze is ge"labeld") wordt alarm geslagen, dit is een programmeerfout.

Bij het passeren van de laatste 'END' van een ELAN-programma ~~gaat~~ doet de assembler zelf nog een keer ~~XEMM~~ 'START' :LABEL waardoor de nog onbekende superglobale worden gelabeld (zie ook 15).

De superglobale staan, met hun adres en adresseringsvorm, in de naamlijst en blijven aldus na assemblage ter beschikking voor post-mortem-(desassemblage- en trace-)programma's.

Vb: ELANPROGRAMMA:

```
'BEGIN' U, V
    "      opdrachten
    "
U:      "
BINNENBLOK:
'BEGIN' X, Y, Z
    "      opdrachten
    "
X:      "
GOTO(:SUPERGLOBAAL)
    'START' :LABEL      " effect:   Y: 'SKIP' k
                           "          Z: 'SKIP' k
                           " k = 1 of 2
SUPERGLOBAAL:   "      opdrachten
    "
    'END' BINNENBLOK
    "
    "
    'START' :LABEL      " effect:   V: 'SKIP' k
    "
'END' ELANPROGRAMMA
```

" Nu worden door 'START' :LABEL de nog onbekende superglobale gelabeld

10H Operanden en index-expressies

Als doel-operand in een opdracht kan optreden:

- a) een variabele: een niet-geïndiceerde naam (die onbekend mag zijn)
een geïndiceerde naam (die bekend moet zijn)

Als bron-operand in een opdracht kan bovendien optreden:

- b) een adres: een ":" gevuld door een variabele (als onder a)
c) een constante (zie 15)

Achter een bekende naam mag tussen vierkante haken een index-expressie worden geschreven, deze definieert het bedrag waarmee het aan de naam toegekende adres moet worden verhoogd resp. verlaagd.

OPM De eis, dat een geïndiceerde naam bekend moet zijn, is indertijd oa. uit implementatie-overwegingen ingegeven. In een later ontwikkelings-stadium van ELAN is hierop voortgeborduurd in de klassificatie van de kantlijn-instructies:

<identificatie> ::= <onbekende naam> :

<inzet-aanwijzing> ::= <bekende naam> [<index-expressie>] :

Niettemin blijft het verbod indices te schrijven bij een onbekende naam een onvolkomenheid van de taal. Het is van belang thans te beslissen of dit verbod tot de taal behoort, of een onvolkomenheid blijft van (sommige?) assemblers.

Een index-expressie staat tussen vierkante haken en is van de vorm:

" p × q + r " waarin "p", "q" en "r" operanden zijn in de zin
a), b) of c)
of eenvoudiger: } overal waar "+" staat mag ook
" p × q " " q + r " " p " } "-" worden geschreven

De uiteindelijke waarde van een index-expressie (alsmede elke tussenwaarde) mag de capaciteit van een FIXT-constante (zie 14) niet overschrijden. Het interval waarbinnen de uiteindelijke waarde moet liggen hangt af van de adresseringsvorm en de adres-waarde van de naam waaraan de index is gehecht.

In plaats van " :M[index-expressie] " mag ook worden geschreven:

" [index-expressie] " (" :" en "M" annihileren elkaar)

Op deze wijze mogen haken worden gezet in index-expressies en kunnen ~~machinewoorden~~ machinewoorden worden opgebouwd uit replica's.

vb: GOTO (:LL[X + Y[12]]) " uitgerekend wordt de waarde van X + Y[12]
" tijdens assemblage, met deze waarde
" wordt de adreswaarde van LL vermeerderd en
" dit is het adres waar naartoe gesprongen
" moet worden

vb: BIT12: '000010000'

"

"

AA: [7 × BIT12 + X[2]]

vb: A = :ALFA [X - [P × Q + R]]

Een bijzonder geval is de B-modificatie:

Wanneer de eerste operand in een index-expressie op het eerste niveau de betekenis "B" heeft, wordt dit opgevat als een B-modificatie. De naam van de operand die deze index draagt moet statisch zijn of worden, en hoeft niet noodzakelijk bekend te zijn, wanneer er niets meer bij "B" wordt opgeteld door de index-expressie.

