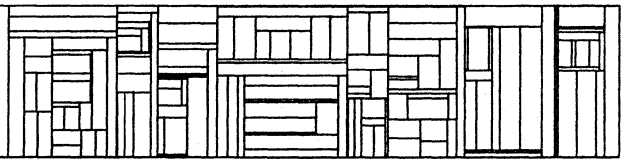
THE MATHILDA DRIVER,

A SOFTWARE TOOL FOR HARDWARE TESTING.

by
Eric Kressel
Ib Holm Sørensen

DAIMI MD-18
October 1975.

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06-12 83 55



PREFACE

Sometimes it is easy to get the impression that testing the hardware of a computer is the task of an engineer in electronics and no one else. This is not the case, however.

To encircle an error in a certain part of the machine, the engineer requires various bit patterns to loop through the erroneous part without effecting too many surrounding sections of the hardware. This usually implies writing complicated programs in a restricted set of the machine code (in order to avoid using critical sections surrounding the part in error). This is a job for software people, not technicians.

In the spring of 1975, the authors were to assist in the hardware testing of Mathilda, a computer designed and built in the department [1].

At this time, RIKKE-1 (a prototype machine of Mathilda, but with a 16-bit word length) was in operation with an Operating System written in BCPL [2] running on it. The I/O interface between Mathilda and RIKKE-1 is extremely simple (a question of connecting them with 2 cables) and our first task was to adjust the RIKKE-1 operating system so it viewed Mathilda as a device. We were now able to exploit the power of a higher level language for generating data and control information to Mathilda. This possibility lead to the development of a software tool on RIKKE-1 to enable that the data generation to and the flow of control of hardware test programs in Mathilda could be entirely driven in an interactive fashion from RIKKE-1's console, given that the test programs were structured suitably.

This approach to the testing of the hardware gave us many advantages, for example:

- the micro code in Mathilda was short, simple to write and used very few parts of the machine other than those of interest.
- the interactive way of working gave us immediate response, and closing in sources of errors was more or less trivial.

This manual describes the facilities of this software tool which we have named the Mathilda Driver, and gives a list of the commands applicable to it. We cannot describe how to hardware test Mathilda, because each test program has its own properties and they are defined by the resource to be tested and through discussions between programmers and technicians on the spot. We hope, however, that the Mathilda Driver is a tool so flexible that any test problem, no matter how intricate, can be solved by using the facilities provided in the Driver in such a way that the test program in Mathilda may be written in a very simple-minded fashion.

We assume the reader has read the references [1] and [2].

Table of Contents page

0. INTRODUCTION

The Mathilda Driver [referredto as MD in the following] provides control and data generation facilities in RIKKE-1 for the support of the hardware testing of Mathilda. Mathilda may be completely driven via commands from RIKKE-1's console, and the size and complexity of the micro code test programs in Mathilda are reduced to a minimum, thus easing readability and reducing design time of the micro code routines in Mathilda.

1. MODE OF OPERATION

MD expects that the microprogram running in Mathilda has a specific structure. A general example looks like:

```
        A :   Read an address for a specific action
              from RIKKE-1
              goto specified address

address 1 :   Read any parameters from RIKKE-1
              necessary for action.
              Execute action 1
              goto A

address 2 :   Ditto as address 1
   .          .
   .          .
   .          .
address n :   .
```

Observe that the operator on RIKKE-1 must know the precise structure of the microprogram in Mathilda; e.g. the addresses of the various actions, and the number of reads from RIKKE-1 and writes to RIKKE-1. Any kind of error in the command sequence from RIKKE-1 may cause a fatal deadlock in the communication between RIKKE-1 and Mathilda. This will usually require a complete restart of Mathilda and MD.

2. COMMAND STRUCTURE

A command to MD consists of a list of single commands, separated by delimiters if necessary. [Delimiters may be omitted if the resulting concatenation of single commands cannot cause an ambiguous interpretation.]

A single command consists of a letter, a double letter, or one of the latter followed by an integer or a bit pattern or both. An integer is always interpreted as a decimal integer. A bit pattern is a sequence of 0's and 1's. The pattern may be no longer than 16 bits.

A delimiter is inserted in the command string by typing VT on the keyboard. A '$' is echoed on the console. The command string is terminated and immediately executed by typing two delimiters in succession.

A single character in the command string may be deleted by typing RUBOUT. The character is echoed on the console. An incomplete command string may be deleted entirely by typing CTRLx. The prompting '#' is echoed on the console.

