stichting
mathematisch
centrum

MC

LR 1.2

APRIL 1971

H.L. OUDSHOORN

BITMANIPULATIE-PROCEDURES

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.

0.1. Lijst van pagina's

Om na te gaan of een exemplaar al dan niet volledig is, kan gebruik worden gemaakt van onderstaande tabel.

Hoofdstuk 0	bestaat uit	2 blz.
Hoofdstuk 1	bestaat uit	1 blz.
Hoofdstuk 2	bestaat uit	3 blz.
Hoofdstuk 3	bestaat uit	6 blz.
Hoofdstuk 4	bestaat uit	6 blz.
Hoofdstuk 5	bestaat uit	1 blz.
Hoofdstuk 6	bestaat uit	1 blz.

Deze blz. zal bij elke aanvulling worden vervangen.

0.2. Inhoudsopgave

1. Inleiding

Hieronder volgt de beschrijving van een aantal kodeprocedures die de programmeur in staat stellen te manipuleren met rijen van 27 bits. De keuze van dit getal is bepaald door de 27 bits woordlengte van de Electrologica X8.

Bij de samenstelling van deze set procedures werd uitgegaan van de bij de T.H. Eindhoven bestaande procedures (zie RC 3615, inf. 30-15/16).

Alhoewel het voor een programmeur die onderstaande procedures zal gebruiken niet strikt noodzakelijk is, zal toch in hoofdstuk 2. een beschrijving worden gegeven van de getalvoorstelling in de EL X8. Dit enerzijds voor een volledig begrip van de in hoofdstuk 4. gegeven ekwivalente ALGOL 60 procedures, anderzijds om een optimaal gebruik van de kodeprocedures te kunnen maken. Dit optimale gebruik zal namelijk in de meeste gevallen liggen in het juist kombineren van de bekende aritmetische operaties en de hier beschreven kodeprocedures.

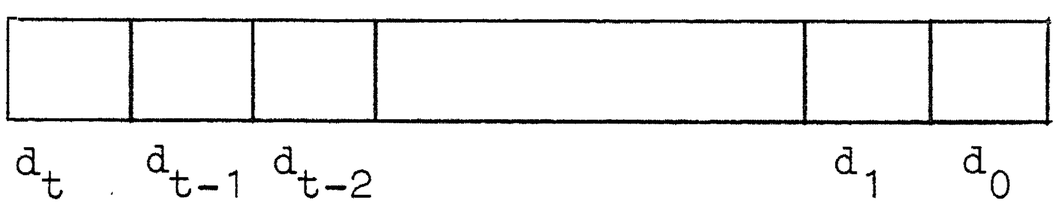
Deze kodeprocedures zitten in de procedurebibliotheek van de EL X8 van het Mathematisch Centrum. Zij kunnen dus zonder declaratie worden gebruikt.

2. Iets over de getalvoorstelling en de aritmetiek van de EL X8.

2.1. Ones' complement system (OCS).

OCS is de methode voor de representatie van gehele getallen welke gebruikt wordt bij de EL X8.

Stel dat er t+1 bits beschikbaar zijn voor de representatie van een geheel getal. De afzonderlijke bits zullen we aangeven met $d_0, d_1, \ldots, d_t$.



Voor positieve getallen geldt:

i) $d_t = 0$ (tekenbit),

ii) het deel $d_{t-1}, \ldots, d_0$ vormt de binaire representatie van het getal.

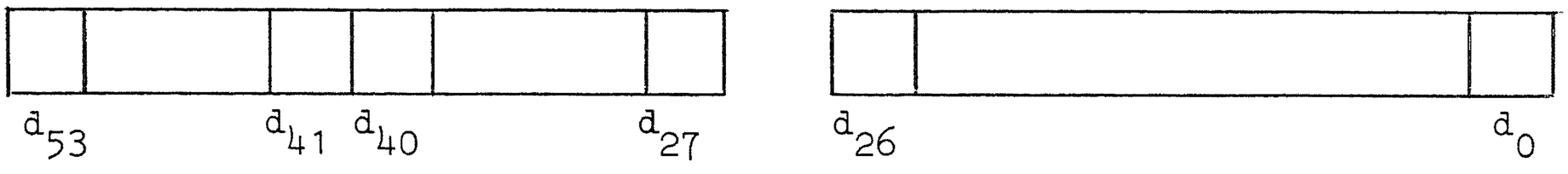
In OCS ontstaat de tekeninverse van een getal als bitinverse van dat getal, d.w.z. door elke 0 door een 1 en elke 1 door een 0 te vervangen. Een gevolg hiervan is dat er twee representaties van het getal nul bestaan, namelijk zowel de nullenrij ($d_i = 0$ voor $0 \leq i \leq t$) als de enenrij ($d_i = 1$ voor $0 \leq i \leq t$). We zullen dit onderscheid, waar nodig, aangeven door respektievelijk +0 en -0 te schrijven.

