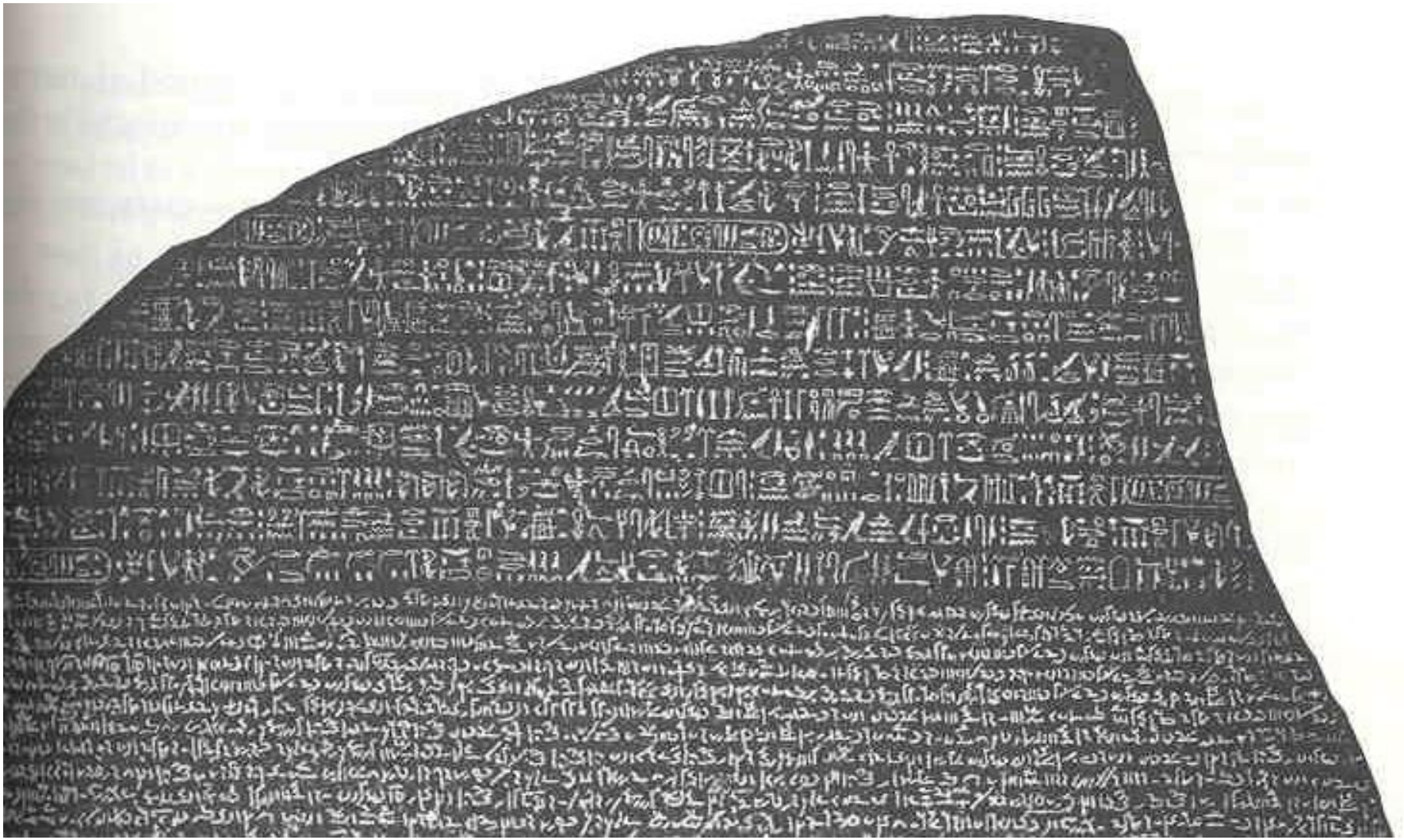
INTENTIONAL

SOFTWARE

# The Rosetta Code: Archeology of the Gier Algol Compiler

Charles Simonyi

# The Rosetta Stone, 196 BC



# MINUTES OF E.C. MEETING #4

Date - November 29, 1945

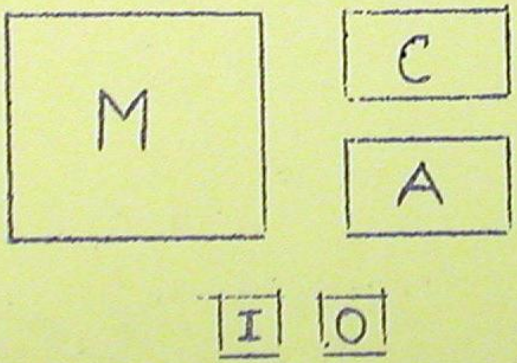
Time - 9:00 A.M.

Place - Office of A. W. Vance

Present: G. W. Brown
J. P. Eckert
H. H. Goldstine
J. von Neumann
J. A. Rajchman
J. W. Tukey
A. W. Vance
V. K. Zworykin

1. Construction of Block Diagram

The whole device, in general arrangement, consists of inputs and outputs, memory, control, and arithmetical parts as shown:




Assume, for this discussion, that we are in the least favorable case,

MINUTES OF THE MEETING OF

THE COMMITTEE ON THE ELECTRONIC COMPUTER

HELD IN THE DIRECTOR'S OFFICE,

OF THE INSTITUTE FOR ADVANCED STUDY

TUESDAY, NOVEMBER 6, 1945

PRESENT: Dean Taylor, Dr. Engstrom, Dr. von Neumann, Dr. Veblen and Dr. Aydelotte.

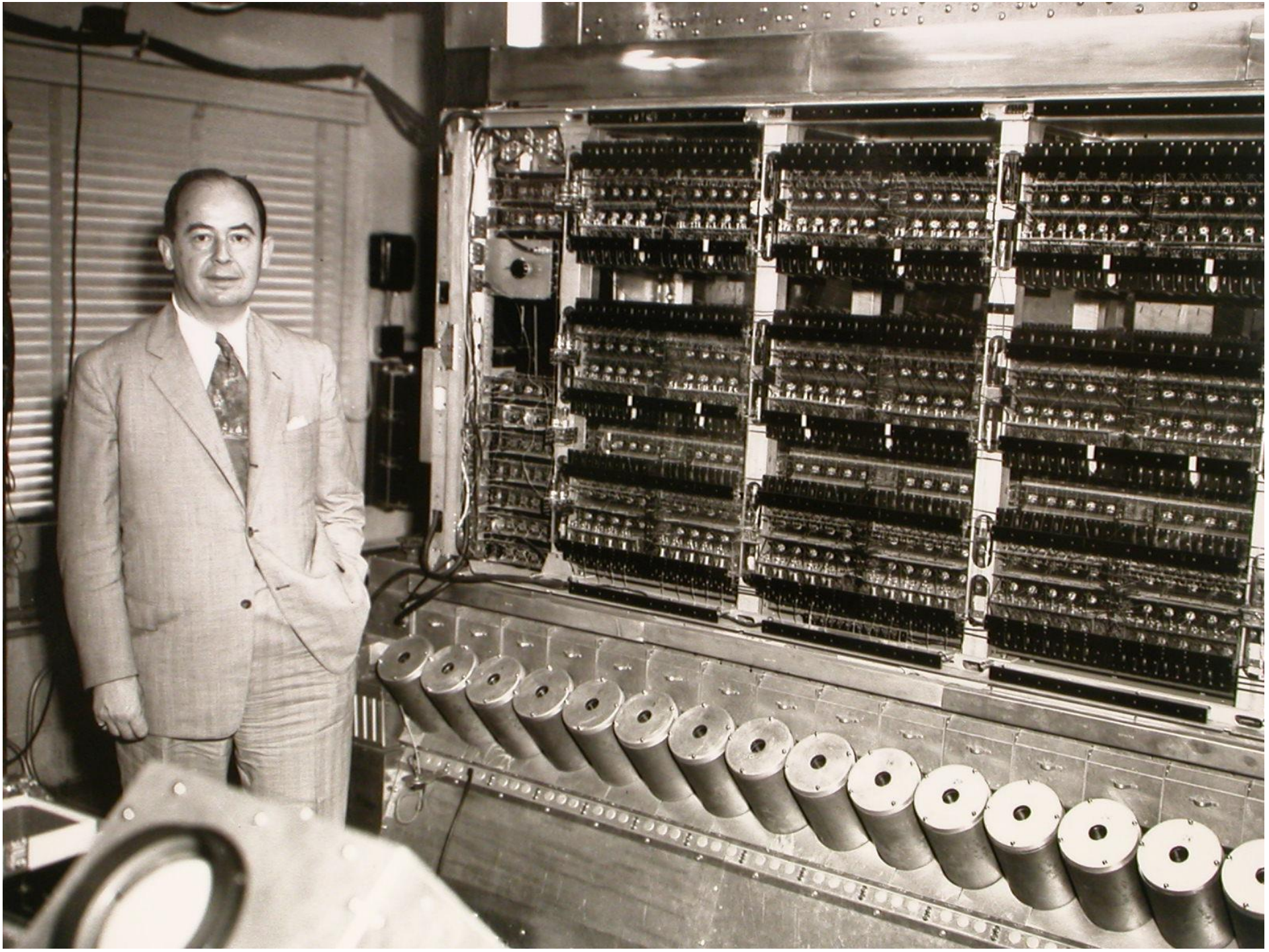
(1) Dr. Aydelotte outlined in some detail the progress which has been made so far in financing the project. The estimated cost is $300,000. The Trustees of the Institute have underwritten the project to the extent of $100.000, the RCA has obligated itself for a similar amount and application has been made to the Rockefeller Foundation to complete the sum. In addition, Dean Taylor assured the

June 5, 1947

Professor John von Neumann
Institute for Advanced Study
Princeton, New Jersey

Dear John:

I am a little troubled about the tea service in the electronic computer building. Apparently the members of your staff consume several times as much supplies as the same number of people do in Fuld Hall and they have been especially unfair in the matter of sugar. Sugar is rationed and for a member of your staff to come up here as Thompson did and carry down a large quantity of sugar in excess of your rations is not cricket. I understand, furthermore, that the tea is served in several different places. We have never undertaken in the Institute to provide tea service in a large number of private offices and I should like to raise the question whether it would not be better for the computer people to come up to Fuld Hall at the end of the day at five o'clock in the afternoon and have their tea here under proper supervision. The only alternative seems to me to be to establish some central place in the computer building and have proper supervision there.