MD is equipped with 4 macro buffers. A command string may be inserted into a macro; the macro may then be executed any number of times required by the operator. If an error is detected during execution, the operation is terminated. A macro definition may be called within another macro definition, but recursive definitions, i.e. a macro calling itself, are not allowed.

3. CONTROL AND DATA GENERATION FEATURES

MD can manipulate buffers in different ways. A list of the buffers contained in MD follows:

Output Buffer.

1 buffer, 64 bits wide, primarily used for sending data to Mathilda.

Control Buffer.

1 buffer, 64 bits wide, used for sending control information to Mathilda, i.e. pointer and function control. Only the 16 least significant bits are accessible.

Address Buffer.

1 buffer, 64 bits wide, used for sending sequencing information to Mathilda. Only the 16 least significant bits are accessible.

Add Buffer.

1 buffer, 64 bits wide, used to contain a 64 bit constant which may be added to the Output Buffer during execution of a command sequence. The addition is performed in groups of 16 bits [63-48, 47-32, 31-16, 15-0]. No carry is taken across the 16 bit bounds.

Input Buffer.

1 buffer, 64 bits wide, used to receive data from Mathilda.

External Register Buffer.

1 buffer, 16 bits wide, used to initialize Mathilda's External Register.

Scratch Buffers.

8 buffers, 64 bits wide, numbered Ø through 7, used for various purposes, e.g. read, write, comparison, etc. The reader will get full details in the list of single commands.

MD is equipped with the following data generation facilities:

Pattern Generation.

Any bit pattern, 0-16 bits wide, may be

1) copied across the 64 bits of the Output Buffer.
2) inserted at any bit position in the Output Buffer.
3) added in at any bit position of the Output Buffer.

Bit Manipulation.

Any bit in the Output Buffer may be

1) inserted [the rest of the bits left untouched]
2) set [the rest cleared]
3) reset [the rest left untouched]

Mask Generation.

A mask generator, equivalent to the Post Mask Generator in Mathilda, is provided to work on the Output Buffer.

Comparison.

The contents of the Input Buffer may be compared with the Output Buffer or any of the scratch buffers. Depending on the result of the

comparison, a choice between 2 different command string executions is made.

Shift.

The contents of the Output Buffer may be shifted cyclic through 64 bits either left or right. The contents of the External Register and Control Buffers may be shifted left 0-15 bits (logically) during transmission to Mathilda. The actual contents are left untouched.

Instruction Generation.

Any micro instruction for Mathilda may be generated with a simple Micro Assembler included in MD. The operator specifies which field in the instruction (i.e. f1, s1, etc.) he wishes to initialise followed by the code in decimal. The micro instruction is placed in the Output Buffer.

64 bit ALU.

MD is equipped with a 64 bit ALU, equivalent to the hardwired ALU in Mathilda. It contains the same functions and operates in the same fashion. A-input is always the Output Buffer, B-input may be any of the Scratch Buffers, and the result is delivered in Scratch Buffer Ø. The function of the ALU is specified in a separate command.

Display.

The contents of any buffer can be displayed either on the console or on the printer.

Load.

MD can read binary paper tape micro programs generated by MARIA in bootstrap loader format [i.e. default format], and transform them to something readable for Mathilda. A special micro program, called the Bootstrap-Normalizer, has been written to enable the loading of such programs from MD. This implies that test programs need not be confined to 16 words, but can use as many control store words as necessary. See appendix A for the source text of the Bootstrap-Normalizer.

4. SINGLE COMMANDS

The following is list of the single commands provided in MD, with a short description of each.

a = <n> Load Address Buffer with <n>.

al = <n> Load ALU function selector with <n>. Any activation of the ALU in MD following this command will give the result of the specified function. Possible functions are :

0	A
1	A ∨ B
2	A ∨ ¬ B
3	-1
4	A + (A ∧ ¬ B)
5	(A ∨ B) + (A ∧ ¬ B)
6	A + ¬ B
7	(A - B) -1
8	A + (A ∧ B)
9	A + B
10	(A ∨ ¬ B) + (A ∧ B)
11	(A ∧ B) -1
12	A + A
13	(A ∨ B) + A
14	(A ∨ ¬ B) + A
15	A - 1
16	¬ A
17	¬ (A ∨ B)
18	¬ A ∧ B
19	0
20	¬ (A ∧ B)
21	¬ B
22	A NEQV B
23	A ∧ ¬ B
24	¬ A ∨ B
25	A EQV B
26	B
27	A ∧ B
28	All '1's
29	A ∨ ¬ B
30	A ∨ B
31	A
32	
33	
34	
35	
36	
37	
38	
39	