2.2. Integers in de EL X8.

De EL X8 is een binaire machine met 27 bits woordlengte. Een integer beslaat één geheugenwoord. De voorstelling van integers is als beschreven in 2.1. met t = 26.

2.3. Reals in de EL X8.

Een real beslaat in de EL X8 twee opeenvolgende geheugenwoorden (d_0 t/m d_{53}).



Het geheugenwoord dat $d_{53}, \ldots, d_{27}$ bevat noemen we de kop, het andere de staart van het real getal. Het geheugenadres van de staart is één hoger dan dat van de kop.
Een real getal x is opgebouwd uit twee gedeelten:
i) de mantisse m
en
ii) de exponent e,
zo dat $x = m * 2^e$.
In de bits $d_{41}, d_{40}, \ldots, d_{27}, d_{25}, \ldots, d_0$ staat de waarde van m in OCS, waarbij d_{41} de rol van tekenbit vervult. d_{26} is een kopie van d_{41}.
In de bits $d_{53}, \ldots, d_{42}$ staat de waarde van e^* in OCS.
Hierbij fungeert d_{53} als tekenbit en geldt verder

$$e^* = \begin{cases} e \text{ als } d_{41} = d_{26} = 0 \\ -e \text{ als } d_{41} = d_{26} = 1. \end{cases}$$

2.4. Iets over de aritmetiek van de EL X8.

De aritmetiek van de EL X8 is -0 - preferent:
een optelling a + b die 0 geeft levert als resultaat -0, tenzij beide operanden +0 waren,
een aftrekking a - b die 0 geeft levert als resultaat -0, tenzij a = +0 en b = -0.
Voor vermenigvuldiging, deling en de operatie ÷ gelden (ook voor de getallen +0 en -0) de normale tekenregels.
Voor de operatie ↑ geldt: (+0) ↑ a = (-0) ↑ a = +0 voor alle $a \neq 0$.

Onder andere als gevolg van de wijze waarop de standaardfunkties in het MC ALGOL 60 systeem zijn gerealiseerd merken we nog de volgende punten op:

i) wanneer door een assignment van een real aan een integer afronding optreedt met resultaat 0, dan is dat altijd -0,
ii) entier (a) met resultaat 0 geeft altijd +0, tenzij a = -0,
iii) sqrt (+0) = sqrt (-0) = +0,
iv) sin (+0) = sin (-0) = +0,
v) arctan (+0) = -0 en arctan (-0) = +0,
vi) ln (1) = -0.

3. Beschrijving van de kodeprocedures.

In deze beschrijving zullen we rijen van 27 bits voorstellen door integer arrays D a[0:26], D b[0:26] enz.. De elementen van deze arrays kunnen slechts de waarden 0 of 1 aannemen.

Aangezien de waarde van een variabele van type integer in de EL X8 één geheugenwoord van 27 bits bezet, kunnen we een één-éénduidig verband leggen tussen het array D a[0:26] en een variabele a van type integer. In het vervolg zullen we steeds stilzwijgend aannemen dat de variabele a korrespondeert met het array (de bitrij) D a, b met D b, and met D and, enz.. Het element D a[0] zal steeds korresponderen met bit d_0 van a, D a[1] met d_1 enz., aansluitend op de in 2. gegeven getalvoorstelling in de EL X8.