# ORBIT STABILITY

REVISED DIAGRAM

CODE 135.01

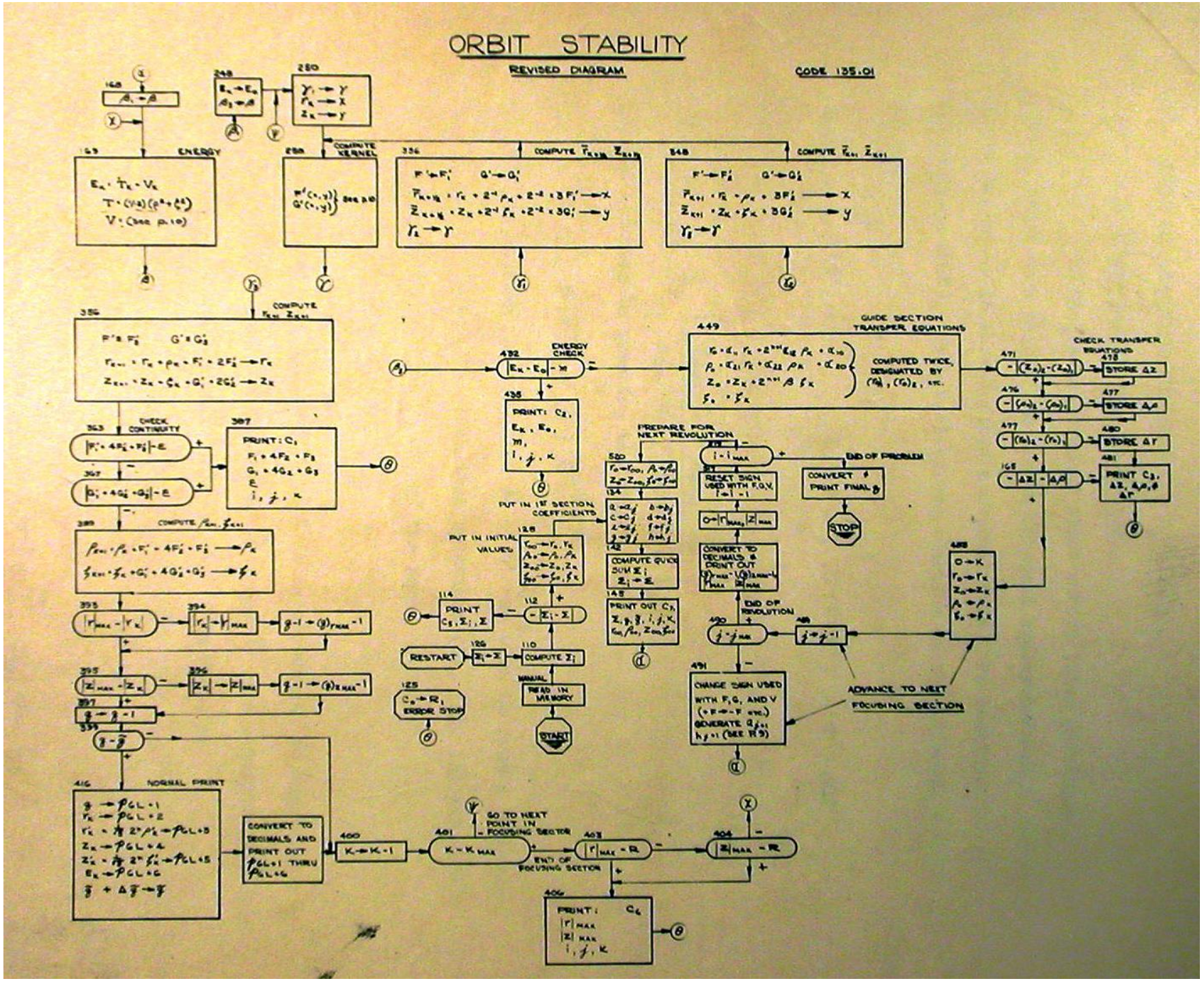


Table 4.2
Computers built using the IAS computer design

| | | |
|---|---|---|
| AVIDAC | Argonne National Laboratory | 1953 |
| BESK | Swedish Board for Computing Machinery, Stockholm | 1953 |
| BESM | Academy of Sciences, Moscow | 1955 |
| DASK | Danish Academy, Institute of Computing Machinery | 1957 |
| GEORGE | Argonne National Laboratory | ? |
| IBM 701 | IBM Corp. | 1952 |
| ILLIAC | University of Illinois | 1952 |
| JOHNNIAC | RAND Corporation | 1954 |
| MANIAC | Los Alamos Scientific Laboratory | 1952 |
| MSUDC | Michigan State University | ? |
| ORACLE | Oak Ridge National Laboratory | 1953 |
| ORDVAC | Aberdeen Proving Grounds | 1952 |
| PERM | Technische Hochschule, Munich | 1954 |
| SILLIAC | University of Sydney | 1956 |
| SMIL | Lund University | 1956 |
| TC-1 | International Telemeter Corp. | 1955 |
| WEIZAC | Weizmann Institute, Rehovoth | 1955 |

DASK

Valves

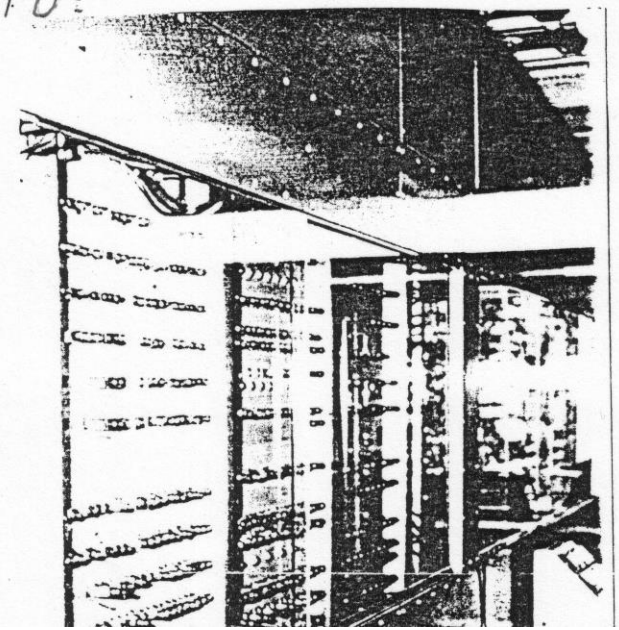
"Home made"

* Copenhagen 29th sep. 57

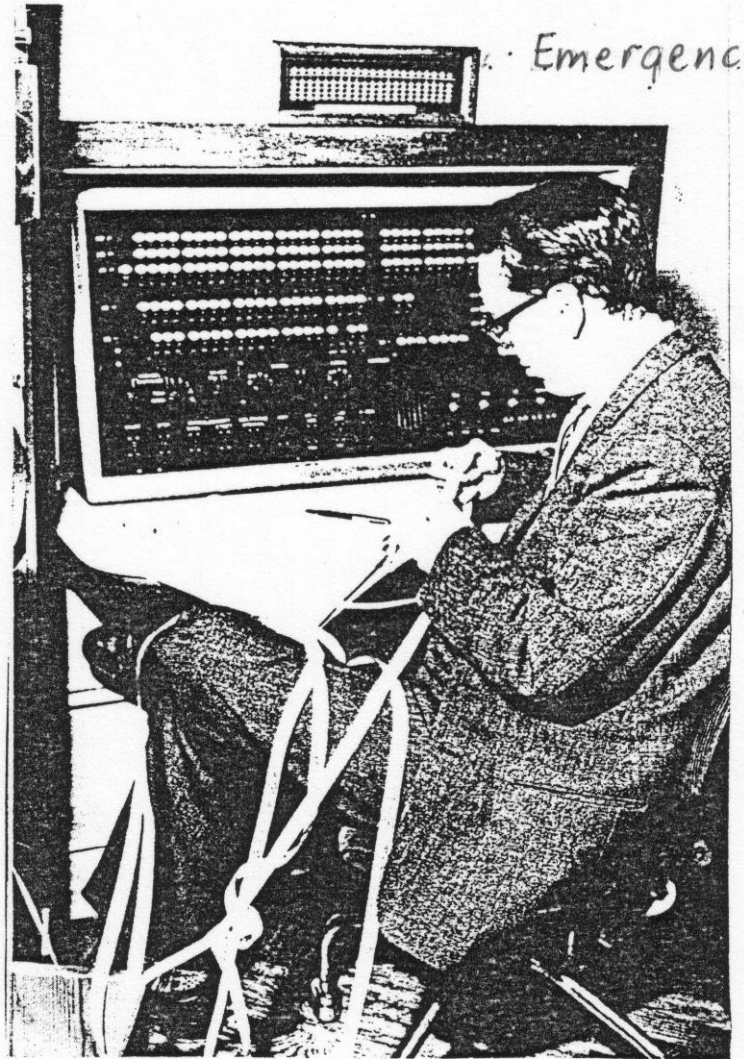
† ? (70?)

Improved "copy" of
BESK, Stockholm:
Index registers
subroutine call.

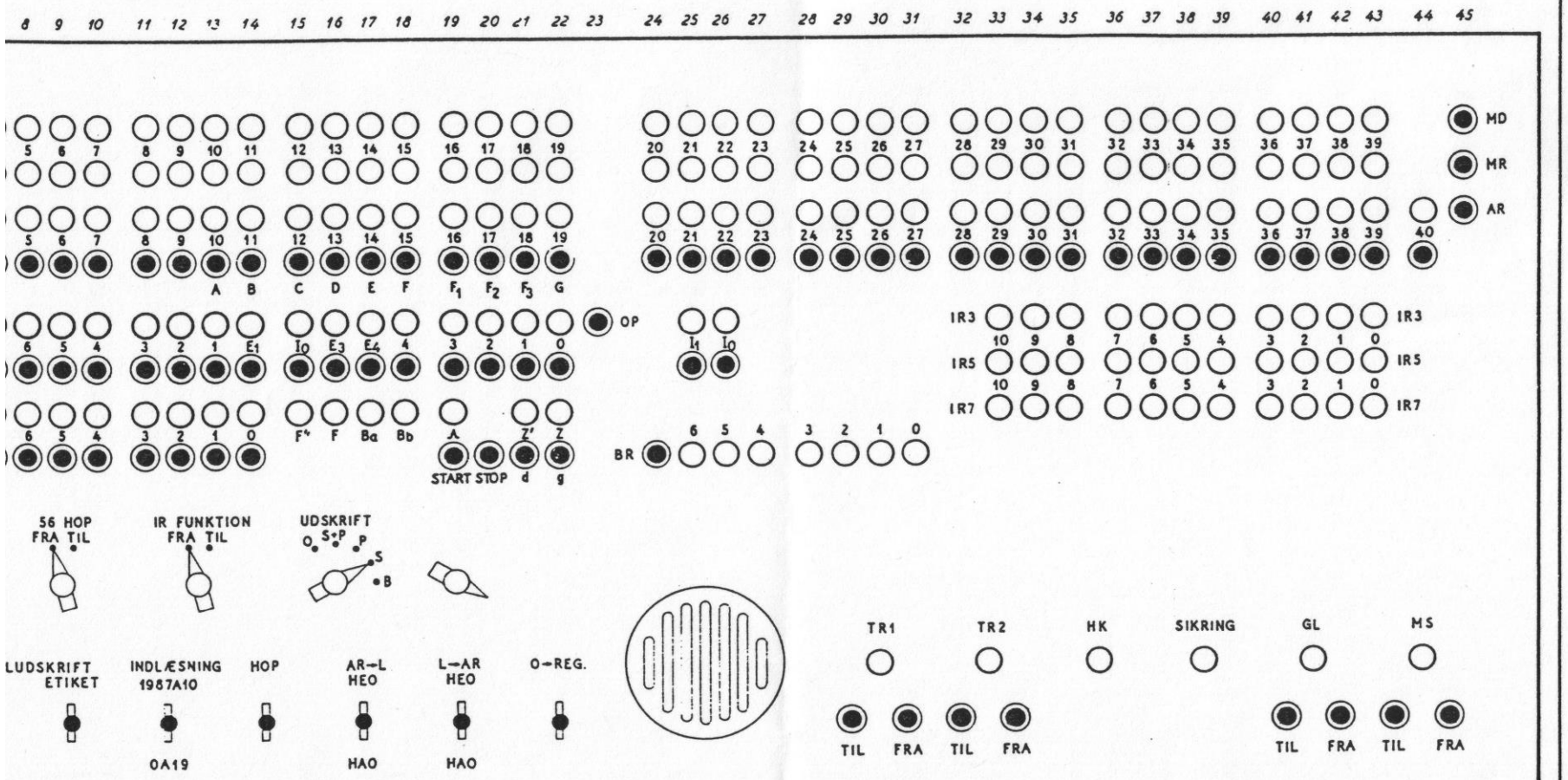
CPU:



DASK



Emergency computer
Abacus




| REGNECENTRALEN Dansk Institut for Matematikmaskiner | Tegnet | T.V. 11.4.59. | | DASK KONTROLBORD | DASK ∧Bm −5 |
|---|---|---|---|---|---|
| | Kontrol | | | | |
| | Godk. | | | | |