40 41 42 43 44 45 46 47	As functions 0 - 15, plus 1, e.g. 32 : A + 1 39 : A - B
ac	Load ALU function selector from Control Buffer's 16 least significant bits.
al <n>	Activate ALU with function given in ALU function selector. A-input is Output Buffer and B-input is Scratch Buffer <n>. Result is delivered in Scratch Buffer ∅.
bc <n>	Clear bit <n> in Output Buffer. All other bits left untouched.
bi <n>	Insert bit <n> [set bit <n>] in Output Buffer. All other bits left untouched.
bs <n>	Set bit <n> in Output Buffer. All other bits are cleared.
c = <n>	Load Control Buffer with <n>.
c +	Increment Control Buffer.
c -	Decrement Control Buffer.
cp<str1>/<str2>	Compare Input Buffer with Output Buffer. If buffers are logically equivalent, execute command string <str1>, otherwise execute command string <str2>. Note: <str2> must <u>not</u> contain any delimiters.
cp<n> <str1>/<str2>	As command above, where Output Buffer is exchanged with Scratch Buffer <n>.
da	Display Address Buffer on printer.
db	Display Output Buffer and Input Buffer, in that order, on printer.
dc	Display Control Buffer on printer.
de	Display External Register Buffer on printer.
di	Display Input Buffer on Printer.
dj	Display Control Buffer and Input Buffer, in that order, on printer.

do	Display Output Buffer on printer.
du	Display ALU function selector on printer.
d<n>	Display Scratch Buffer <n> on printer.
e = <n>	Load External Register Buffer with <n>.
e +	Increment External Register Buffer.
e -	Decrement External Register Buffer.
f	Give form feed on printer.
h	Terminate execution of current command string.
i <field1>...<fieldn>$	Create micro instruction in Output Buffer. <fieldk> contains field description and value to be inserted in field, and must have the following format: <fd> = <m>$ <fd> may be written as: f1 f3 vs bi (bisb) s1 s3 ds ci (cisb) m2 m4 be (enable) sc f2 f4 bd af m3 as bs at The field specified by <fd> is then set to the value <m>. <m> is masked to suit the bit width of the field.
l	Load Mathilda with contents of binary paper tape in RIKKE-1's reader. Paper tape must be in Bootstrap Loader format from MARIA, i.e. default format. MD then transmits to Mathilda: 1) Load address in Control Store 2) Count of instructions-1 3) Instructions 4) Execution start address Note : Contents of Address Buffer, and Output Buffer are undefined after execution of 'l' instruction.
m<n>	Generate <n> 1's in Output Buffer.

if

<n> ≥ 0, generate <n> 0's, starting at most significant bit, set rest of bits to 1.

<n> > 63, generate 128 - <n> 0's, starting at least significant bit, set rest of bits to 1.

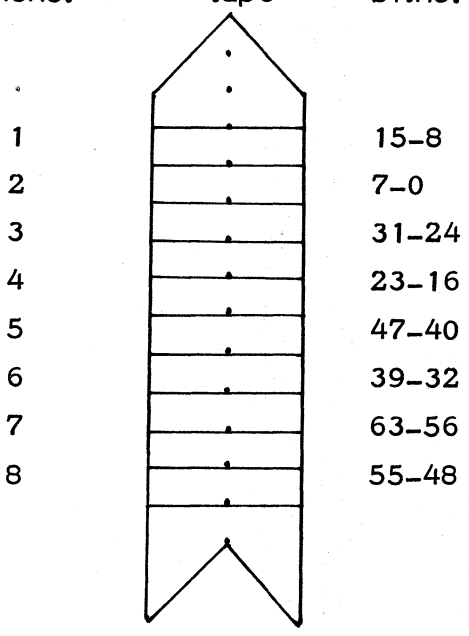
mc — Generate mask as above, but take control information from Control Buffer's 16 least significant bits.

oe — Punch approximately 15 cm paper feed [i.e. blank tape, no feed holes] on puncher.

of — Punch 8 frames with feedholes [no data] on puncher.

on<n> — Punch the integer <n> as 2 frames on puncher; the 8 most significant bits first followed by the 8 least significant btis.

o — Punch Output Buffer on puncher. The bits of the buffer appear on the paper tape as follows:



[This format is equivalent to various microloaders format for RIKKE-1 and Mathilda].

o<n>	Punch Scratch Buffer <n> [as above].
oa	Punch 8 Scratch Buffers in the same format as above on puncher, starting with Scratch Buffer ∅ and ending with Scratch Buffer 7.
pc<p>	Copy pattern <p> across Output Buffer. <p> may be any combination of bits and of length between 1 and 16 bits. <p> is then repeated as many times in the Output Buffer as there is room for it.
ps<n>, <p>	Set pattern <p> in Output Buffer starting at bit position <n>. All other bits are cleared.
pi<n>, <p>	Insert pattern <p> in Output Buffer, starting at bit position <n>. All other bits are left untouched.
pm	Move pattern in Output Buffer to Add Buffer.
pm<n>	Move pattern in Output Buffer to Scratch Buffer <n>.
pb	Move pattern in Add Buffer back to Output Buffer.
pb<n>	Move pattern in Scratch Buffer <n> back to Output Buffer.
pa	Add Add Buffer to Output Buffer. The adding is performed in 16 bit groups [63-48, 47-32, 31-16, 15-0] and no carry is transported between the groups.
pa<n>	Add Scratch Buffer <n> to Output Buffer.
q	Terminate MD session and return to opeting system.
r	Read from Mathilda into Input Buffer.
r<n>	Read from Mathilda into Scratch Buffer <n>.
rs	Reset Input Flag in RIKKE-1 for Mathilda. This instruction must be used with care. If the Input Flag is reset already, an input deadlock will occur.

s<n>	Shift Output Buffer cyclic <n> positions. if <n> > 0, execute left shift [i.e. from least significant towards most significant bits]. <n> < 0, execute right shift [i.e. from most significant towards least significant bits]. <n> = 0, no action
sc	Shift Output Buffer as above, take control from 16 least significant bit of Control Buffer.
ta	Display Address Buffer on console.
tb	Display Output Buffer and Input Buffer, in that order, on console.
tc	Display Constrol Buffer on console.
te	Display External Register Buffer on console.
ti	Display Input Buffer on console.
tj	Display Control Buffer and Input Buffer, in that order, on console.
to	Display Output Buffer on console.
tu	Display ALU function selector on console.
t<n>	Display Scratch Buffer <n> on console.
wa	Transmit Address Buffer to Mathilda.
wb	Transmit Output Buffer to Mathilda.
wc	Transmit Control Buffer to Mathilda.
wc<n>	Transmit Control Buffer, left shifted <n>, to Mathilda.
we	Transmit External Register Buffer to Mathilda's External Register.
we<n>	Transmit External Register Buffer, left shifted <n>,to Mathilda's External Register.
wc<n>	Transmit Scratch Buffer <n> to Mathilda.
vm<str>	Define contents of Macro Buffer v to <str>. The complete command string <str> defines v, no other command than the definition is executed.
v?	Display contents of Macro Buffer v on console.

v	Execute Macro v once.
v<n>	Execute Macro v <n> times.
xm<str>	Define contents of Macro Buffer x to <str>.
x?	Display contents of Macro Buffer x on console.
x	Execute Macro x once.
x<n>	Execute Macro x <n> times.
ym<str>	Define contents of Macro Buffer y to <str>.
y?	Display contents of Macro Buffer y on console.
y	Execute Macro y once.
y<n>	Execute Macro y <n> times.
zm<str>	Define contents of Macro Buffer z to <str>
z?	Display contents of Macro Buffer z on console.
z	Execute Macro z once.
z<n>	Execute Macro z <n> times.

5. TECHNICAL DETAILS

The Input Port B of Mathilda is connected to Output Port B, device 11 on RIKKE-1 in such a way that the 16 bits transmitted from RIKKE-1 are input in the Input Port's least significant bits. This means that the micro coded Read routine in Mathilda must read 4 times and shift and logically OR the results together to comprise a full 64 bit Mathilda data word. The current Standard Read Routine in Mathilda does this and expects RIKKE-1 to send the 64 bit word in descending bit order, i.e. first bits 63-48, then bits 47-32 etc. MD's transmit routine is devised to do this. See appendix B.

The Output Port B of Mathilda is connected to the Input Port B, device 11, on RIKKE-1. As in the input case, only 16 least significant bits may be transmitted. The current Write routine in Mathilda transmits a 64 bit word in descending bit order, and MD's read routine is devised to accept this. See appendix B.