Van de lezer wordt in dit hoofdstuk, met betrekking tot de exakte formulering, een ruime mate van soepelheid gevraagd. Zo zal in plaats van "de waarde van de aritmetische expressie c van type integer" wel eens gesproken worden over "het getal c", terwijl tevens zowel sprake zal zijn van "het i-de element van D a" als van "het i-de bit van a" of ook "d_i (van a)". Dit alles niet met het doel de verwarring te vergroten, maar slechts als poging de leesbaarheid enigszins te bevorderen. Een gunstig neveneffekt zou tevens kunnen zijn dat de lezer de beide opvattingen, nl.:

i) het werken met rijen van 27 willekeurige (array) elementen die slechts de waarden 0 en 1 kunnen aannemen

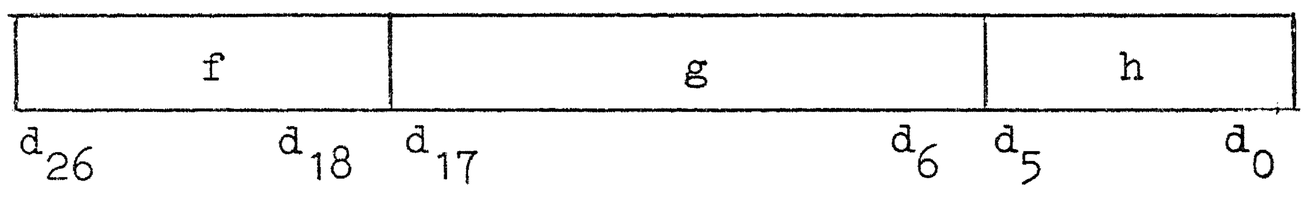
en

ii) het werken met de 27 bits waaruit de binaire voorstelling van een integer in de EL X8 bestaat

naast elkaar en door elkaar leert gebruiken. Slechts dan zal namelijk een optimaal gebruik van deze procedures mogelijk zijn.

Met nadruk zij er op gewezen dat de 27 elementen van, bijvoorbeeld, D a (ofwel de 27 bits van a) in de hierna te geven procedures volkomen gelijkwaardig zijn. D a[26] vervult beslist géén bijzondere rol: De korrespondentie van D a[26] met het tekenbit van a komt slechts

naar voren wanneer we de rij van 27 bits waarmee we werken werkelijk als een getal in OCS gaan interpreteren, iets wat in de meeste gevallen niet de bedoeling zal kunnen zijn. De programmeur is uiteraard wel in staat aan bepaalde bits een specifieke funktie toe te kennen. Hij kan bijvoorbeeld de 27 bits van één X8 geheugenwoord in 3 velden van respektievelijk 9, 12 en 6 bits verdelen en hierin drie gehele getallen, f, g en h opbergen:



Vanzelfsprekend moet dan wel voldaan zijn aan:

$$0 \leq f \leq 511,\ 0 \leq g \leq 4095,\ 0 \leq h \leq 63.$$

In zo'n geval zullen we in het vervolg spreken over veld (26:18), veld (17:6) en veld (5:0).

3.1. integer procedure and (a,b); value a, b; integer a, b;

De waarde van D and[i] blijkt uit de volgende tabel:

D a[i] / D b[i]	0	1
0	0	0
1	0	1

Vergelijk de truth-table behorend bij de logische operator $\wedge$.

3.2. integer procedure or (a,b); value a, b; integer a, b;

De waarde van D or[i] blijkt uit de volgende tabel:

D a[i] / D b[i]	0	1
0	0	1
1	1	1

Vergelijk de truth-table behorend bij de logische operator V.

3.3. integer procedure xor (a,b); value a, b; integer a, b;

De waarde van D xor[i] blijkt uit de volgende tabel:

D b[i] \ D a[i]	0	1
0	0	1
1	1	0

xor is de 'afkorting' van exclusive or.

3.4. integer procedure bit (j,a); value j, a; integer j, a;

bit krijgt de waarde van bit d_j van a.

Voor j moet gelden: $0 \leq j \leq 26$.

Is niet aan deze voorwaarde voldaan, dan wordt een foutmelding gegeven met foutnummer 532.

3.5. integer procedure bitstring (u,l,a); value u, l, a;
integer u, l, a;

Als u = 26 én l = 0 dan is D bitstring een kopie van D a, d.w.z. de waarde van bitstring (26,0,a) is gelijk aan de waarde van a. In alle andere toegestane gevallen krijgt *bitstring* de waarde van veld (u:l) van a, i.e. de waarde van de deelrij D a[u], D a[u-1], ..., D a[l], opgevat als positief binair getal.