11. Operatoren en voortekens

In ELAN mogen opdrachten worden geschreven met als bron-operand een constante die niet bij de opdracht kan worden geborgen (zie 15). Het is mede daardoor noodzakelijk, vast te stellen of een "voorteken" (indien toegestaan) bij de operand hoort, dan wel bij de operator (resp het functiedeel, zie ook 12):

vb:

Een opdracht als

A = -100000

kan worden geïnterpreteerd als

A = MINHONDERDDUIZEND

resp.

A == HONDERDDUIZEND

Door het "voorteken" bij de operator te trekken wordt de grootste gelijkvormigheid bereikt (voor A = -X resp. A = -100 is slechts een interpretatie mogelijk!), bovendien wordt aldus de beste aansluiting gevonden met de binaire representatie van de opdrachten in de machine.

We onderscheiden derhalve de volgende operatoren in ELAN:

+		
-		
=		
==		
x		
/	(zie ook 4)	
'+'		
'+'-		
'x'		
'x'-		
		++
		'+'
		'+'-
		'x'
		'x'-
		(samengestelde operatoren)

12. Functienamen en voortekens, ~~MACRO's~~ octale functienamen

Hetgeen onder 11) is gezegd over met operator genoteerde opdrachten geldt overeenkomstig voor met functienaam genoteerde opdrachten:

vb:

Een opdracht als

MULAS (-100000)

wordt geïnterpreteerd als

MULAS (- HONDERDDUIZEND)

Het "voorteken" wordt "bij de functienaam getrokken". Duidelijker gezegd: een voorteken wordt altijd in het functiedeel van de opdracht tot gelding gebracht en nooit door de inhoud van een geadresseerd geheugenwoord.

Niet alle op de machine mogelijke opdrachten hebben in ELAN een standaardnaam; de invoer van octaal-genoteerde constanten maakt het mogelijk elke bitconfiguratie voor te schrijven. Het is echter prettig dergelijke opdrachten, vooral wanneer ze een ~~MACRO~~operand hebben, "normaal" te kunnen noteren. Via een MACRO-declaratie kan dit worden bereikt:

vb:

```
'MACRO' SUB9 ( X ):
'BEGIN'
  '563000000'(X)
'END' SUB9
```

Een (waarschijnlijk wel altijd octaal geschreven) FIXT-constante gevolgd door een operand tussen functiehaken, wordt geïnterpreteerd als een octale functienaam waaraan de operand met de noodzakelijke controles en aanpassingen van het functiedeel moet worden toegevoegd.

vb: De bovengedeclareerde macro kan alleen worden aangevraagd met een statisch adres:

SUB9 (:LL) " LL moet statisch geadresseerd zijn

13. Alfanumerieke strings

Wanneer de assembler in de rusttoestand de stringquote-open '(' leest, worden alle volgende symbolen letterlijk (spaties, ~~newlines~~ etc. inbegrepen) in hun interne representatie (zie 4) in drie per woord gepakt en in het objectprogramma ingezet.

In een alfa-numerieke string worden karakters onderscheiden die ELAN niet hoeft te kennen (kleine letters, vraagtekens etc.)

Een alfanumerieke string wordt beëindigd door de combinatie "'')'" ; tussen de accent, de ronde sluithaak en de accent mag uiteraard geen spatie staan wil het een stringquote-sluiten zijn.

Vb:

'(' abc <>V alles doet mee ')' het is nog niet uit ')' " einde #string

14. Constanten

ELAN kent twee soorten constanten:

- a) FIXT-constanten (enkellengte)
- b) FLOT-constanten (dubbellengte)

Het lijkt gewenst te waarborgen, dat de decimaal-conversie die de assembler gebruikt exact dezelfde resultaten ~~geenkenmerken heeft~~ als de decimaal-conversie die met name "INDIT" gebruikt. ~~(heeft)~~

FIXT-constanten:

- a1) decimale notatie een rij cijfers 0 t/m 9 al of niet voorafgegaan door een voorteken,
er mag geen "." of "₁₀" in een FIXT-constante voorkomen,
de waarde mag absoluut niet groter zijn dan 67 108 863 .