Table 4.2
Computers built using the IAS computer design

| | | |
|---|---|---|
| AVIDAC | Argonne National Laboratory | 1953 |
| BESK | Swedish Board for Computing Machinery, Stockholm | 1953 |
| BESM | Academy of Sciences, Moscow | 1955 |
| DASK | Danish Academy, Institute of Computing Machinery | 1957 |
| GEORGE | Argonne National Laboratory | ? |
| IBM 701 | IBM Corp. | 1952 |
| ILLIAC | University of Illinois | 1952 |
| JOHNNIAC | RAND Corporation | 1954 |
| MANIAC | Los Alamos Scientific Laboratory | 1952 |
| MSUDC | Michigan State University | ? |
| ORACLE | Oak Ridge National Laboratory | 1953 |
| ORDVAC | Aberdeen Proving Grounds | 1952 |
| PERM | Technische Hochschule, Munich | 1954 |
| SILLIAC | University of Sydney | 1956 |
| SMIL | Lund University | 1956 |
| TC-1 | International Telemeter Corp. | 1955 |
| WEIZAC | Weizmann Institute, Rehovoth | 1955 |

```
0000      16 0014 4    0040      02 6215 0
   1      02 0000 0       1      22 6156 4
   2      16 0013 0       2      02 6077 0
   3      40 0004 0       3      22 5011 0
   4      21 0003 0       4      02 7526 0
   5      40 0004 0       5      22 6156 4
   6      27 0014 0       6      02 7552 0
   7      25 0003 0       7      16 5500 0

0010      11 1010 0    0050      02 7546 0
   1      40 0004 4       1      16 5501 0
   2      24 0010 0       2      22 7342 4
   3      25 0000 4       3      00 0003 0
   4      25 0000 4       4      14 5062 4
   5     -16 7572 4       5      21 5022 0
   6      27 0013 0       6      02 5064 4
   7      24 0005 0       7      22 6671 0

0020      16 0021 0    0060      02 5501 0
   1      22 7560 0       1      16 7546 0
   2      00 0000 0       2      22 7342 4
   3      00 0000 0       3      00 0002 0
   4      00 0000 0       4      16 5501 0
   5      00 0000 0       5      03 7565 0
   6      00 0000 0       6      21 5032 0
   7      00 0000 0       7      22 6631 4

0030      00 0000 0    0070      22 6570 4
   1      00 0000 0       1      02 6077 0
   2      00 0000 0       2      01 7552 0
   3      00 0000 0       3      30 5500 0
   4      00 0000 0       4      11 0000 4
   5      00 0000 0       5      16 0000 0
   6      25 7534 4       6      02 6776 0
   7     -16 7572 4       7      13 5500 0
```


GIER - DISADEC

# REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 60*

P. NAUER, EDITOR

*Dedicated to the Memory of WILLIAM TURANSKI*

## SUMMARY

The report gives a complete defining description of the international algorithmic language ALGOL 60. This is a language suitable for expressing a large class of numerical processes in a form sufficiently concise for direct automatic translation into the language of programmed automatic computers.

The introduction contains an account of the preparatory work leading up to the final conference, where the language was defined. In addition, the notions, reference language, publication language and hardware representations are explained.

In the first chapter, a survey of the basic constituents and features of the language is given, and the formal notation, by which the syntactic structure is defined, is explained.

The second chapter lists all the basic symbols, and the syntactic units known as identifiers, numbers and strings are defined. Further, some important notions such as quantity and value are defined.

The third chapter explains the rules for forming expressions and the meaning of these expressions. Three different types of expressions exist: arithmetic, Boolean (logical) and designational.

The fourth chapter describes the operational units of the language, known as statements. The basic statements are: assignment statements (evaluation of a formula), go to statements (explicit break of the sequence of execution of statements), dummy statements, and procedure statements (call for execution of a closed process, defined by a procedure declaration). The formation of more complex structures, having statement character, is explained. These include: conditional statements, for statements, compound statements, and blocks.

In the fifth chapter, the units known as declarations, serving for defining permanent properties of the units entering into a process described in the language, are defined.

The report ends with two detailed examples of the use of the language and an alphabetic index of definitions.

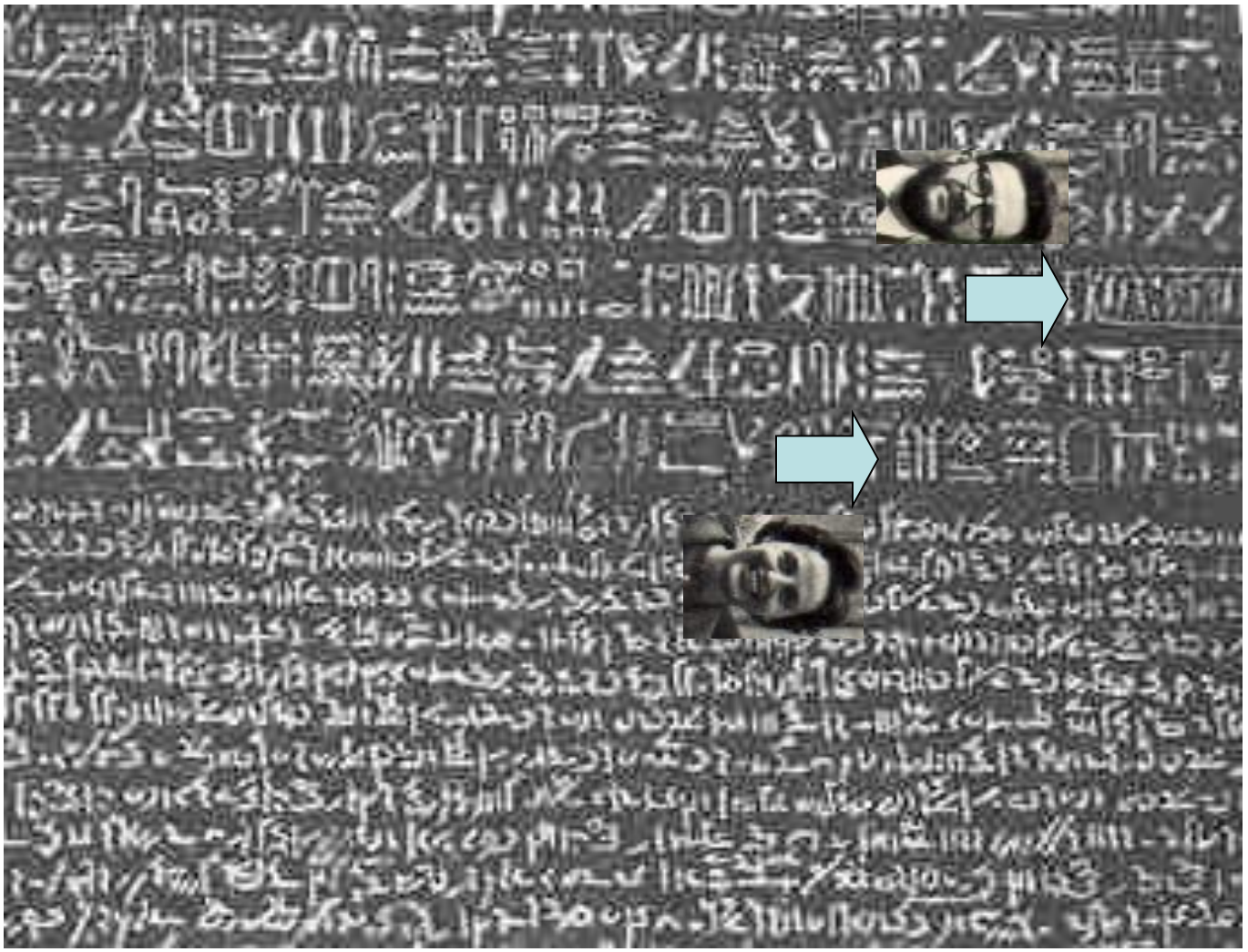
## CONTENTS



*Reprinted from *Comm ACM*, 6, 1, 1963, 1-17, copyright 1963.


A/S REGNECENTRALEN
COMPILER GRUPPEN
SEPTEMBER 1964

```
    pm   a20      V                ;        stackidentifier:= false;
a20:hv   c51      ,   hv  c51      ;    store[0]:= jump to error 2;
    gm   0                MA     ;    for i:= upper stack limit step -1 until 0 do
a21:grn  40e25    t   -1  M        ;        store with marks(stack[i], 0, 0);
    bs   (a21)    t   d1           ;    ds:= 0; state:= 27;
    hv   a21                       ;    go to NEXT;
    pp   d1       ,   hh  c66      ;
c62:pm   (e1)     X   1            ; END PASS 3: output(input); comment final value
    hs   e2               LA       ;    of short from pass 2;
    ps   e92-1    ,   hv  e3       ;    transfer from drum (endpass track)
                                   ;        to:(place for endpass);
                                   ;    go to PLACE FOR ENDPASS;
c7: arn  s        ,   ps  c66      ; ALARM: procedure alarm(n); value n; integer n;
    hv   c6               LQA      ;    begin if introuble then go to AFTER OPERAND;
    sr   6b       ,   ac  a10      ;    errormessage(n);
    hs   e5                        ;    byteword:= byteword - 1;
a10:qq   d7       ,   qq           ;    byte:= trouble;
    pa   c1       ,   grn a10      ;    introuble:= first after trouble:= true;
    pa   a10      t   d7           ;    operand:= 0; go to NOT OPERAND
    qq   (e1)     t   -1           ;    end;
    pi   12       t   -13          ;    comment errormessages: 38: stack, 51: -delimi
    arn  a10      ,   hv  c5       ;    ter, 52: operand, 53: delimiter, 54: -operand
                                   ;    55: number, 56: termination;
c8: sy   64               NKC      ; Special output, stack: CARRET
    arn  (a1)      D      NKC      ;
    hs   e9               NKC      ;                byte
    sy   27               NKC      ;                comma
    sy   (c1)             NKC      ;                operand
    pmn  (a1)     X                ; reset A and M
d i=i-d40-d40-d40-d40-d40-d40      ; remove special output
    ga   a11      V       LA       ; SEARCH: if marks = 1 then alarm(53);
    hs   c7       ,   qq  s+e53    ;    comment delimiter; base:= part 1(R);
    pm   (c1)     DX               ;    if marks = 3 then go to a12;
    hv   a12              LC       ;    if operand = 0 then alarm(54);
    pm   d8       DVX     NZ       ;    comment - operand;
a14:hs   c7       ,   qq  s+e54    ;    if stack[ds] ≠ elseex then go to a13;
    nc   (p)      ,   hv  a13      ;
    sy   (p)              NKC      ; Special output: top of stack
d i = i - d40                      ; remove special output unless d40 = 0
    pm   d9       DV               ;    R:= end else expr;
c55:cl   -9                        ; c55:
    hs   e3       X                ;    output(R);
c54:pp   p-1                       ; c54: ds:= ds - 1;
a13:sy   (p)              NKC      ; Special output: top of stack
d i = i - d40                      ; remove special output unless d40 = 0
    pmn  (a1)     X                ; a13: if -,bit(stack[ds],
                                   ;    allowed stackpart(table[byte])
    ck   (p)      ,   is  (p)      ;    then alarm(56); comment terminator;
a11:pmn  s[base]  XV      LQ       ;    R:= table[stack[ds] + base];
    hs   c7       ,   qq  s+e56    ;    marks:= marks of table[stack[ds] + base];
    hv   c10              NC       ;    if marks = 0 then go to NORMAL ACTION;
    hv   c55              LA       ;    if marks = 1 then begin R:= part 4(R);
    hv   c54                       ;    go to c55 end; go to c54;
```