Er moet gelden: $26 \geq u \geq l \geq 0$.

Is niet aan deze voorwaarde voldaan, dan wordt een foutmelding gegeven met foutnummer 532.

3.6. integer procedure set (c,u,l,a); value c, u, l, a;
integer c, u, l, a;

Deze procedure zou men de inverse van *bitstring* (zie 3.5.) kunnen noemen. Terwijl *bitstring* gebruikt kan worden om de waarde van veld (u:l) van a uit te lezen, krijgt *set* juist de waarde die korrespondeert met de bitrij van a, waarin het oorspronkelijke veld (u:l) vervangen is door het bitpatroon van het getal c.

Er moet gelden:

i) $26 \geq u \geq l \geq 0$,

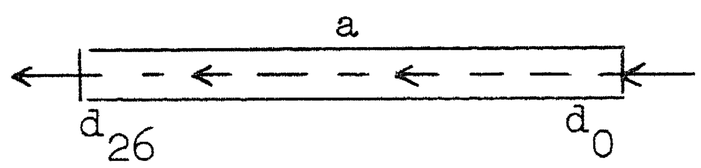
ii) indien niet geldt u = 26 én l = 0:
$0 \leq c \leq 2^{u-l+1}$.
Het zal duidelijk zijn dat in dit geval alleen positieve getallen c in aanmerking komen. Daarom geldt hierbij steeds dat wanneer c = 0, ongeacht of dat +0 of -0 is, het veld (u:l) met nullen wordt gevuld.

Is niet aan i) voldaan, dan wordt een foutmelding gegeven met foutnummer 532; wanneer wel aan i), maar niet aan ii) voldaan is, dan wordt een foutmelding gegeven met foutnummer 533.

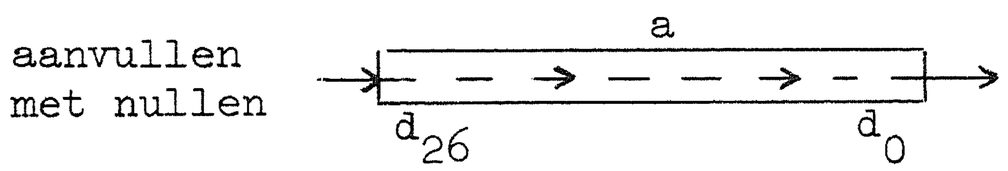
3.7. integer procedure clear shift (a,j); value a, j; integer a, j;

Onder 'schoonschuiven' van a verstaan we een verschuiving van de elementen van het array D a, waarbij aan de ene kant elementen verdwijnen en aan de andere kant nullen worden binnengeschoven. Denkt men zich D a[26] als meest links en D a[0] als meest rechts element, dan geldt:
wanneer $j \geq 0$ dan wordt over j plaatsen naar links geschoven,
wanneer $j < 0$ dan wordt over -j plaatsen naar rechts geschoven.

$j \geq 0$:



$j < 0$:



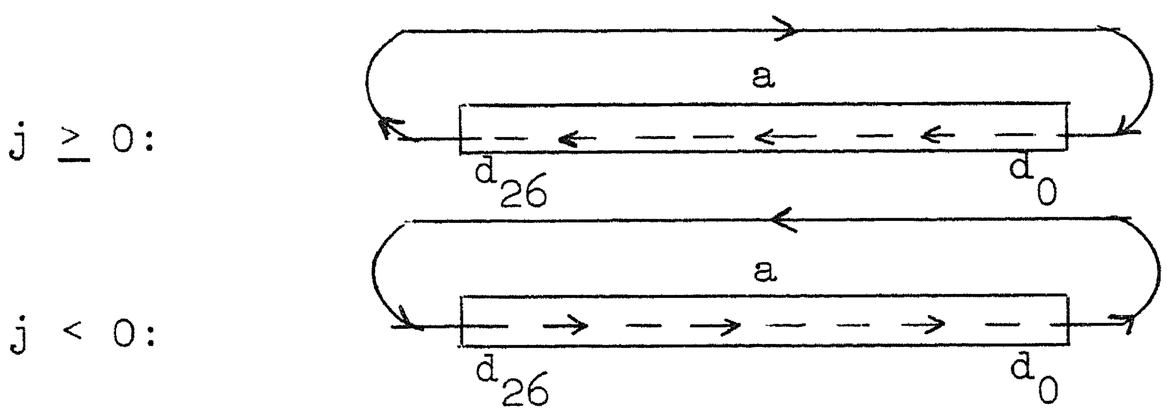
clear shift krijgt de waarde die a zou hebben na deze verschuiving. Voor j moet gelden: $-26 \leq j \leq 26$.
Is niet aan deze voorwaarde voldaan, dan wordt een foutmelding gegeven met foutnummer 532.