- a2) octale notatie

een rij van tenhoogste 9 cijfers 0 t/m 7 tussen accenten, al of niet voorafgegaan door een voorteken.

- a3) echte breuk

een constante gevuld door een breukstreep gevuld door een constante,
de deling wordt uitgevoerd en mag geen resultaat hebben dat na afronding overloopt in het gehele deel,
het resultaat wordt na schaling met 2^{126} afgerond op 26 bits,
een echte breuk mag worden voorafgegaan door een voorteken

FLOT-constanten:

- b1) mantissee een rij cijfers 0 t/m 9 waarin ergens een ".." staat eventueel voorafgegaan door een voorteken.

- b2) exponentdeel

"₁₀" gevuld door een decimaal genoteerde FIXT-constante,
~~geenkenmerken~~

- b3) drijvend getal

een mantissee gevuld door een exponentdeel,

Het kenmerk van een FLOT-constante is, dat er tenminste hetzij een ".", hetzij een "₁₀" in voorkomt (daardoor is het onderscheiden van een FIXT-constante)

<u>Vb:</u>	FIXTconstanten:	12345678	'123456701'	1234567890/12345678900
	FLOTconstanten:	.12345	12345.	0.12345 12345000.0
		₁₀ 123	₁₀ -123	₁₀ +123
		12345 ₁₀ 0	12.345	0.0000001 ₁₀ 7

15. Constanten in opdrachten

In ELAN mag in een opdracht een constante als (bron-)operand optreden (zie 10), een eventueel voorteken wordt in het functiedeel van de opdracht tot gelding gebracht (zie 11 en 12).

Een FIXT-constante < 32768 wordt bij de opdracht geborgen in de vorm:
" :M[constante] " .

vb:

A = 12 wordt geconverteerd als A = :M[12]

A'x'''77777' wordt geconverteerd als A'x':M[32767]

Alle FIXTconstanten \geq 32768 en alle FLOTconstanten in een opdracht moeten worden geadresseerd (als "variabele") en ergens in of achter het objectprogramma worden ingezet.

In principe worden dergelijke constanten behandeld als super-globalen. Het enige verschil ~~is~~ met super-globale operand -namen is, dat ~~de~~ de waarde van de constante op het (de) adres(sen) ingevuld.

Voor de vraag welke constanten in welke opdrachten zijn toegestaan, worden de volgende regels:

a) We onderscheiden:

- 1) enkellengte-opdrachten: alle opdrachten met enkelvoudige geheugenselectie (de G, A, S en B, alsmede alle bestemmings-opdrachten)
 - 2) dubbellengte-opdrachten: alle F-opdrachten met tweevoudige geheugenselectie

b) In enkellengte-opdrachten mogen alleen FIXT-constanten worden gebruikt.

c) In dubbellengte-opdrachten mogen alleen FLOT-constanten worden gebruikt.

EXEMPLARISCH
De quintessens van deze regel is, dat niet alleen de "undisturbed-F-opdrachten" met "G" worden genoteerd, maar ook de "absoluut-F-opdrachten" (die tot nu toe abusievelijk met F werden genoteerd, wat vele misverstanden met zich mee heeft gebracht).

viii

A = 12 A = 120000 G = 12 G = 120000
F = 12.0 F = 12.4

fout is:

$$F = 12 \quad F = 120000$$

$$A = 12.0 \quad A = 120000.0$$

fountain is open.

A = 120000000 want "120000000" is geen FIXTconstante
(overschrijding van de enkellengte-capaciteit)
F = 120000000 want "120000000" is evenmin een FLOTconstante
(er staat geen "." of ";" in)
120000000 is geen ELANconstante

Een (welkom) consequentie van bovenstaande regels is, dat we ook niet meer schrijven:

$$F = \star pp$$

maari:

G = :PP wat in verband met de notatie "MG" ook veel natuurlijker is.

OPM De enige anomalie in deze afspraak is, dat G-opdrachten met een FIXT-constante < 32768 of met adresoperator ":" wel conditievolgend mogen zijn (in tegenstelling tot alle andere G-opdrachten). Gezien echter het feit dat dit een extra is en geen valkuil, lijkt de anomalie overkomelijk!