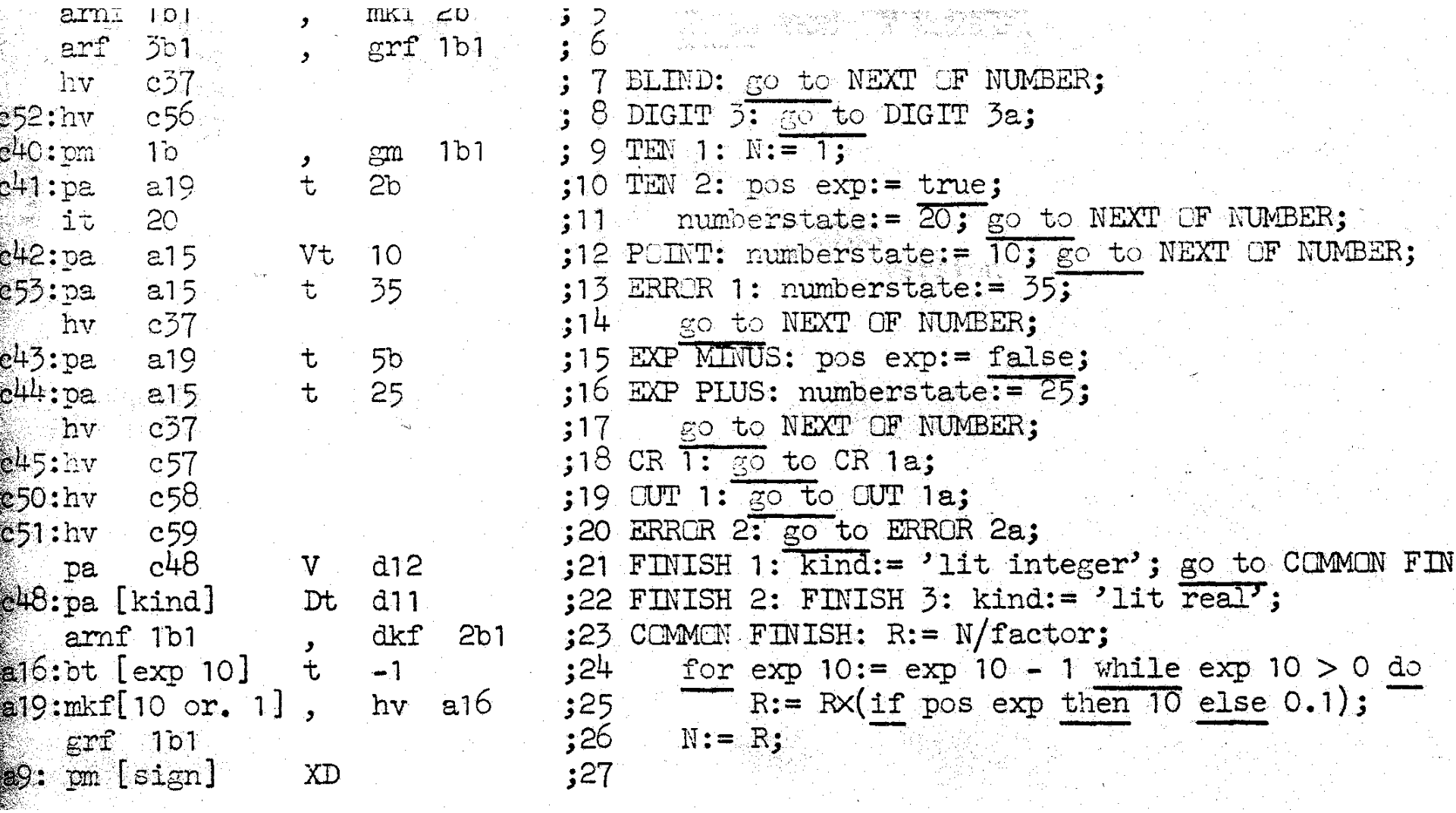


# The “GIER Algol compiler” scrolls, 1963 AD

```
    arnf  1b1       ,   mkf  2b      ; 5
    arf   3b1       ,   grf  1b1     ; 6
    hv    c37                        ; 7 BLIND: go to NEXT OF NUMBER;
c52:hv    c56                        ; 8 DIGIT 3: go to DIGIT 3a;
c40:pm    1b        ,   gm   1b1     ; 9 TEN 1: N:= 1;
c41:pa    a19       t   2b           ;10 TEN 2: pos exp:= true;
    it    20                         ;11    numberstate:= 20; go to NEXT OF NUMBER;
c42:pa    a15       Vt  10           ;12 POINT: numberstate:= 10; go to NEXT OF NUMBER;
c53:pa    a15       t   35           ;13 ERROR 1: numberstate:= 35;
    hv    c37                        ;14    go to NEXT OF NUMBER;
c43:pa    a19       t   5b           ;15 EXP MINUS: pos exp:= false;
c44:pa    a15       t   25           ;16 EXP PLUS: numberstate:= 25;
    hv    c37                        ;17    go to NEXT OF NUMBER;
c45:hv    c57                        ;18 CR 1: go to CR 1a;
c50:hv    c58                        ;19 OUT 1: go to OUT 1a;
c51:hv    c59                        ;20 ERROR 2: go to ERROR 2a;
    pa    c48       V   d12          ;21 FINISH 1: kind:= 'lit integer'; go to COMMON FIN
c48:pa [kind]       Dt  d11          ;22 FINISH 2: FINISH 3: kind:= 'lit real';
    arnf 1b1        ,   dkf  2b1     ;23 COMMON FINISH: R:= N/factor;
a16:bt [exp 10]     t   -1           ;24    for exp 10:= exp 10 - 1 while exp 10 > 0 do
a19:mkf[10 or. 1]   ,   hv   a16     ;25        R:= Rx(if pos exp then 10 else 0.1);
    grf  1b1                         ;26    N:= R;
a9: pm [sign]       XD               ;27
```

```
F12: RX  W3          B38     ;   W3 CROSS CLOCK;
     AL  W3        X3+1      ;   W3:= W3 + 1;
     SO  W2           1      ;   IF MODE = SINGLE
     JL.              H6,    ;     THEN GOTO STORE CLOCK;
     AL  W2           0      ;   W2:= 0; COMMENT TASK 0;
H1:  AL  W2        X2+2      ; STEP: W2:= W2 + 2;
     SN  W2        A60   x 2 ;    IF W2 = NO OF TASK
     JL.              H4,    ;      THEN GOTO FIN LOOP;
     RL. W1        X2+C8.    ;   W1:= NEXT EXECUTION[W2];
     SN. W1           (D9.)  ;    IF W1 = <INFINITE>
     JL,              H1,    ;      THEN GOTO STEP;
     RL. W0        X2+C4.    ;   W0:= DUMP ACT [W2];
     SN  W0           0      ;    IF W0 = <PASSIVE>
     SL  W1        X3+1      ;      v W1 > W3
     JL.              H3,    ;      THEN GOTO TEST DAY;
     RL. W0        X2+C7.    ;   DUMP ACT[W2]:=
     RS. W0        X2+C4.    ;      ADDR TAB[W2];
     AL  W0           1      ;
     SL. W0       (X2+C9.  ) ;    IF PERIODE[W2]<1 THEN
     RS. W0        X2+C9.    |      PERIODE[W2]:=1;
H2:  HA. W1        X2+C9.    J   FOR W1:= W1 + PERIODE [W2]
     SH  W1        X3        ;      WHILE W1 < W3 DO;
     JL.              H2,    ;
H3:  SL. W3           (D13.) ; TEST DAY: IF W3 > DAY IN SECONDS
     WS. W1           D13.   ;      THEN W1:=W1= DAY IN SECONDS;
```

# The “GIER Algol compiler” scrolls, 1963 AD



```
    arnf 1b1        ,   mkf 2b      ; 5
    arf  3b1        ,   grf 1b1     ; 6
    hv   c37                        ; 7 BLIND: go to NEXT OF NUMBER;
c52:hv   c56                        ; 8 DIGIT 3: go to DIGIT 3a;
c40:pm   1b         ,   gm  1b1     ; 9 TEN 1: N:= 1;
c41:pa   a19        t   2b          ;10 TEN 2: pos exp:= true;
    it   20                         ;11    numberstate:= 20; go to NEXT OF NUMBER;
c42:pa   a15        Vt  10          ;12 POINT: numberstate:= 10; go to NEXT OF NUMBER;
c53:pa   a15        t   35          ;13 ERROR 1: numberstate:= 35;
    hv   c37                        ;14    go to NEXT OF NUMBER;
c43:pa   a19        t   5b          ;15 EXP MINUS: pos exp:= false;
c44:pa   a15        t   25          ;16 EXP PLUS: numberstate:= 25;
    hv   c37                        ;17    go to NEXT OF NUMBER;
c45:hv   c57                        ;18 CR 1: go to CR 1a;
c50:hv   c58                        ;19 OUT 1: go to OUT 1a;
c51:hv   c59                        ;20 ERROR 2: go to ERROR 2a;
    pa   c48        V   d12         ;21 FINISH 1: kind:= 'lit integer'; go to COMMON FIN
c48:pa [kind]       Dt  d11         ;22 FINISH 2: FINISH 3: kind:= 'lit real';
    arnf 1b1        ,   dkf  2b1    ;23 COMMON FINISH: R:= N/factor;
a16:bt [exp 10]     t   -1          ;24    for exp 10:= exp 10 - 1 while exp 10 > 0 do
a19:mkf[10 or. 1]   ,   hv  a16     ;25        R:= Rx(if pos exp then 10 else 0.1);
    grf  1b1                        ;26    N:= R;
a9: pm [sign]       XD              ;27
```

nts and enclosed between **begin** and **end** con-
a block. Every declaration appears in a block in
y and is valid only for that block.
ogram is a block or compound statement which is
tained within another statement and which makes
of other statements not contained within it.
e sequel the syntax and semantics of the language
given.[4]

FORMALISM FOR SYNTACTIC DESCRIPTION

syntax will be described with the aid of metalin-
formulae.[5] Their interpretation is best explained
example

⟨ab⟩ ::= ( | [ | ⟨ab⟩ ( | ⟨ab⟩⟨d⟩

ces of characters enclosed in the brackets ⟨ ⟩ repre-
etalinguistic variables whose values are sequences
bols. The marks ::= and | (the latter with the
g of **or**) are metalinguistic connectives. Any mark
rmula, which is not a variable or a connective,
itself (or the class of marks which are similar to it).
osition of marks and/or variables in a formula
juxtaposition of the sequences denoted. Thus the
above gives a recursive rule for the formation of
of the variable ⟨ab⟩. It indicates that ⟨ab⟩ may
e value ( or [ or that given some legitimate value
another may be formed by following it with the
er ( or by following it with some value of the vari-
). If the values of ⟨d⟩ are the decimal digits, some
of ⟨ab⟩ are:

[(((1(37(
(12345(
(((
[86

er to facilitate the study, the symbols used for
ishing the metalinguistic variables (i.e. the se-
of characters appearing within the brackets ⟨ ⟩
the above example) have been chosen to be words
ing approximately the nature of the corresponding
e. Where words which have appeared in this manner
d elsewhere in the text they will refer to the corre-
g syntactic definition. In addition some formulae
en given in more than one place.
ition:

⟨empty⟩ ::=
(i.e. the null string of symbols).

never the precision of arithmetic is stated as being in
not specified, or the outcome of a certain process is left
d or said to be undefined, this is to be interpreted in the
at a program only fully defines a computational process
companying information specifies the precision assumed,
of arithmetic assumed, and the course of action to be
all such cases as may occur during the execution of the
tion.

J. W. Backus, The syntax and semantics of the proposed
ional algebraic language of the Zürich ACM–GAMM
ce. Proc. Internat. Conf. Inf. Proc., UNESCO, Paris,
9.

The reference language is built up from the following basic symbols:

⟨basic symbol⟩ ::= ⟨letter⟩|⟨digit⟩|⟨logical value⟩|⟨delimiter⟩

### 2.1. LETTERS

⟨letter⟩ ::= *a*|*b*|*c*|*d*|*e*|*f*|*g*|*h*|*i*|*j*|*k*|*l*|*m*|*n*|*o*|*p*|*q*|*r*|*s*|*t*|*u*|*v*|*w*|*x*|*y*|*z*|
*A*|*B*|*C*|*D*|*E*|*F*|*G*|*H*|*I*|*J*|*K*|*L*|*M*|*N*|*O*|*P*|*Q*|*R*|*S*|*T*|*U*|*V*|*W*|*X*|*Y*|*Z*

This alphabet may arbitrarily be restricted, or extended with any other distinctive character (i.e. character **not** coinciding with any digit, logical value or delimiter).