Opm.: Wanneer men niet met nullen, maar met kopieen van het tekenbit van a wil aanvullen (waardoor deze operatie gelijk wordt aan $a * 2^j$ als $j \geq 0$ -wanneer tenminste het resultaat binnen de integerkapaciteit blijft- en $a \div 2^{-j}$ als $j < 0$) dan is dit te verkrijgen door:

```
sign (a) * clear shift (abs (a),j).
```

3.8. integer procedure circ shift (a,j); value a, j; integer a, j;

Bij 'rondschuiven' denke men zich de bitrij als een gesloten circuit, waarbij dus D a[26] en D a[0] naast elkaar liggen.
Analoog aan 3.7. geldt:
wanneer j ≥ 0 dan wordt over j plaatsen naar links rondgeschoven,
wanneer j < 0 dan wordt over -j plaatsen naar rechts rondgeschoven.



circ shift krijgt de waarde die a zou hebben na deze verschuiving.
Voor j moet gelden: -26 ≤ j ≤ 26.
Is niet aan deze voorwaarde voldaan, dan wordt een foutmelding gegeven met foutnummer 532.

3.9. integer procedure norm shift (a,normed a); value a;
integer a, normed a;

Onder 'normeren' van a verstaan we naar links rondschuiven van a totdat D a[26] ≠ D a[25].
norm shift krijgt als waarde het aantal plaatsen waarover geschoven moet worden om dit te bereiken. Wanneer a de waarde +0 of -0 heeft, dus wanneer alle elementen van D a dezelfde waarde hebben, krijgt *norm shift* de waarde 26.
normed a is een output-parameter waarin de waarde afgeleverd wordt die a na normering zou hebben.

Hierna volgen nog drie procedures die betrekking hebben op real getallen:

3.10. integer procedure head of (x); value x; real x;

head of krijgt de waarde van de kop van x (zie 2.3.), dus:
d_i van *head of* = d_{i+27} van x ($0 \leq i \leq 26$).

3.11. integer procedure tail of (x); value x; real x;

tail of krijgt de waarde van de staart van x (zie 2.3.), dus:

d_i van *tail of* = d_i van x ($0 \leq i \leq 26$).

3.12. real procedure compose (a,b); value a, b; integer a, b;

De waarde van *compose* is die van het real getal waarvan het bitpatroon van de kop een kopie is van het bitpatroon van a en het bitpatroon van de staart een kopie is van dat van b.
Teneinde aan de in 2.3. gegeven getalvoorstelling van een real getal te voldoen moet gelden:

d_{14} van a = d_{26} van b.

Is niet aan deze voorwaarde voldaan, dan wordt een foutmelding gegeven met foutnummer 531.

4. Ekwivalente ALGOL 60 procedures.

Allereerst volgen hier drie hulpprocedures waarvan de eerste twee, *comp* en *decomp*, slechts dienen ter verbetering van de leesbaarheid van de beschrijving der kodeprocedures in ALGOL 60. De derde, *errormessage*, is opgenomen om te voldoen aan de syntaktische korrektheid van het geheel, en kan worden gezien als een beschrijving in ALGOL 60 van een routine uit het MC ALGOL 60 systeem voor de EL X8.

