RCSL: 31-D14
Author: J. Lindballe,
H. Kold Mikkelsen
Edited: July 1971

TEST OF PERIPHERAL DEVICES

MAIN CHARACTERISTICS

KEY: RC 4000, Hardware Testprogram, Reference Manual

ABSTRACT: This paper describes the features of the loader 2. Further more some of the standard facilities of different testprograms are described.

A/S REGNECENTRALEN
Falkoneralle 1
DK 2000 Copenhagen F

CONTENTS:

Chapter 1: MAIN CHARACTERISTICS

PREFACE:

This paper is an extension of 51-VB431 by Jørgen Lindballe.
All testprograms designed for the previous loader will equally well run in loader 2.

1.1. INTRODUCTION

For each of the belowmentioned RC 4000 peripheral devices are made a number of test programs. The purpose of some of these is to check the peripheral device in question (checking programs), while the purpose of others is to help the operator to localize errors if these occur (motion programs):

Kind of Peripheral Devices	Number of Programs
Paper Tape Reader	2
Paper Tape Punch	4
Typewriter	4
Lineprinter, Data Product	3
Lineprinter, Anelex	2
Plotter	2
Magnetic Tape Station, 7 tracks	4
Magnetic Tape Station, 9 tracks	5
Drum	4
Disc	5
Interval Timer	1
Teletypewriter	4
Display	2

Besides the test programs are programmed partly a set of standard procedures used of the programs mainly for output on and input from the operator's typewriter and partly a loader program, the aim of which is to place the test programs of a given kind of peripheral devices and the mentioned procedures in the core store, because the test programs are used independent of the RC 4000 monitor and operative system (because it is a demand to these programs that they are possible to test any kind of peripheral devices within a core store of minimum size: 4096 words).

The loader 2, the test programs as well as the procedures, are written in SLANG.

A test by means of these programs demands perfectly operating:

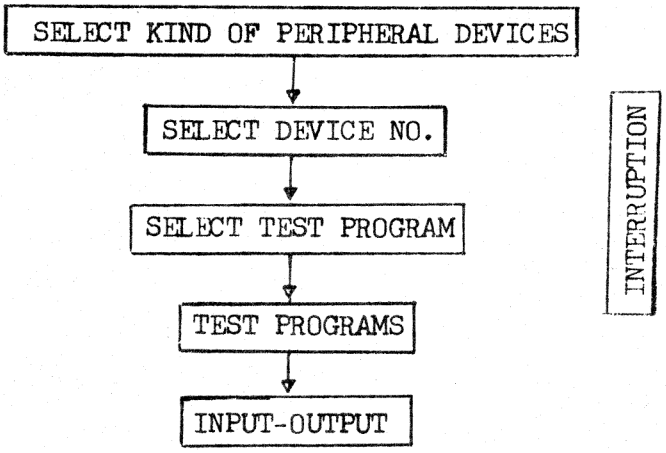
1. a central unit including
2. a core store not less than 4096 words.
3. a paper tape reader (device No. 0).
4. a typewriter (device No. 2).

Furthermore it should be desirable that

5. an operatorkey (interrupt channel No. 3), and
6. a magnetic tape station (7 or 9 tracks)

are available.

The abovementioned programs are linked together in accordance with this hierarchy:



so that, by the aid of the loader, all the test programs of a given kind of peripheral devices (say magnetic tape station or typewriter) can be placed in the core store, and after this the operator may select the device number and then the test program he wants to be executed. It is always possible to break the execution of a test program by pushing the operatorkey; after this he may select for the same device number a new test program, or for the same kind of peripheral devices he may select another device number, or he may load the test programs of a new kind of peripheral devices. (Finally, after such an operator termination, it is possible to have the contents of a part of the core store typed out).