The External Register of Mathilda is connected to Output Port B, device 8, on RIKKE-1. When MD transmits to the External Register Buffer, it sends the contents out on this line. There is no flag communication on this line, so logically speaking, Mathilda's External Register is currently equivalent to RIKKE-1's Output Port B device buffer 8.

MD is not in the RIKKE-1 operating system's resident library, but must be loaded and run as a user program.

6. CROSS REFERENCE INDEX

Appendix A.

```
**********************************************************************************************************************

43      READ:   ;                       ALF:=ALLOS,                     CA:=9733
44              AS:=AL,<16;                                                     ;R+2
45              LR:=IB;                 BSS:=CM,  CA+1,     IBA                 ;IF CA(3) THEN RA+1 ELSE R-1
46              ;                       ALF:=A@B, LRPC,                         ;IF IBDA THEN HERE-1 ELSE HERE

47      WRITE:  ;                               CA:=3
48              ;                       BSS:=CM                                 ;UNLESS OBSA THEN HERE
49              VS,OB:=VS,<16;          CA-1,                           OBA;    IF CA THEN RA+1 ELSE HERE-1
```

Appendix B.

```
 LINENO   CS ADDRESS
  0:              ************************************************************************************************************************
****
  1:
  2:              . BOOTSTRAPLOADER AND NORMALIZER FOR MATHILDA
  3:              .
  4:              . AT EXIT WE HAVE
  5:              .    MA[0] = LA[0] = LB[0] = PA[0] = NOMASK
  6:              .    MA[1] = PA[1] = PB[0] = FULLMASK
  7:              .    MAP = LAP = PAP = PBP = 0
  8:              .    LRP = 0
  9:              .    BSS = PGS = CM
 10:              .    RAP = RBP = 0
 11:              .    WAP AND WBP ARE UNCOUPLED
 12:              .    SCUALF IS SET TO A+B
 13:              .    KC AND KD ARE CLEARED
 14:
 15:              ************************************************************************************************************************
****
 16:
 17:      0               ;                   ALF:=     ALLOS,    PABC,     LBPC
 18:      1               ALL1S:              *****     MAPC,     PA:=BUS,  LAPC
 19:      2               MA:=ALL1S:          *****     LB:=SB,   SCUALF+,  LA:=SB
 20:      3               AL:                 MAP+1,    RAPC,     PB:=BUS,  PAP+1
 21:      4               MA:=AL:             IBA,      RA!,      PA:=BUS,  *****     ; R-READ
 22:      5               AL:                           SA:=SB,   RA!       *****     ; R-READ
 23:      6               AL:                 CB:=SB,   RA!,      PGS:=CM,  *****     ; R+READ.
 24:      7             * AL:                 RB!,      OC:=BUS,            KDC
 25:      8               ;                             CSLOAD,             ;         ; SA.
 26:      9               ;                   RA!,                          *****     ; R+READ.
 27:     10               ;                   CB-1,     KCC,                SA+1      ; UNLESS CB THEN RB.
 28:     11               AL:                 RBPC,     SA:=SB                        ; SA
 29:     12      READ:    ;                   ALF:=     ALLOS,              CA:=9733
 30:     13               AS:=AL,<16:         PAPC,     MAPC      *****     *****     ; R+2
 31:     14               LR:=IB:             BSS:=CM,  CA+1,     UNCPLB,   IBA       ;IF CA(3) THEN RA+1 ELSE HERE-1
 32:     15               ;                   ALF:=     A@B,      LRPC,     UNCPLA    ; IF IBDA THEN HERE-1 ELSE HERE
```

Appendix C.

The test program given in this example tests the Loading Mask A register group, LA. The LA mask, specified by the LA pointer, LAP, is active when Working Register A, WA, is loaded from the shifted bus.

The mask works as follows:

If bit i of the LA mask is 1 then
bit i of WA is bit i of the shifted bus.
If bit i of the LA mask is 0 then
bit i of WA is left unchanged.

We will now discuss the commands needed to test the LA mask register group.

First of all, the Bootstrap-Normalizer microprogram must be dead-started into Mathilda.

Mathilda is ready to be loaded with the test program. The program is set into RIKKE-1's reader and the command

```
I$$
```

is typed on the console. Mathilda loads the program into it's control store.