Letters do not have individual meaning. They are used for forming identifiers and strings[6] (cf. sections 2.4. Identifiers, 2.6. Strings).

#### 2.2.1. DIGITS

⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9

Digits are used for forming numbers, identifiers, and strings.

#### 2.2.2. LOGICAL VALUES

⟨logical value⟩ ::= **true**|**false**

The logical values have a fixed obvious meaning.

### 2.3. DELIMITERS

⟨delimiter⟩ ::= ⟨operator⟩|⟨separator⟩|⟨bracket⟩|⟨declarator⟩|⟨specificator⟩
⟨operator⟩ ::= ⟨arithmetic operator⟩|⟨relational operator⟩|⟨logical operator⟩|⟨sequential operator⟩
⟨arithmetic operator⟩ ::= +|−|×|/|÷|↑
⟨relational operator⟩ ::= <|≦|=|≧|>|≠
⟨logical operator⟩ ::= ≡|⊃|∨|∧|¬
⟨sequential operator⟩ ::= **go to**|**if**|**then**|**else**|**for**|**do**[7]
⟨separator⟩ ::= ,|.|₁₀|:|;|:=|**u**|**step**|**until**|**while**|**comment**
⟨bracket⟩ ::= (|)|[|]|**begin**|**end**
⟨declarator⟩ ::= **Boolean**|**integer**|**real**|**array**|**switch**|**procedure**
⟨specificator⟩ ::= **label**|**value**

Delimiters have a fixed meaning which for the most part is obvious or else will be given at the appropriate place in the sequel.

Typographical features such as blank space or change to a new line have no significance in the reference language. They may, however, be used freely for facilitating reading.

For the purpose of including text among the symbols of

[6] It should be particularly noted that throughout the reference language underlining [in typewritten copy; boldface type in printed copy—Ed.] is used for defining independent basic symbols (see sections 2.2.2 and 2.3). These are understood to have no relation to the individual letters of which they are composed. Within the present report [not including headings—Ed.], boldface will be used for no other purpose.

[7] **do** is used in for statements. It has no relation whatsoever to the *do* of the preliminary report, which is not included in ALGOL 60.

⟨delimiter⟩ ::= ⟨operator⟩|⟨separator⟩|⟨bracket⟩|⟨declarator⟩|
⟨specificator⟩

⟨operator⟩ ::= ⟨arithmetic operator⟩|⟨relational operator⟩|
⟨logical operator⟩|⟨sequential operator⟩

⟨arithmetic operator⟩ ::= +|−|×|/|÷|↑

⟨relational operator⟩ ::= <|≦|=|≧|>|≠

⟨logical operator⟩ ::= ≡|⊃|∨|∧|¬

⟨sequential operator⟩ ::= **go to**|**if**|**then**|**else**|**for**|**do**[7]

⟨separator⟩ ::= ,|.|$_{10}$|:|;|:=|␣|**step**|**until**|**while**|**comment**

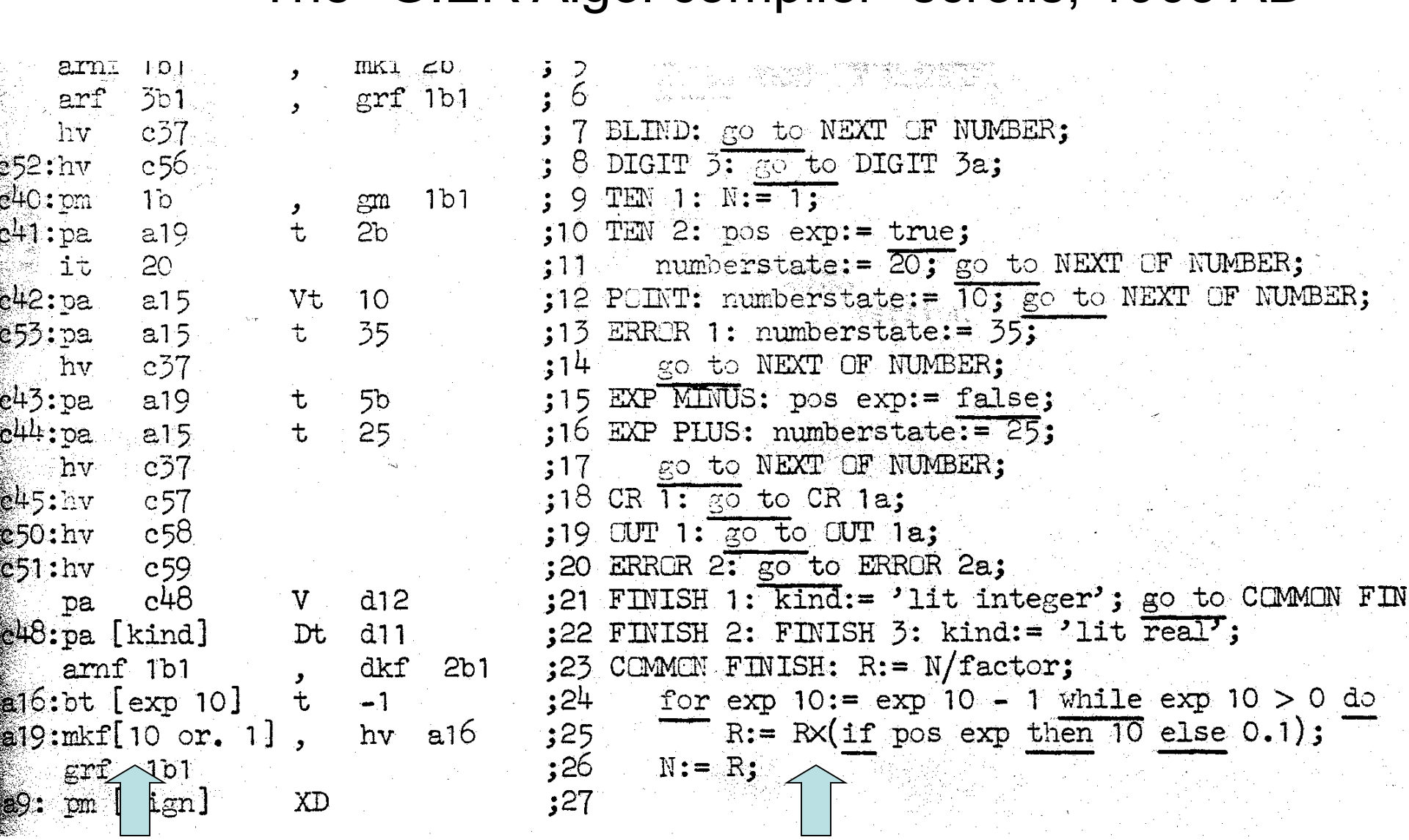
⟨bracket⟩ ::= (|)|[|]|'|'|**begin**|**end**

⟨declarator⟩ ::= **own**|**Boolean**|**integer**|**real**|**array**|**switch**|
**procedure**

⟨specificator⟩ ::= **string**|**label**|**value**

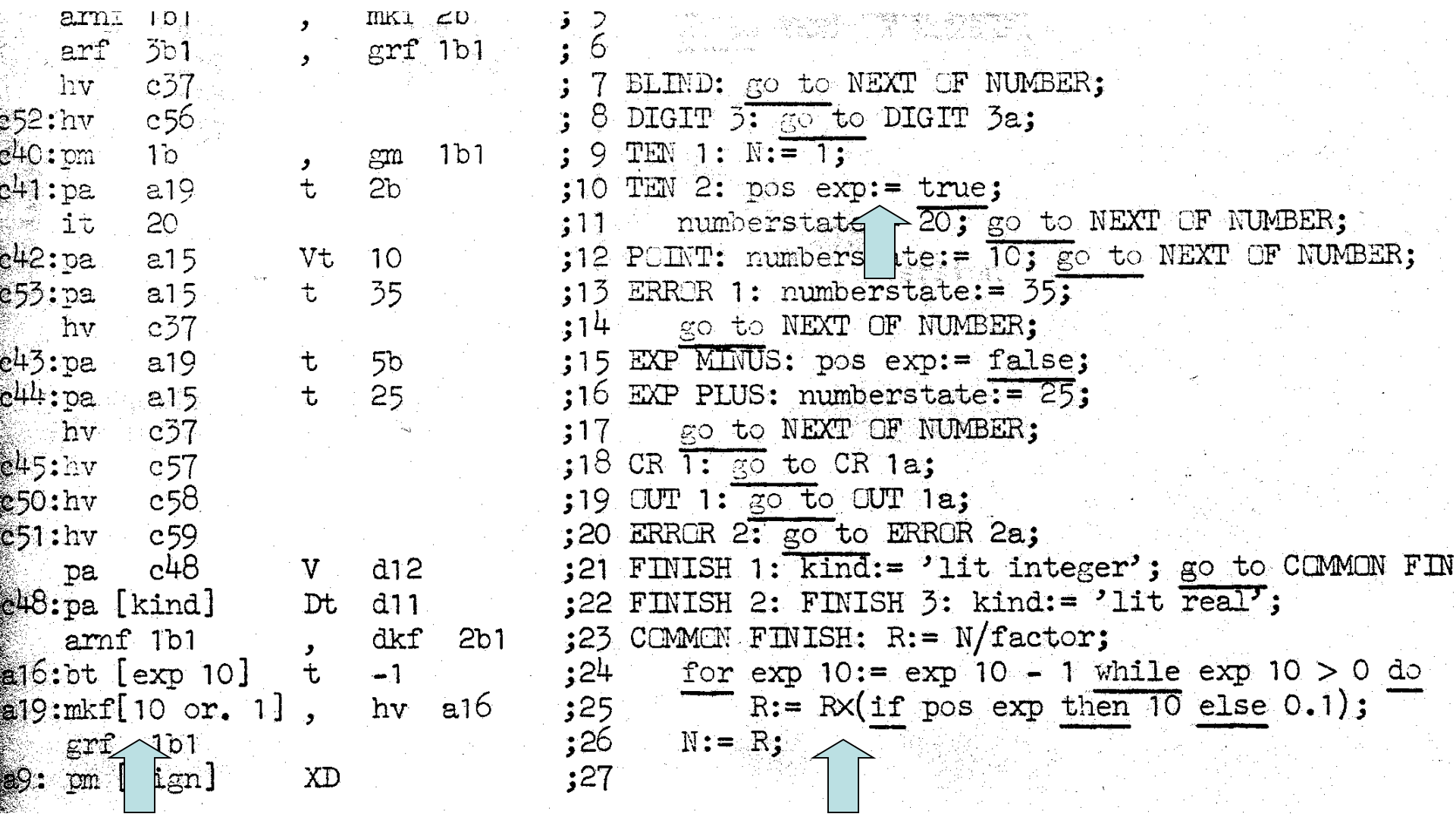
Algol: $2.7_{10}$-31 Fortran: 2.7E-31

# The “GIER Algol compiler” scrolls, 1963 AD



```
    arnf  1b1       ,   mkf  2b      ; 5
    arf   3b1       ,   grf  1b1     ; 6
    hv    c37                        ; 7 BLIND: go to NEXT OF NUMBER;
c52:hv    c56                        ; 8 DIGIT 3: go to DIGIT 3a;
c40:pm    1b        ,   gm   1b1     ; 9 TEN 1: N:= 1;
c41:pa    a19       t   2b           ;10 TEN 2: pos exp:= true;
    it    20                         ;11    numberstate:= 20; go to NEXT OF NUMBER;
c42:pa    a15       Vt  10           ;12 POINT: numberstate:= 10; go to NEXT OF NUMBER;
c53:pa    a15       t   35           ;13 ERROR 1: numberstate:= 35;
    hv    c37                        ;14    go to NEXT OF NUMBER;
c43:pa    a19       t   5b           ;15 EXP MINUS: pos exp:= false;
c44:pa    a15       t   25           ;16 EXP PLUS: numberstate:= 25;
    hv    c37                        ;17    go to NEXT OF NUMBER;
c45:hv    c57                        ;18 CR 1: go to CR 1a;
c50:hv    c58                        ;19 OUT 1: go to OUT 1a;
c51:hv    c59                        ;20 ERROR 2: go to ERROR 2a;
    pa    c48       V   d12          ;21 FINISH 1: kind:= 'lit integer'; go to COMMON FIN
c48:pa [kind]       Dt  d11          ;22 FINISH 2: FINISH 3: kind:= 'lit real';
    arnf  1b1       ,   dkf  2b1     ;23 COMMON FINISH: R:= N/factor;
a16:bt [exp 10]     t   -1           ;24    for exp 10:= exp 10 - 1 while exp 10 > 0 do
a19:mkf[10 or. 1]   ,   hv   a16     ;25        R:= Rx(if pos exp then 10 else 0.1);
    grf   1b1                        ;26    N:= R;
a9: pm [  ign]      XD               ;27
```