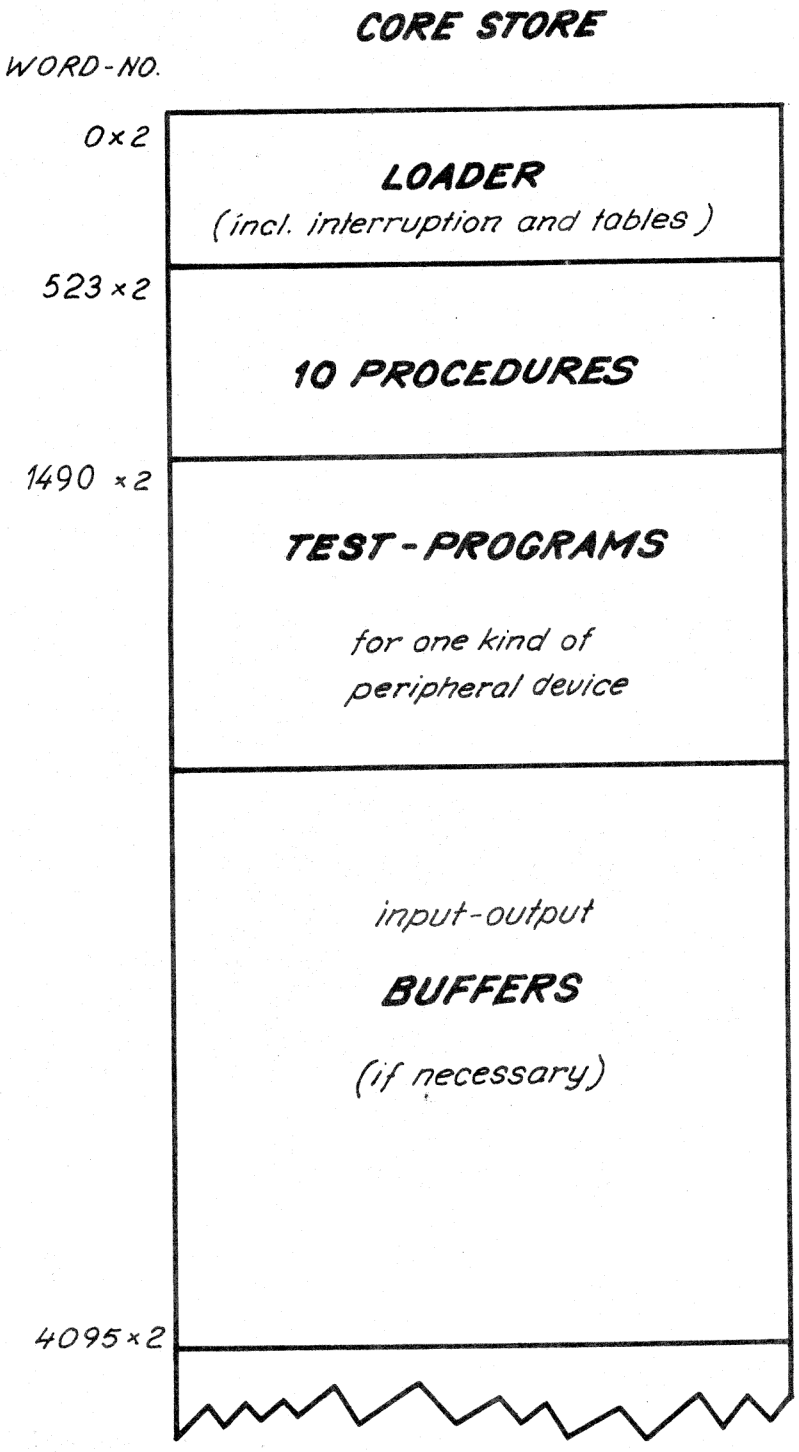
The loader 2 and the procedures are found in a binary version punched on a paper tape which may be read by the paper tape reader after activating the autoload pushbutton.

The test programs are found in a binary version both punched on paper tape (so that all the test programs for a given kind of peripheral devices are collected on one tape) and written on magnetic tape (so that all the test programs for a given kind of peripheral devices are collected in one file, and in such a way that each program forms a block).

Loader, procedures, and test programs are loaded as shown on the next page. It is noticed that only the first 4096 words are necessary for a test.

Before the detailed description of this complex of programs, it must be mentioned that everywhere in this report, the output on the typewriter from the programs is underlined so that it is not mistaken for the operator input.

Finally it is a rule that the operator, when he has carried out what a program has asked him to do, he must type in the character <NL>.



CORE STORE LAYOUT

1.2. THE LOADER 2.

When the binary tape, containing the loader and the procedures, has been placed in the paper tape reader (device No. 0), and the operator has activated the RESET and then the AUTOLOAD pushbutton, the loader is read into the core store by means of autoload word instructions in a 'bootstrap' as shown on the next page.