16. MACRO'S

Een macro is een (ELAN-)tekst die bij declaratie niet wordt geëvalueerd, maar als "string" wordt gebufferd in de assemblage-stapel.

Eerst bij aanvraag wordt een dergelijke macro-buffer gelezen en geëvalueerd. Daarbij worden bepaalde, als formelen aangewezen, namen vervangen door hun actualisering.

We beschouwen afzonderlijk:

a) Macro-declaratie:

In een declaratielijst kondigt de woorddelimiter 'MACRO' een macro-declaratie aan. Deze bestaat uit een naam (de macro-naam) eventueel gevolgd door tussen haken een lijst van formele namen, gevolgd door een ":" gevolgd door een ongelabeld blok: de macro-body die in de ~~gebruikte~~ macro-buffer wordt opgeslagen, inclusief de delimiters 'BEGIN' en 'END'.

Een macro-body wordt geacht bij aanvraag een ELAN-tekst te zijn, bij declaratie is dit niet noodzakelijk.

De macro-body wordt als volgt gebufferd:

- 1) alle spaties worden weggelaten,
- 2) alle "comments" worden weggelaten,
- 3) alle samengestelde delimiters worden in hun interne representatie gebufferd.

b) Macro-aanvraag:

Een macro-aanvraag is een programma-instructie bestaand uit de macro-naam gevolgd door tussen haken evenveel actualiseringen als bij declaratie formele namen werden genoteerd.

Een dergelijke macro-aanvraag heeft het volgende effect:

De leesroutine van de assembler gaat de macro-buffer lezen. Elke formele naam wordt hierbij vervangen door z'n actualisering. De aldus gelezen tekst wordt als ELAN-tekst geëvalueerd. Wanneer de macro-buffer leeggelezen is, keert de leesroutine terug in de toestand van voor het lezen uit deze macro-buffer.

c) Formal-Actual correspondence

Bij aanvraag mag op de plaats van een formele-naam-bij-declaratie een actualisering worden geschreven.

Een actualisering mag zijn:

- 1) een variabele: een niet-geïndiceerde naam (die onbekend mag zijn)
 een geïndiceerde naam (die bekend moet zijn)
- 2) een adres: een ":" gevolgd door een variabele (als onder 1)
- 3) een constante (zie 15)
(voor 1 t/m 3 zie 10)
- 4) een alfanumerieke string (zie 13)
- 5) een macro-aanvraag (zie punt b) hierboven)

OPM De macro-faciliteit heeft (niet geheel toevallig) enkele punten van overeenkomst met de ALGOL-procedure.

Het is derhalve nuttig de volgende verschilpunten goed te zien:

- 1) Een macro-body hoeft geen ELAN-tekst te zijn,
- 2) de namen in een macro-body krijgen hun betekenis pas bij aanvraag,
- 3) de formal-actual correspondence vindt plaats tijdens assemblage,
- 4) de macro-faciliteit is in wezen een speciale vorm van string-handling.

16. MACRO'S (vervolg)vb:

```

'BEGIN' ALFA, BETA, GAMMA,
  'MACRO' REP1P (X):
  'BEGIN'
    '554400000'(X)
  'END';

ALFA:  "
  " stuk programma
  "
  'BEGIN' 'MACRO' REPETE ( JOB, NTIMES ) :
  'BEGIN' LL,
    'MACRO' SETCOUNT1 ( R, N ) :
    'BEGIN'
      R = N
      COUNT[-1] = R
    'END';
    SETCOUNT1 ( S, NTIMES )
  LL:   JOB
        REP1P ( :LL )
  'END' REPETE,
  'MACRO' SCATTER ( LABEL, ITEM ) :
  'BEGIN'
    A = ITEM
    LUA ( 1 )
    A + LABEL
    DOS ( MA )
  'END' SCATTER ;

  " voortzetting van het programma:

BETA:  REPETE ( SCATTER (:GAMMA, COUNT[-1]), 24 )
"
"
"
GAMMA:  "
  "
  " etc.