Ter verduidelijking moge dienen dat de statement

```
pos:= 1/a > 0
```

gebruikt wordt om het teken van het getal a vast te stellen omdat hierdoor ook +0 en -0 van elkaar worden onderscheiden. Verder wordt in de procedures *head of*, *tail of* en *compose* gebruik gemaakt van de bibliotheekprocedures *to drum* en *from drum*. Zie voor een beschrijving hiervan "Handleiding Milli-systeem voor de EL X8" (LR 1.1).

```
procedure decomp (a) into: (D a); value a; integer a; integer array D a;
begin   comment verzorgt de transformatie van integer naar bitrij;
        integer i, absa;
        boolean pos;
        pos:= 1 / a > 0; absa:= abs (a);
        for i:= 0 step 1 until 25 do
        begin   if absa : 2 × 2 = absa
                then D a[i]:= if pos then 0 else 1
                else D a[i]:= if pos then 1 else 0;
                absa:= absa : 2
        end;
        D a[26]:= if pos then 0 else 1
end decomp;

integer procedure comp (D a); integer array D a;
begin   comment verzorgt de transformatie van bitrij naar integer;
        integer i;
        comp:= sum (i, 0, 25,
                    (if D a[26] = 0 then D a[i] else D a[i] - 1) × 2 ↑ i)
end compose;

procedure errormessage (n); value n; integer n;
begin   nlcr; nlcr; printtext (<er>); absfixt (3, 0, n);
        exit
end errormessage;
```

```
integer procedure and (a, b); value a, b; integer a, b;
begin    comment zie 2.1.;
         integer i;
         integer array D a, D b, D and[0:26];
         decomp (a) into: (D a);
         decomp (b) into: (D b);
         for i:= 0 step 1 until 26 do
         D and[i]:= if D a[i] = 0 then 0 else D b[i];
         and:= comp (D and)
end and;

integer procedure or (a, b); value a, b; integer a, b;
begin    comment zie 2.2.;
         integer i;
         integer array D a, D b, D or[0:26];
         decomp (a) into: (D a);
         decomp (b) into: (D b);
         for i:= 0 step 1 until 26 do
         D or[i]:= if D a[i] = 1 then 1 else D b[i];
         or:= comp (D or)
end or;

integer procedure xor (a, b); value a, b; integer a, b;
begin    comment zie 2.3.;
         integer i;
         integer array D a, D b, D xor[0:26];
         decomp (a) into: (D a);
         decomp (b) into: (D b);
         for i:= 0 step 1 until 26 do
         D xor[i]:= if D a[i] = D b[i] then 0 else 1;
         xor:= comp (D xor)
end xor;

integer procedure bit (j, a); value j, a; integer j, a;
begin    comment zie 2.4.;
         integer array D a[0:26];
         if j < 0 ∨ j > 26 then errormessage (532);
         decomp (a) into: (D a);
         bit:= D a[j]
end bit;
```

```
integer procedure bitstring (u, l, a); value u, l, a; integer u, l, a;
begin    comment zie 2.5.;
         integer i;
         integer array D a, D bitstring[0:26];
         if u > 26 ∨ l < 0 ∨ u < l then errormessage (532);
         decomp (a) into: (D a);
         for i:= 26 step -1 until u - l + 1 do D bitstring[i]:= 0;
         for i:= u - l step -1 until 0 do D bitstring[i]:= D a[i + l];
         bitstring:= comp (D bitstring)
end bitstring;

integer procedure set (c, u, l, a); value c, u, l, a;
integer c, u, l, a;
begin    comment zie 2.6.;
         integer i;
         integer array D c, D a, D set[0:26];
         if u > 26 ∨ l < 0 ∨ u < l then errormessage (532);
         decomp (c) into: (D c);
         decomp (a) into: (D a);
         if u = 26 ∧ l = 0
         then begin for i:= 26 step -1 until 0 do D set[i]:= D c[i] end
         else begin if c < 0 ∨ c > 2 ↑ (u - l + 1)
                    then errormessage (533);
                    for i:= 26 step -1 until u + 1 do D set[i]:= D a[i];
                    for i:= u step -1 until l do
                    D set[i]:= if c = 0 then 0 else D c[i - l];
                    for i:= l - 1 step -1 until 0 do D set[i]:= D a[i]
              end;
         set:= comp (D set)
end set;
```

```
integer procedure clear shift (a, j); value a, j; integer a, j;
begin    comment zie 2.7.;
         integer i, absj;
         integer array D a, D clear shift[0:26];
         absj:= abs (j);
         if absj > 26 then errormessage (532);
         decomp (a) into: (D a);
         if j > 0
         then begin for i:= 26 step -1 until j do
                    D clear shift[i]:= D a[i - j];
                    for i:= j - 1 step -1 until 0 do
                    D clear shift[i]:= 0
              end
         else begin for i:= 26 step -1 until 26 - absj + 1 do
                    D clear shift[i]:= 0;
                    for i:= 26 - absj step -1 until 0 do
                    D clear shift[i]:= D a[i + absj]
              end;
         clear shift:= comp (D clear shift)
end clear shift;

integer procedure circ shift (a, j); value a, j; integer a, j;
begin    comment zie 2.8.;
         integer i, absj;
         integer array D a, D circ shift[0:26];
         absj:= abs (j);
         if absj > 26 then errormessage (532);
         decomp (a) into: (D a);
         if j > 0
         then begin for i:= 26 step -1 until j do
                    D circ shift[i]:= D a[i - j];
                    for i:= j - 1 step -1 until 0 do
                    D circ shift[i]:= D a[i - j + 27]
              end
         else begin for i:= 26 step -1 until 26 - absj + 1 do
                    D circ shift[i]:= D a[i + absj - 27];
                    for i:= 26 - absj step -1 until 0 do
                    D circ shift[i]:= D a[i + absj]
              end;
         circ shift:= comp (D circ shift)
end circ shift;

integer procedure norm shift (a, normed a); value a; integer a, normed a;
begin    comment zie 2.9.;
         integer i, k;
         integer array D a[0:26];
         decomp (a) into: (D a);
         k:= 0;
         for i:= 25 step -1 until 0 do
         if D a[26] = D a[i] then k:= k + 1 else goto L;
L:       normed a:= circ shift (a, k);
         norm shift:= k
end norm shift;
```

```
integer procedure head of (x); value x; real x;
begin   comment zie 2.10.;
        integer array I[0:0];
        real array R[0:0];
        R[0]:= x;
        to drum (R, 0);
        from drum (I, 0);
        head of:= I[0]
end head of;

integer procedure tail of (x); value x; real x;
begin   comment zie 2.11.;
        integer array I[0:0];
        real array R[0:0];
        R[0]:= x;
        to drum (R, 0);
        from drum (I, 1);
        tail of:= I[0]
end tail of;

real procedure compose (a, b); value a, b; integer a, b;
begin   comment zie 2.12.;
        integer array I[0:1];
        real array R[0:0];
        if bit (14, a) ≠ bit (26, b) then errormessage (531);
        I[0]:= a; I[1]:= b;
        to drum (I, 0);
        from drum (R, 0);
        compose:= R[0]
end compose;
```