Next the loader, which occupies 523 words, is executed in a way explained on the following pages.

First the loader is initialized whereby words Nos. 16, 18, 20, and 22 are filled with the start addresses of 4 tables, which in this way are made available for all the programs which later on are placed in the core store. These tables are gradually filled with the following data:

TABLE 1 (startaddress in word 16):

word 1: The address of the first free word. This address is placed by the loader when the last test program has been loaded. It is used by the 7th procedure when a buffer area is reserved.

word 2: 2×the number of test programs (incl. the name (chapter 4)). This number is placed by the loader when the last test program has been loaded. It is used when the 9th procedure delivers the directory (a description of the test programs stored (chapter 1.3)).

word 3: Last word address of the core store + 2. This address is stored by the loader when initialized, and it is used by the 7th procedure.

word 4: When a test program, which tests the interrupt signal from the peripheral device, is initialized, it stores in this word the start address of its own interrupt sequence. Then, in case of interrupt signal from the peripheral device, a return jump from the loaders interrupt sequence to this start address is performed.

word 5: Device No. of operator's typewriter < 6. This device No. is stored by the loader when initialized, and it is used by the 1st and the 4th procedure.

word 6: Output dev. No. < 6 and is stored by the loader when initialized, and is used by the 1st and the 4th procedure.

The Loader Bootstrap

The Paper Tape

```
k

 0  aw                    2
 2  aw                    4
 4  jl                    0
 6  aw                 a401    (=x+0)
 8  aw          x1        4
10  aw                 a402    (=x+2)
12  al     w1   x1        2
14  aw                 a403    (=x+4)
16  jl.                   -4
18  jl     w1          a401    (=x+0)
20                        0

y a300:

x+0  a401:                0
x+2  a402:                0
x+4  a403:                0
x+6                       0
x+8                       0
x+10                      0
x+12 jl.    w1         a300.   (=y-x-12)
```

The Core Store

```
a

 0  aw                    2
 2  aw                    4  x+0 etc.
 4  jl                    0
 6
 8                        0

y-12  a300:                 ; initialize

x-6                       0
x-4                       0
x-2                       0
x+0   aw          x1      4  jl. w1 y-x-12
x+2   al    w1    x1      2
x+4   jl.                -4
```

TABLE 2 (startaddress in word 18):

word 1-10: Contain the addresses of the entry points of the 10 procedures. These addresses are stored by the loader successively when the procedures are stored. They are used by all of the programs.

TABLE 3 (startaddress in word 20):

word 1-16: Contain the addresses of the entry points of the name (chapter 1.4) and the test programs. These addresses are stored by the loader successively when the programs are stored. In connection with the administration of a test they are used by the 9th procedure. (chapter 1.3).

TABLE 4 (startaddress in word 22):

word 1,3,5: Device number(s) < 6
word 2,4,6: Interrupt channel number(s) ≠(-1)

When a test program is initialized, it fetches from here the device number(s), and if it tests the interrupt signal then the interrupt channel number(s) too.

When the loader has been initialized, it reads (IO-instructions) from the paper tape reader (device No. 0) the 9 procedures, and now it is able to communicate with the operator's typewriter.

The loader first writes:

channel no. of operator-key =

dev. no. of operator's typewriter =

and the operator types the interrupt channel No. of the operator key and the device No. of the typewriter he wants to use. (These numbers may be altered later by activating the RESET and then the START push-button after which the loader writes the above-mentioned questions again).

When after the message and question:

loader 2

input from device no.:

the operator has specified whether further inputs are wanted from the paper tape reader (device No. = 0) or from magnetic tape station (device No. > 0) (in the last case the typewriter writes:

file no.

and the operator must specify the file No. (chapter 1.4)), the test programs are loaded.

The loader now writes:

<name of the peripheral device>

After the questions:

device no.=

channel no.=

output dev. no.=

it reads the device number(s), the interrupt channel number(s) and the wanted output device number (> 0), it jumps with IM(operator-key) = 1, IR = 0 and interrupt enabled to the 9th procedure ('directory', chapter 1.3).