# The “GIER Algol compiler” scrolls, 1963 AD



```
    arnf  1b1        ,   mkf  2b      ; 5
    arf   3b1        ,   grf  1b1     ; 6
    hv    c37                         ; 7 BLIND: go to NEXT OF NUMBER;
c52:hv    c56                         ; 8 DIGIT 3: go to DIGIT 3a;
c40:pm    1b         ,   gm   1b1     ; 9 TEN 1: N:= 1;
c41:pa    a19        t   2b           ;10 TEN 2: pos exp:= true;
    it    20                          ;11    numberstate:= 20; go to NEXT OF NUMBER;
c42:pa    a15        Vt  10           ;12 POINT: numberstate:= 10; go to NEXT OF NUMBER;
c53:pa    a15        t   35           ;13 ERROR 1: numberstate:= 35;
    hv    c37                         ;14    go to NEXT OF NUMBER;
c43:pa    a19        t   5b           ;15 EXP MINUS: pos exp:= false;
c44:pa    a15        t   25           ;16 EXP PLUS: numberstate:= 25;
    hv    c37                         ;17    go to NEXT OF NUMBER;
c45:hv    c57                         ;18 CR 1: go to CR 1a;
c50:hv    c58                         ;19 OUT 1: go to OUT 1a;
c51:hv    c59                         ;20 ERROR 2: go to ERROR 2a;
    pa    c48        V   d12          ;21 FINISH 1: kind:= 'lit integer'; go to COMMON FIN
c48:pa [kind]        Dt  d11          ;22 FINISH 2: FINISH 3: kind:= 'lit real';
    arnf  1b1        ,   dkf  2b1     ;23 COMMON FINISH: R:= N/factor;
a16:bt [exp 10]      t   -1           ;24    for exp 10:= exp 10 - 1 while exp 10 > 0 do
a19:mkf[10 or. 1]    ,   hv   a16     ;25       R:= Rx(if pos exp then 10 else 0.1);
    grf   1b1                        ;26    N:= R;
a9: pm [  ign]       XD               ;27
```

# Good vs. Bad Comments

- *Good*:

  pa a19 t 2b ; pos exp:= true;

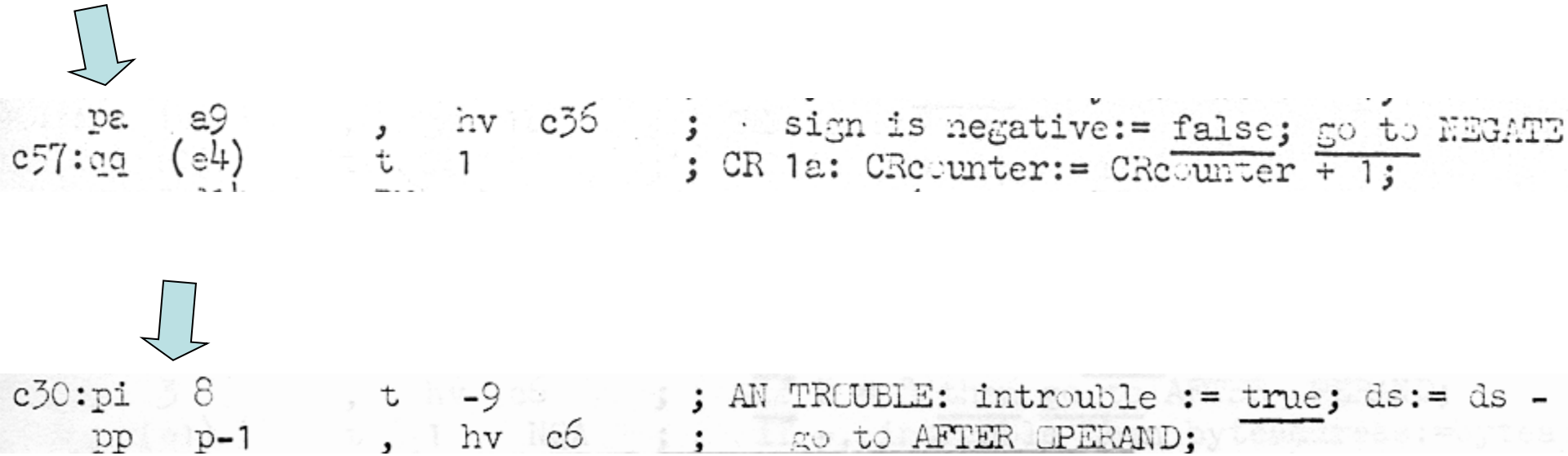
- *Not so good*:

  pa a19 t 2b ; /* kludge!!! set multiply address part to address of 10 */

- *Bad*:

  pa a19 t 2b ; // set address part of a19 to 2b

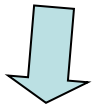
# 2 more implementations of Boolean:



```
    pa   a9        ,   hv  c36    ;  sign is negative:= false; go to NEGATE
c57:qq   (e4)      t   1          ; CR 1a: CRcounter:= CRcounter + 1;

c30:pi   8         t   -9         ; AN TROUBLE: introuble := true; ds:= ds -
    pp   p-1       ,   hv  c6     ;    go to AFTER OPERAND;
```

# Improved with comment?



```
c30:pi   8        t    -9 [= ¬8]   ; AN TROUBLE: introuble := true; ds:= ds -
     pp   p-1     ,    hv  c6      ;    go to AFTER OPERAND;
```

```
bt [exp 10]    t    -1
mkf[10 or. 1] ,    hv   a16
```

```
for exp 10:= exp 10 - 1 while exp 10 > 0 do
    R:= Rx(if pos exp then 10 else 0.1);
```

10.5 BYTES

40 YEARS LATER….

```
      while (exp10 > 0)
00411CA7  cmp         dword ptr [exp10],0
00411CAB  jle         foo+82h (411CE2h)
            exp10--;
00411CAD  mov         eax,dword ptr [exp10]
00411CB0  sub         eax,1
00411CB3  mov         dword ptr [exp10],eax
            R=R * (posExp ? 10.0 : 0.1);
00411CB6  movzx       eax,byte ptr [posExp]
00411CBA  test        eax,eax
00411CBC  je          foo+6Ah (411CCAh)
00411CBE  mov         dword ptr [ebp-100h],41200000h
00411CC8  jmp         foo+74h (411CD4h)
00411CCA  mov         dword ptr [ebp-100h],3DCCCCCDh
00411CD4  fld         dword ptr [R]
00411CD7  fmul        dword ptr [ebp-100h]
00411CDD  fstp        dword ptr [R]
```

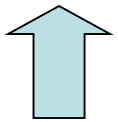
58 BYTES

*5.5 times larger*

```
    pm   a20      V                ;        stackidentifier:= false;
a20:hv   c51      ,   hv  c51      ;    store[0]:= jump to error 2;
    gm   0                MA     ;    for i:= upper stack limit step -1 until 0 do
a21:grn  40e25    t   -1  M        ;        store with marks(stack[i], 0, 0);
    bs   (a21)    t   d1           ;    ds:= 0; state:= 27;
    hv   a21                       ;    go to NEXT;
    pp   d1       ,   hh  c66      ;
c62:pm   (e1)     X   1            ; END PASS 3: output(input); comment final value
    hs   e2               LA       ;    of short from pass 2;
    ps   e92-1    ,   hv  e3       ;    transfer from drum (endpass track)
                                   ;        to:(place for endpass);
                                   ;    go to PLACE FOR ENDPASS;
c7: arn  s        ,   ps  c66      ; ALARM: procedure alarm(n); value n; integer n;
    hv   c6               LQA      ;    begin if introuble then go to AFTER OPERAND;
    sr   6b       ,   ac  a10      ;    errormessage(n);
    hs   e5                        ;    byteword:= byteword - 1;
a10:qq   d7       ,   qq           ;    byte:= trouble;
    pa   c1       ,   grn a10      ;    introuble:= first after trouble:= true;
    pa   a10      t   d7           ;    operand:= 0; go to NOT OPERAND
    qq   (e1)     t   -1           ;    end;
    pi   12       t   -13          ;    comment errormessages: 38: stack, 51: -delimi
    arn  a10      ,   hv  c5       ;    ter, 52: operand, 53: delimiter, 54: -operand
                                   ;    55: number, 56: termination;
c8: sy   64               NKC      ; Special output, stack: CARRET
    arn  (a1)      D      NKC      ;
    hs   e9               NKC      ;                byte
    sy   27               NKC      ;                comma
    sy   (c1)             NKC      ;                operand
    pmn  (a1)     X                ; reset A and M
d i=i-d40-d40-d40-d40-d40-d40      ; remove special output
    ga   a11      V       LA       ; SEARCH: if marks = 1 then alarm(53);
    hs   c7       ,   qq  s+e53    ;    comment delimiter; base:= part 1(R);
    pm   (c1)     DX               ;    if marks = 3 then go to a12;
    hv   a12              LC       ;    if operand = 0 then alarm(54);
    pm   d8       DVX     NZ       ;    comment - operand;
a14:hs   c7       ,   qq  s+e54    ;    if stack[ds] ≠ elseex then go to a13;
    nc   (p)      ,   hv  a13      ;
    sy   (p)              NKC      ; Special output: top of stack
d i = i - d40                      ; remove special output unless d40 = 0
    pm   d9       DV               ;    R:= end else expr;
c55:cl   -9                        ; c55:
    hs   e3       X                ;    output(R);
c54:pp   p-1                       ; c54: ds:= ds - 1;
a13:sy   (p)              NKC      ; Special output: top of stack
d i = i - d40                      ; remove special output unless d40 = 0
    pmn  (a1)     X                ; a13: if -,bit(stack[ds],
                                   ;    allowed stackpart(table[byte])
    ck   (p)      ,   is  (p)      ;    then alarm(56); comment terminator;
a11:pmn  s[base]  XV      LQ       ;    R:= table[stack[ds] + base];
    hs   c7       ,   qq  s+e56    ;    marks:= marks of table[stack[ds] + base];
    hv   c10              NC       ;    if marks = 0 then go to NORMAL ACTION;
    hv   c55              LA       ;    if marks = 1 then begin R:= part 4(R);
    hv   c54                       ;    go to c55 end; go to c54;
```

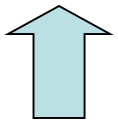
# Domain intentions & notations

```
c57:qq  (e4)        t   1          ; CR 1a: CRcounter:= CRcounter + 1;
    pmn  d14        DX             ;     output('CARRET');
```

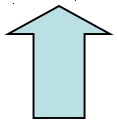


# Domain intentions & notations

```
c57:qq  (e4)      t   1          ; CR 1a: CRcounter:= CRcounter + 1;
    pmn  d14      DX             ;     output('CARRET');
```



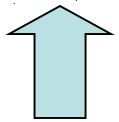
```
    pa   c48      V   d12        ;21 FINISH 1: kind:= 'lit integer';
```

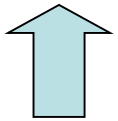


# Domain intentions & notations

```
pa    c48        V    d12          ;21 FINISH 1: kind:= 'lit integer';

pm    d9         DV                ;    R:= end else expr;
```