```

Bij de aanvraag achter "BETA" wordt geassembleerd:

```

S = 24
COUNT[-1] = S
LL: A = COUNT[-1]
LUA (1)
A + :GAMMA
DOS (MA)
'554400000' ( :LL )

```

OPM Dit voorbeeld wil niet anders zijn dan een demonstratietje-in-kort-bestek van wat oa. mag. De perspectieven voor de praktijk vormen een heel ander chapiter.

17. Varianten

De ELAN-opdrachten (zie 7) kunnen worden onderscheiden in vijf (elkaar gedeeltelijk overlappende) klassen:

- 1) opdrachten die U, Y, en N, conditie kunnen volgen
- 2) opdrachten die Y, en N, conditie kunnen volgen
- 3) opdrachten die geen conditie kunnen volgen
- 4) opdrachten die ,P ,Z en ,E conditie kunnen zetten
- 5) opdrachten die geen conditie kunnen zetten.

De ELAN-programmeur moet eenvoudig weten (en liefst ook nog snel kunnen naslaan) welke "varianten" bij welke opdrachten zijn toegestaan.

Het begrip "opdracht" is echter minder eenvoudig als uit 7) zou kunnen worden geconcludeerd:

- a) een inhouds-replica zou men het liefst willen beschouwen als een "opdracht"

vb: RETURN: GOTOR (MC[-1])

een programmeur die "RETURN" schrijft, bedoelt "GOTOR (MC[-1])"

- b) een macro-aanvraag is te beschouwen als een samengestelde "opdracht" aan de machine

vb: OPEN (TARE1 , FLEXI , :ALARM , ' ('externe naam infostroom')')

De vraag doet zich voor, of het zin heeft bij dergelijke opdrachten-inruimere-betekenis varianten (i.h.b. conditie-volging) toe te staan, en zo ja: op welke wijze dubbelzinnigheden en dwaze constructies vermeden kunnen worden. Hieronder een mogelijkheid die vrij direct realiseert wat men zou willen en niet moeilijk te implementeren lijkt:

We onderscheiden twee soorten programma-instructies (zie 7):

- I) opdrachten (nu in ruimere zin): genoteerd met operator

genoteerd als functie (met geen, een of meer parameters)
(hieronder vallen nu ook de inhouds-replica en de macro-aanvraag)

- II) informatie-eenheden: getallen

assemblages van replica's (zie 10)
strings

We omhelzen de volgende regel:

- A) Bij opdrachten mogen varianten worden geplaatst tenzij ~~deze~~ deze variant in de opdracht reeds "bezett" is (zoals conditievolging in een Gopdracht, conditiezetting in een bestemmingsopdracht etc)
- B) Varianten bij informatie-eenheden worden ~~ge~~ genegeerd.

vb: AMA: '670000400'
YAMA: '670200400'

nu is: N, AMA het toevoegen van een N-conditievolging aan "AMA"
maar: N,YAMA is fout want de UYN-bits zijn al bezet.

vb: 'MACRO' FUN (X):
'BEGIN'
 F = X ; SUBC (:ROUTINE FUN)
'END'

de macro-aanvraag: Y, FUN (X) betekent: Y, F = X ; Y, SUBC (:ROUXFUN)

OPM indien een der opdrachten in de macrobody conditie-~~ge~~ggend was, zou alarm worden geslagen,
indien informatie-eenheden in de macrobody zouden staan, zou de conditievolging hierop (regel B) geen effect hebben:
dit is precies wat men eigenlijk bedoelt.

18. Varianten, notatie

De notatie der varianten: U, , P
Y, <opdracht>, Z
N, , E

is niet erg gelukkig verzonnen: het geeft enige ~~problemen~~ problemen bij de implementatie, maar het geeft vooral aanleiding tot dubbelzinnigheden:

vb:

XMEMORY

U: '020000000'

P: '120000000'

nu kan U, P betekenen: U, S = M
maar ook: A = M, P

We hebben de keuze tussen twee vervelende alternatieven:

- 1) een aparte regel opstellen die bovengesigneerde dubbelzinnigheid uit de taal verwijdert (noodzakelijk een regel die bepaalde namen in een uitzonderingspositie plaatst)
- 2) de notatie van de varianten wijzigen