In case of interrupt No. 0 or when the operator-key is activated or in case of interrupt signal from the peripheral device (if the test program tests interrupt) a jump is performed to the loader's interrupt sequence, the start address of which is placed by the loader in word No. 12. A detailed description of this sequence is given in chapter 1.5.

The following rules apply to every program (a procedure or a test program) which is read and stored by the loader:

1. It is stored with the protection key = 0 so that every test is performed in the monitor mode.

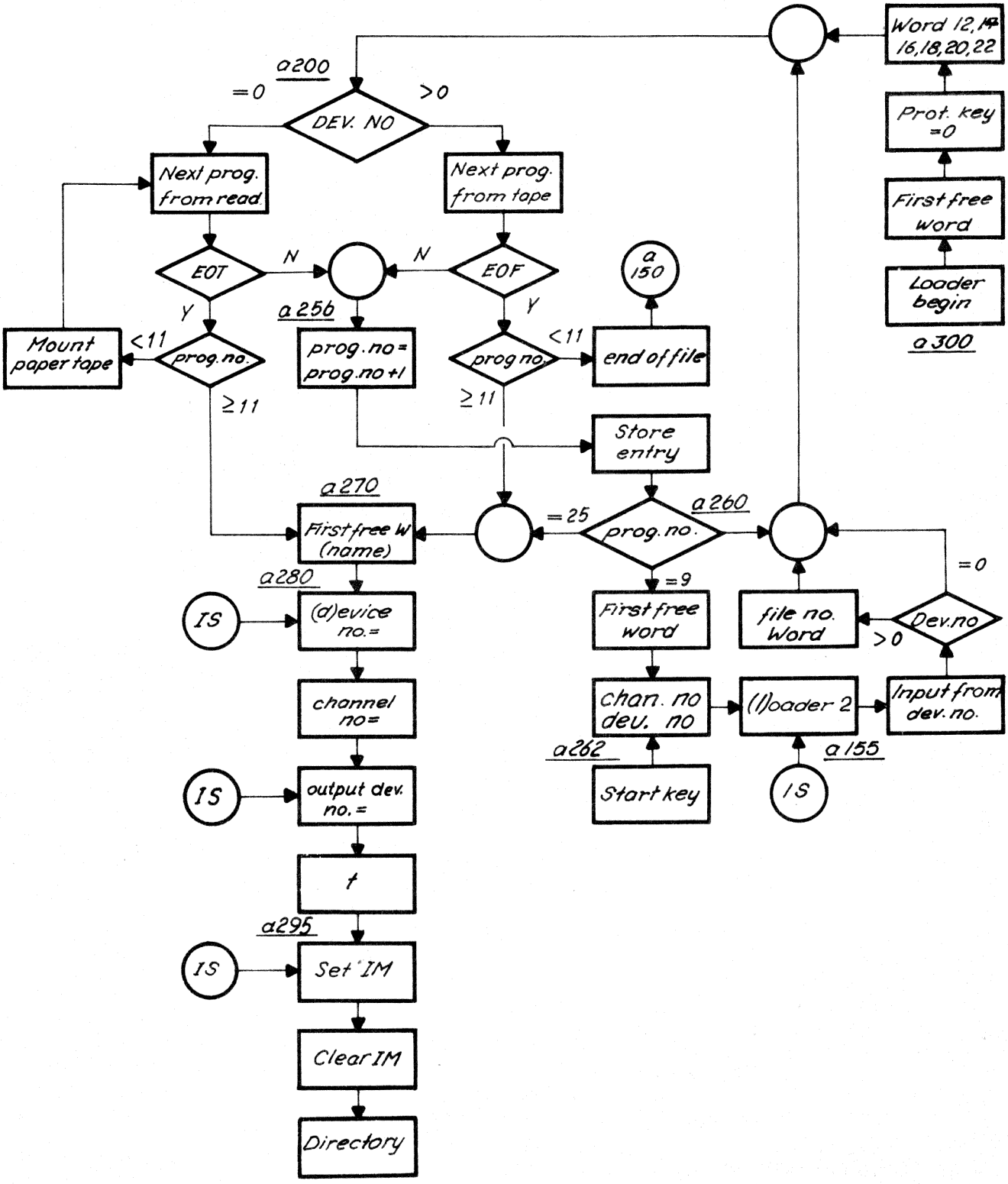
2. The parity and (when punched on paper tape) the check sum are examined, and in case of error, the following messages are given:

   parity error in <name of program>

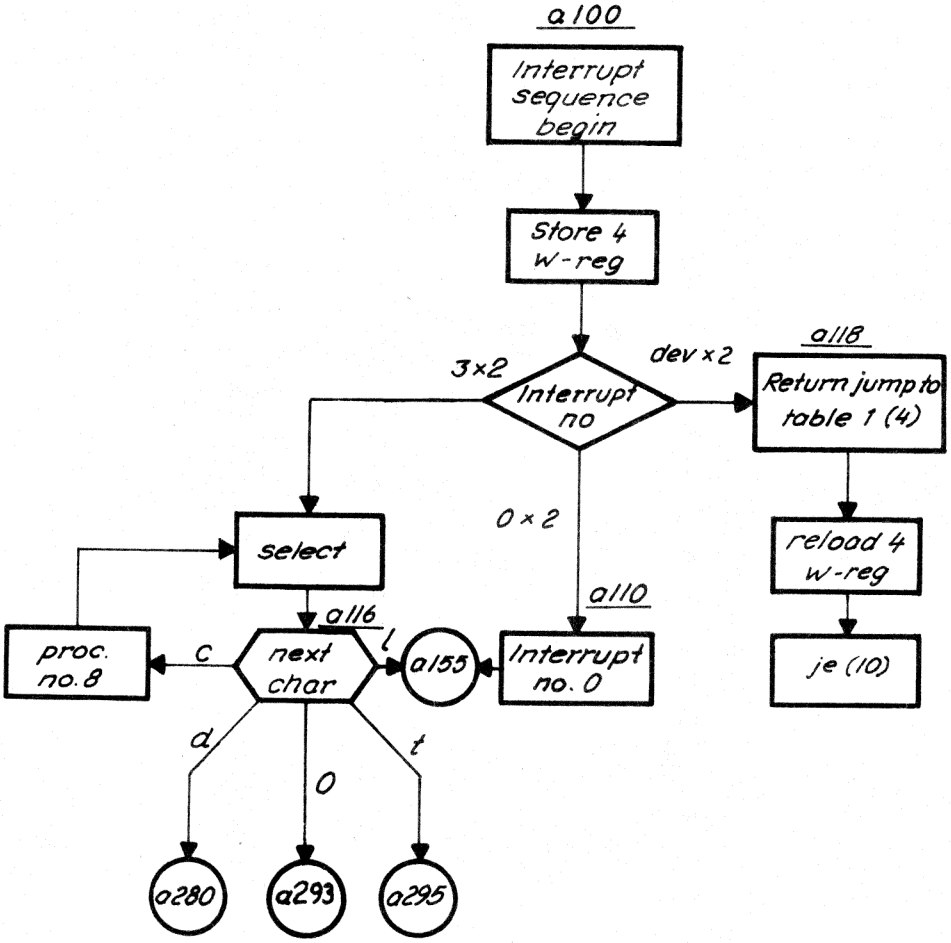
   check sum error in <name of program>

3. The first words of a program must contain a text string finished with the character <0> and giving a description of the program.

4. Entry point of the program must be the first word after this textstring.


a200
=0
DEV. NO
>0
Next prog. from read
Next prog. from tape
EOT
N
N
EOF
Y
Y
a 256
prog. no = prog. no +1
Mount paper tape
<11
prog. no.
≥11
<11
prog no.
≥11
a 150
end of file
Store entry
a 270
First free W (name)
=25
prog. no.
a 260
=9
=0
Dev. no
>0
file no. Word
First free word
Chan. no dev. no
a 262
Start key
(l)oader 2
a 155
IS
Input from dev. no.
Word 12,14 16,18,20,22
Prot. key =0
First free word
Loader begin
a 300
a280
IS
(d)evice no.=
channel no=
IS
output dev. no. =
t
a295
IS
Set IM
Clear IM
Directory

THE LOADER 2



THE INTERRUPT SEQUENCE

1.3. THE PROCEDURES

The first programs which are read (by means of the IO-instruction) and stored by the loader are the belowmentioned 10 procedures which in the binary version are punched on the same paper tape as the binary loader:

1. write a text
2. write a decimal number
3. write a binary number
4. read one character from the typewriter.
5. read a decimal number from the typewriter.
6. read a binary number from the typewriter.
7. reserve buffer area.
8. write the contents of a part of the core store.
9. administrate the test.
10. compare 2 binary words, write the result.