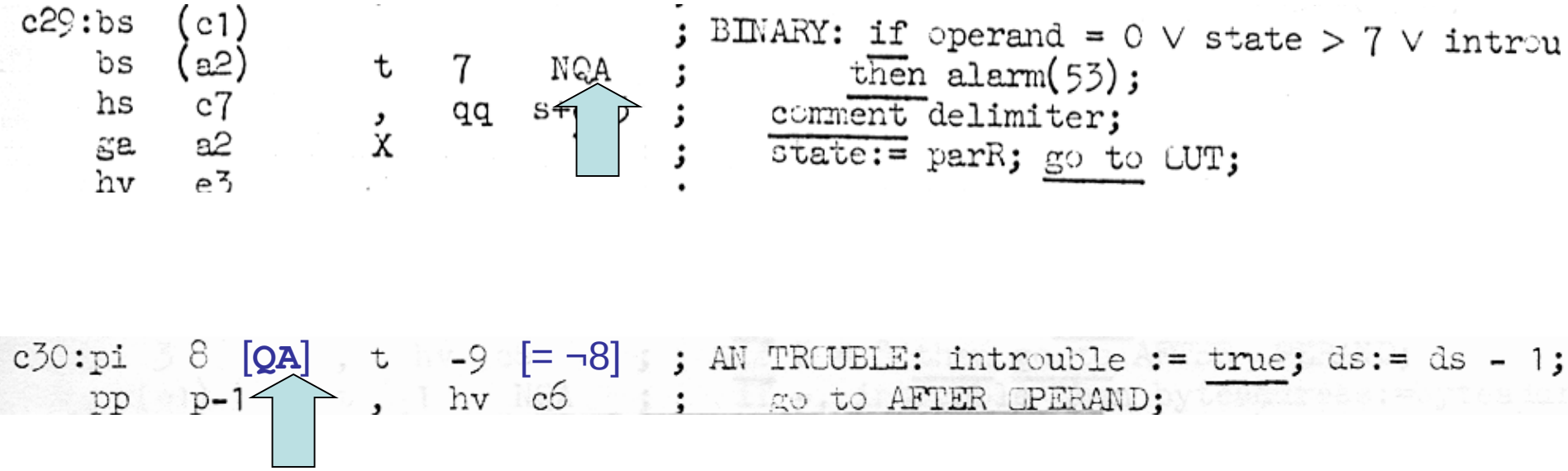




# Domain intentions & notations

```
c29:bs  (c1)                                ; BINARY: if operand = 0 ∨ state > 7 ∨ introu
    bs  (a2)      t   7    NQA              ;         then alarm(53);
    hs   c7       ,   qq   s+e53            ;    comment delimiter;
    ga   a2       X                         ;    state:= parR; go to CUT;
    hv   e3                                 .
```

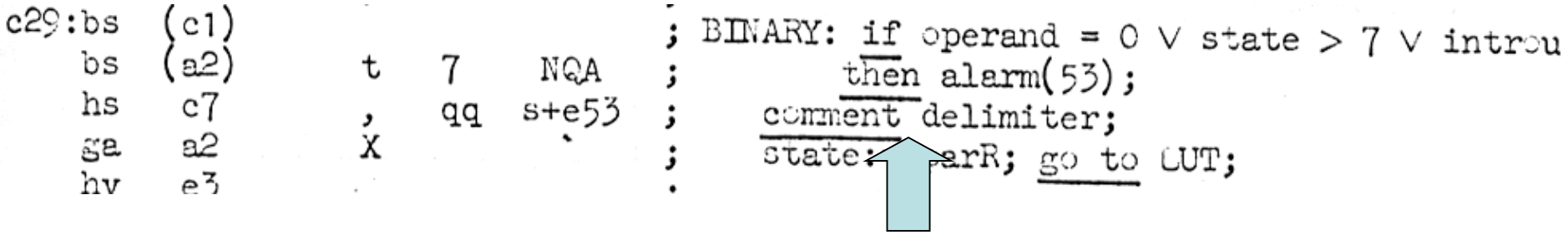
# Domain intentions & notations



# Domain intentions & notations

```
c29:bs  (c1)                              ; BINARY: if operand = 0 ∨ state > 7 ∨ introu
    bs  (a2)      t   7    NQA            ;         then alarm(53);
    hs   c7       ,   qq   s+e53          ;     comment delimiter;
    ga   a2       X                       ;     state:=parR; go to CUT;
    hv   e3                               ;
```

# Domain intentions & notations




Algol 60: «delimiter»

Gier Algol: ⫞delimiter⫟

Today: "delimiter"

In expres + - × / ÷
Exp ex , arth if step
Exp ex else
Exp left part or exp :=
Exp unc ex. then
After subscr bou
Bool ex ¬ < ≤ = ≥ > ≠
Exp unc st then :
Exp st else
After proc end
Exp st do :
In value part value
After str segm ]
In spec array switch p
In spec integer real
In heading proce
In type list ,
After formals )
In decl integer
After end or )
In formals (
In for st for
In decl switch
In decl array
In decl own
After ]for
Exp statement
Exp st or decl
After ]assign
Exp value or spec
Exp body or spec
Trouble in

⟨delimiter⟩ ::= ⟨operator⟩|⟨separator⟩|⟨bracket⟩|⟨declarator⟩|
⟨specificator⟩

⟨operator⟩ ::= ⟨arithmetic operator⟩|⟨relational operator⟩|
⟨logical operator⟩|⟨sequential operator⟩

⟨arithmetic operator⟩ ::= +|−|×|/|÷|↑

⟨relational operator⟩ ::= <|≦|=|≧|>|≠

⟨logical operator⟩ ::= ≡|⊃|∨|∧|¬

⟨sequential operator⟩ ::= **go to**|**if**|**then**|**else**|**for**|**do**[7]

⟨separator⟩ ::= ,|.|$_{10}$|:|;|:=|␣|**step**|**until**|**while**|**comment**

⟨bracket⟩ ::= (|)|[|]|‘|’|**begin**|**end**

⟨declarator⟩ ::= **own**|**Boolean**|**integer**|**real**|**array**|**switch**|
**procedure**

⟨specificator⟩ ::= **string**|**label**|**value**

# Rosetta Code Lesson…

- Comments contain intentions that are not in the code.
- There is always a next level of comment, a next level of intention...

# State of Programming 1963 - 2003

```
comment complex 2nd order equation-8th October 1963;
begin comment: SECRETARY - October 1963;
integer pagecount, linecount, job no, day, month, year, drum;

procedure outpage; outline(100);

procedure outline(a);
value a; integer a;
begin
if linecount-6 < a then a := linecount+2;
linecount := linecount-a;
for a := a-1 step -1 until 0 do outcr;

if linecount < 0 then begin
pagecount := pagecount+1;
linecount := linecount+64;
if pagecount > 1 then
begin outsp(32); output(¢-ddd¢,-pagecount,outtext(¢<-¢)) end;
outtext(¢<
¢);
end of linecount<0;
end of outline procedure;

procedure tape feed(n);
value n; integer n;
for n := n step -1 until 0 do outchar(63);

drum := drumplace;
linecount := 0;
tape feed(30); outclear;

start:
drumplace := drum;
pagecount := 0;
job no := inone;
if job no < 0 then goto finis;
input(day,month,year);
tape feed(30); outpage;
```

```
public CodeTable()
{
   rgcod = new ArrayList();
}
public ArrayList rgcod;

public void Pass4(XCOD xcod, int i, NTE nte)
{
   Console.WriteLine("P4: " + xcod.ToString());
   this.rgcod.Add(new MICOP(xcod, i, nte));
}

public MICOP MicopLast()
{
   return (MICOP)this.rgcod[this.rgcod.Count - 1];
}
public void DeleteLastMicop()
{
   this.rgcod.RemoveAt(this.rgcod.Count - 1);
}

public void Px()
{
   Console.WriteLine("Produced code");
   int i = 0;
   foreach (MICOP micop in this.rgcod)
   {
      Console.WriteLine("{0,4}\t{1,-14}\t{2}\t{3}",
         i++,
         micop.xcod.ToString(),
         micop.i,
         micop.nte == null ? " " : micop.nte.ToString());
   }

}
```

# State of Programming 1963 - 2003

```
comment complex 2nd order equation-8th October 1963;
begin comment: SECRETARY - October 1963;
integer pagecount, linecount, job no, day, month, year, drum;

procedure outpage; outline(100);

procedure outline(a);
value a; integer a;
begin
if linecount-6 < a then a := linecount+2;
linecount := linecount-a;
for a := a-1 step -1 until 0 do outcr;

if linecount < 0 then begin
pagecount := pagecount+1;
linecount := linecount+64;
if pagecount > 1 then
begin outsp(32); output(¢-ddd¢,-pagecount,outtext(¢<-¢)) end;
outtext(¢<
¢);
end of linecount<0;
end of outline procedure;

procedure tape feed(n);
value n; integer n;
for n := n step -1 until 0 do outchar(63);

drum := drumplace;
linecount := 0;
tape feed(30); outclear;

start:
drumplace := drum;
pagecount := 0;
job no := inone;
if job no < 0 then goto finis;
input(day,month,year);
tape feed(30); outpage;
```

```
public CodeTable()
{
   rgcod = new ArrayList();
}
public ArrayList rgcod;

public void Pass4(XCOD xcod, int i, NTE nte)
{
   Console.WriteLine("P4: " + xcod.ToString());
   this.rgcod.Add(new MICOP(xcod, i, nte));
}

public MICOP MicopLast()
{
   return (MICOP)this.rgcod[this.rgcod.Count - 1];
}
public void DeleteLastMicop()
{
   this.rgcod.RemoveAt(this.rgcod.Count - 1);
}

public void Px()
{
   Console.WriteLine("Produced code");
   int i = 0;
   foreach (MICOP micop in this.rgcod)
   {
      Console.WriteLine("{0,4}\t{1,-14}\t{2}\t{3}",
         i++,
         micop.xcod.ToString(),
         micop.i,
         micop.nte == null ? " " : micop.nte.ToString());
   }

}
```