5. De tijden in het MC ALGOL 60 systeem voor de EL X8

De tijden die hieronder gegeven worden gelden bij benadering voor een aanroep van de desbetreffende procedure, wanneer de aktuele parameters konstanten en/of simple variables zijn.

3.1.	and	310 mmsec
3.2.	or	310 mmsec
3.3.	xor	310 mmsec
3.4.	bit	330 mmsec
3.5.	bitstring	605 mmsec
3.6.	set	710 mmsec
3.7.	clear shift	360 mmsec
3.8.	circ shift	335 mmsec
3.9.	norm shift	370 mmsec
3.10.	head of	165 mmsec
3.11.	tail of	165 mmsec
3.12.	compose	315 mmsec.

6. Foutmeldingen

Hieronder volgt een opsomming van de nieuwe foutnummers en een overzicht van de situaties waarin foutmeldingen met deze foutnummers optreden:

531 Bij een aanroep van compose (a,b) geldt niet:
d_{14} van a = d_{26} van b.

532 Deze fout kan optreden:
i) in *bit* wanneer niet geldt: $0 \leq j \leq 26$,
ii) in *bitstring* en *set* wanneer niet geldt: $26 \geq u \geq 1 \geq 0$,
iii) in *clear shift* en *circ shift* wanneer niet geldt: $-26 \leq j \leq 26$.

533 Bij een aanroep van set (c,u,l,a) voldoet c niet aan de voorwaarde dat de binaire representatie moet 'passen' in u-l+1 bits (zie hiervoor 3.6.).