We are now interested in testing that the LAP pointer functions. This is accomplished by filling a count in each register of the group while adjusting the pointer, i.e. if LAP = 6, we fill 6 into the register pointed to by LAP. When we read the registers again we can check if this really happened. As the physical register is built of 8 bit modules each having their own pointer connection we wish to check that the pointer is correct for all 8 bit modules so we want a count in 8-bit groups. We do this by using the Add Buffer. We write:

```
pc00000001$ pm$$
```

This command sets 1 bit in each 8 bit group of the Output Buffer and puts it in the Add Buffer as well. We define the macro that is to initialise 1 register:

```
xm a = 21 $ we12 $ wa $ wb $$
```

(Please read the listing of the program while studying this macro).

Why "we12"? Because when LAP loads it's input from the External Register, it takes bits 15-12. The good hardware tester should always have these peculiarities in mind, and look them up every time - never trust your memory.

We define a macro that loads a register and increments the Output Buffer and the External Register.

```
ym x $ pa $ e + $$
```

We are ready to load the masks. We write:

```
e = 0 $$
y 16 $$
```

The LA registers are now initialised with the count, and we define a macro to read them and then we execute it.

```
zm a = 25 $  we12 $ wa $ r $ di $ e + $$
e = 0 $$
z 16 $$
```

The output from the printer is analyzed to detect any errors.

We encourage the reader to visualize the rest of the test process - or even try it!

```
 LINENO   CS ADDRESS
  0:              *ORG=17
  1:              *ENTRY=17
  2:
  3:              ********************************************************************************************************************
****
  4:
  5:              .LAP TEST ,TEST LAP FUNCTION AND LAS1,LAS2 . LA INPUT FUNCTION
  6:              ********************************************************************************************************************
****
  7:
  8:       17     START:    ;                 ALF:=ALLOS
  9:       18               WA:=AL
 10:       19               WA;               RA!                                   ;R-READ
 11:       20               AL;                         SA:=SB                      ;SA
 12:
 13:             ********************************************************************************************************************
****
 14:
 15:       21               ;                 RA!                                   ;R-READ
 16:       22               ;                 LAP:=EX
 17:       23               AL;                                     LA:=SB
 18:       24               ;                                                       ;R-START
 19:
 20:             ********************************************************************************************************************
****
 21:
 22:       25               WA;               LAP:=EX
 23:       26               WA:=ALL1S;
 24:       27               ALL1S;                                  LA:=SB
 25:       28               VS:=WA;           RA!                                   ;R-WRITE
 26:       29               ;                                                       ;R-START
 27:
 28:             ********************************************************************************************************************
****
 29:
 30:       30               ;                 LAS1:=EX,             LAPC
 31:       31               ;                 LAP:=S1
 32:       32               WA:=ALL1S;
 33:       33               ALL1S;                                  LA:=SB
 34:       34               VS:=WA;           RA!                                   ;R-WRITE
 35:       35               ;                                                       ;R-START
 36:
 37:             ********************************************************************************************************************
****
 38:
 39:       36               ;                 LAP:=EX
 40:       37               ;                 LAS2:=LAP,                          LAPC
 41:       38               ;                 LAP:=S2
 42:       39               WA:=ALL1S;
 43:       40               ALL1S;                                  LA:=SB
 44:       41               VS:=WA;           RA!                                   ;R-WRITE
 45:       42               ;                                                       ;R-START
 46:
 47:             ********************************************************************************************************************
****
 48:
 49:
 50:       43     READ:     ;                 ALF:=ALLOS,           CA:=9733
 51:       44               AS:=AL,<16;                                             ;R+2
 52:       45               LR:=IB;           BSS:=CM,  CA+1,     IBA               ;IF CA(3) THEN RA+1 ELSE R-1
 53:       46               ;                 ALF:=A@B, LRPC,                     ;IF IBDA THEN HERE-1 ELSE HERE
 54:
 55:       47     WRITE:    ;                           CA:=3
 56:       48               ;                 BSS:=CM                             ;UNLESS OBSA THEN HERE
 57:       49               VS,OB:=VS,<16;    CA-1,                 OBA;          IF CA THEN RA+1 ELSE HERE-1
```

REFERENCES

[1] P. Kornerup, Bruce D. Shriver:
A Description of the MATHILDA Processor
DAIMI PB-52, Department of Computer Science,
University of Aarhus, Denmark, july 1975.

[2] Eric Kressel, Ib Holm Sørensen:
The first BCPL System on RIKKE-1
DAIMI MD-17, Department of Computer Science,
University of Aarhus, Denmark, july 1975.