by John von Neumann

# The Computer and the Brain

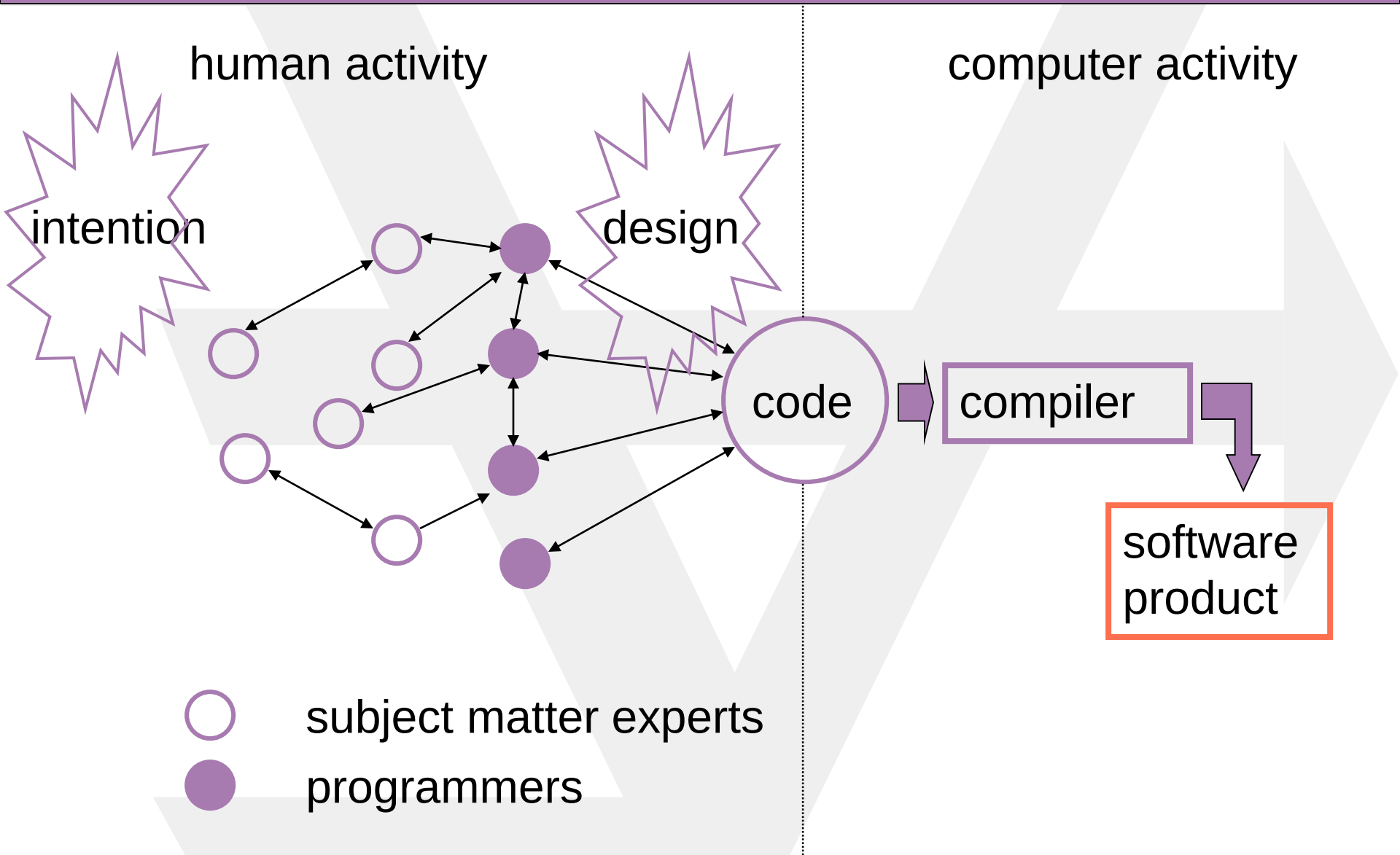
New Haven: Yale University Press

totality of the (electrical) connections referred to constitutes the set-up of the problem—the expression of the problem to be solved, i.e. of the intention of the user. So this is again a plugged co l. As in the case referred to, the plugged pattern an

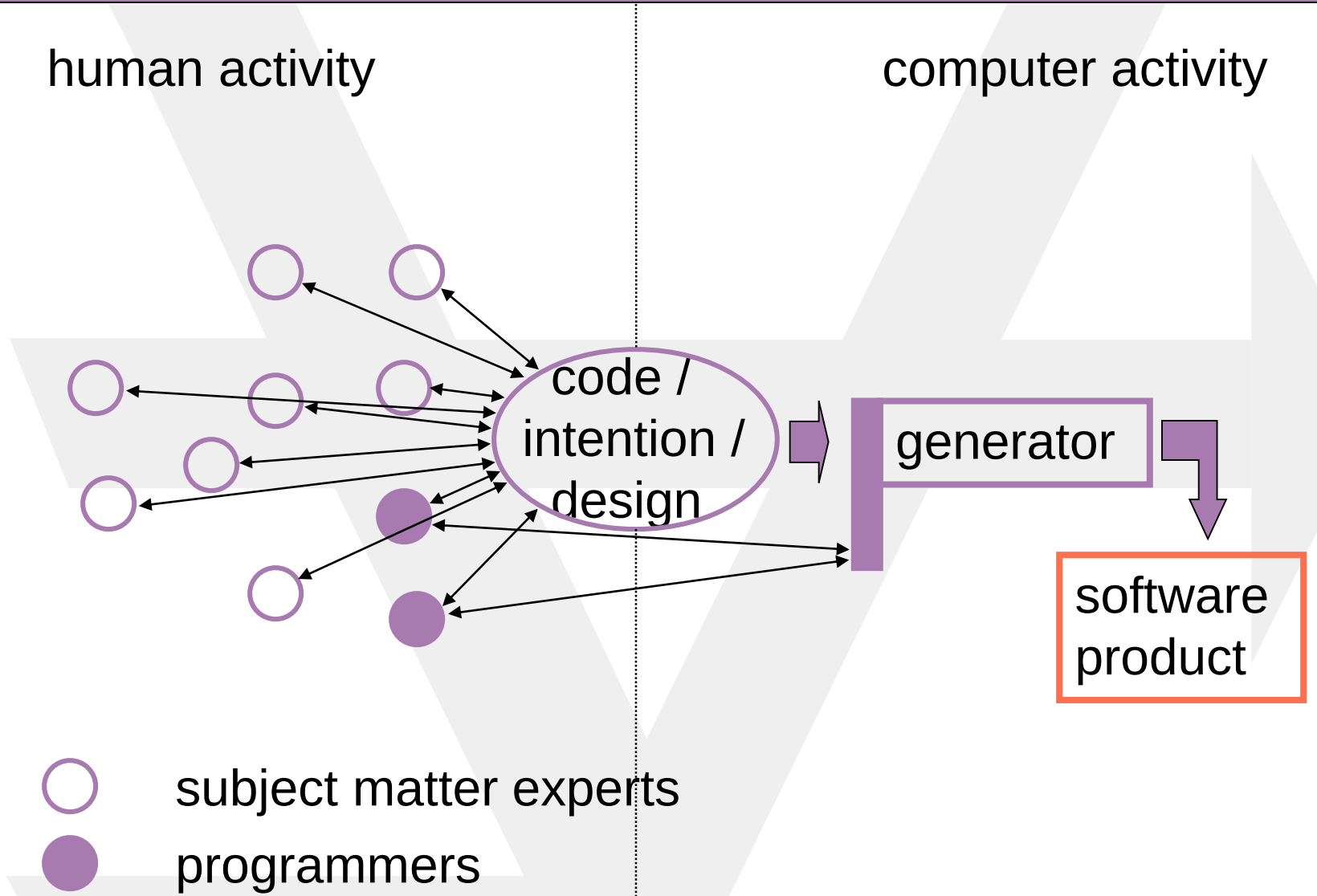
Of course, the order-system—this means the problem to be solved, the intention of the user—is communicated to the machin y "loading" it into the memory. This is usually d e from a previously

For its proper functioning—to solve the problems for which it is intended—a machine may need a capacity of a certa number, say $N$ words, at a

# Typical Software Development Process

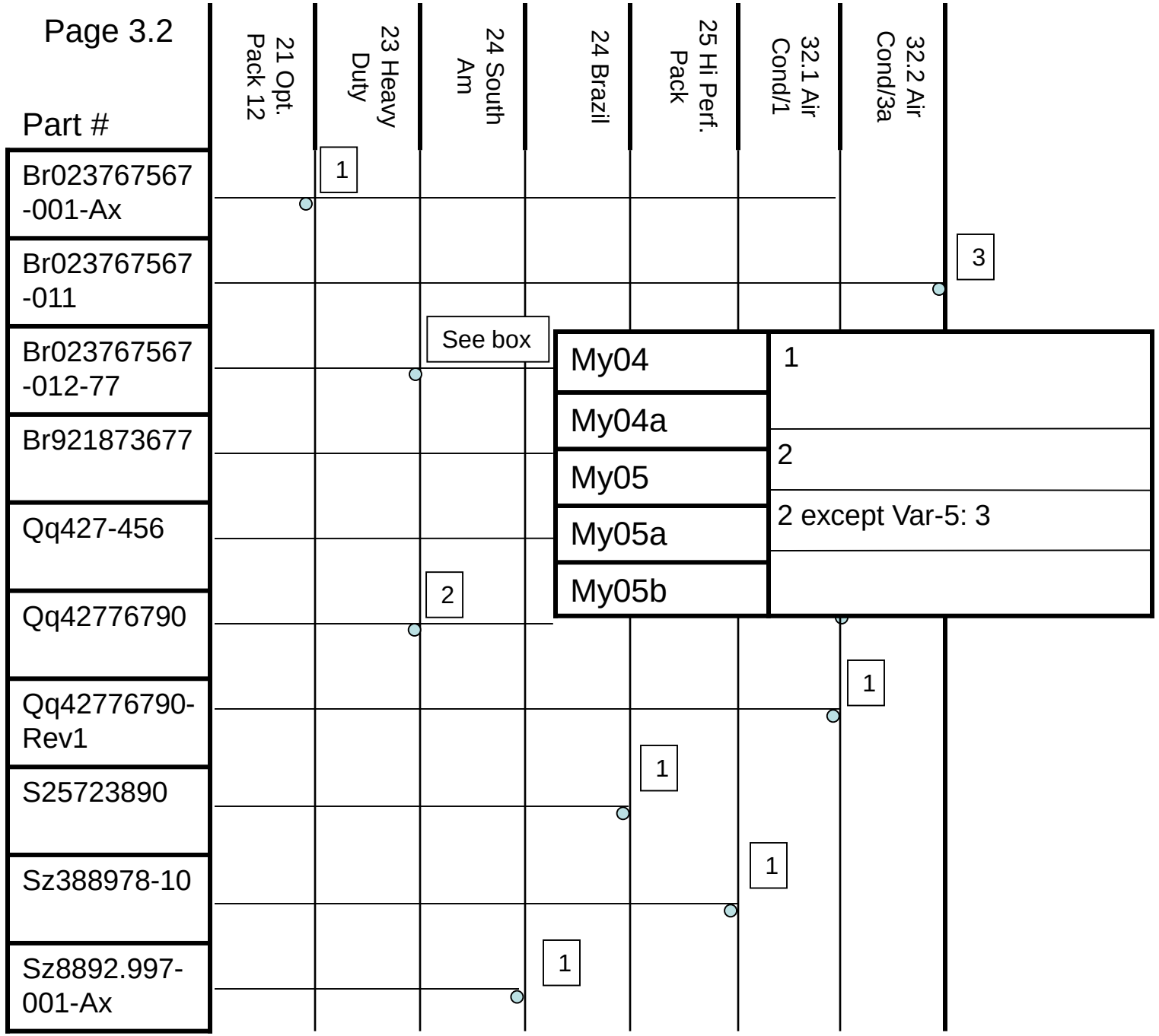


# Making the code look like the design…



# When code looks more like design…

- Input is edited with a sophisticated multi-view editing program by the SMEs – a "Super Powerpoint"
- Input is not in *any* Programming Language
  - Input is expressed in the Subject Matter Experts' usual terms of art, diagrams, formulas, tables, rules.


Part #
Br023767567-001-Ax
Br023767567-011
Br023767567-012-77
Br921873677
Qq427-456
Qq42776790
Qq42776790-Rev1
S25723890
Sz388978-10
Sz8892.997-001-Ax
21 Opt. Pack 12
23 Heavy Duty
24 South Am
24 Brazil
25 Hi Perf. Pack
32.1 Air Cond/1
32.2 Air Cond/3a
1
3
See box
2
1
1
1
1
My04
My04a
My05
My05a
My05b
1
2
2 except Var-5: 3

In expres + - × / ÷
Exp ex , arth if step
Exp ex else
Exp left part or exp :=
Exp unc ex. then
After subscr bou
Bool ex ¬ < ≤ = ≥ > ≠
Exp unc st then :
Exp st else
After proc end
Exp st do :
In value part value
After str segm ]
In spec array switch p
In spec integer real
In heading proce
In type list ,
After formals )
In decl integer
After end or )
In formals (
In for st for
In decl switch
In decl array
In decl own
After ]for
Exp statement
Exp st or decl
After ]assign
Exp value or spec
Exp body or spec
Trouble in

# When code looks more like design…

- Input becomes executable after being mechanically combined with the programmer's contributions.
- Input should be precise and consistent, but completeness is not a requirement; it is a process.

# When code looks more like design…

- SMEs have better control, because they can edit the code. Programmers are freed from much domain work and repetitive work.
- Speed, precision and costs move progressively from manual scale to machine scale.

- This was the last column on the Rosetta Code. To read more, enter "Biggest Damn Opportunity" into Google and press "I Feel Lucky"

INTENTIONAL

SOFTWARE