Each of these procedures, together occupying 967 words, are described in detail one by one on the following pages.

1st procedure: write a text

This procedure writes a text; the text, which may consist of an arbitrary number of characters, must be finished with the character <0>.

input	output
(w0) = + or - text start address	(w0) = undefined
(w2) = return address	(w2) = undefined

It may be called in this way:

```
al.  w0    <text start>.
am         (18)
jl   w2    (+0)
```

if output on the operators typewriter.

```
ac.  w0    <text start>.
am         (18)
jl   w2    (+0)
```

if output on the spec. output device.

2nd procedure: write a decimal number

This procedure writes in decimal a 24-bit integer. The integer may be negative, zero, or positive.

input	output
(w0) = + or - address of integer	(w0) = undefined
(w2) = return address	(w2) = undefined

It may be called in this way:

```
al.  w0     <int. addr.>.
am          (18)
jl   w2     (+2)
```

if output on the operators typewriter.

```
ac.  w0     <int. addr.>.
am          (18)
jl   w2     (+2)
```

if output on the spec. output device.

3rd procedure: write a binary number part 1

This procedure writes the leftmost bits of a word.

input	output
(w0) = + or - word addr.	(w0) = undefined
(w1) = No. of bits	(w1) = undefined
(w2) = return address	(w2) = undefined

It may be called in this way:

```
al.  w0    <word addr.>.
al   w1    <No. of bits>
am         (18)
jl   w2    (+4)
```

if output on the operators typewriter.

```
ac.  w0    <word addr.>.
al   w1    <No. of bits>
am         (18)
jl   w2    (+4)
```

if output on the spec. output device.

4th procedure: read one character

This procedure reads one character from the operator's typewriter.

input	output
(w2) = return address	(w2) = status and char.

It may be called in this way:

```
        am          (18)
        jl   w2     (+6)
```

<SP> is treated as a blind character.

If a parity error occurs, the character is replaced by a slash.

5th procedure: read a decimal number

This procedure reads a decimal integer typed on the operator's typewriter. The integer, which may be negative, zero, or positive, must be followed by a terminator (that is an arbitrary character which is not a digit or a space).

input	output
(w0) = undefined	(w0) = integer
(w2) = return address	(w2) = terminator

It may be called in this way:

```
        am          (18)
        jl    w2    (+8)
        sn    w2     10
        sh    w0      0
        jl.          -8
```

if the call demands an integer greater than 0 and a terminator equal to <NL>.

6th procedure: read a binary number

This procedure reads a positive binary integer typed on the operator's typewriter. The integer must be followed by a terminator (that is an arbitrary character which is not a 0 or a 1 or a space).

input	output
(w0) = undefined	(w0) = integer
(w2) = return address	(w2) = terminator

It may be called in this way:

```
        am          (18)
        jl   w2     (+10)
        se   w2      10
        jl.          -6
```

if the call demands a terminator equal to <NL>.

7th procedure: reserve buffer area

This procedure reserves a part of the core store and it writes on the operator's typewriter:

fbw = <address of first buffer word>

lbw = <address of last buffer word>

input	output
(w0) = No. of words wanted	(w0) = No. of words
(w2) = return address	(w2) = start address

If it was not possible to reserve the wanted number of words within the available core store, the output value of both w0 and w2 is 0.

The procedure may be called in this way:

```
al   w0     <No. of words>
am             (18)
jl   w2        (+12)
sh   w2          0
jl.
```

If the input value of w0 is equal to -(No. of words wanted) the procedure waits for input after having written <first buffer word>; if the operator inputs a character different from <NL> (for example /), it writes

fbw =

and waits for another start address. This address must be within the free part of the core store, i.e.

1) lower than the loader start address but not less than 24 (only significant when using the relocatable loader)
2) higher than the last address of the test programs.

If <fbw> is outside the free core the loader will request for a new <fbw>. After input from the operator it calculates and writes <last buffer word>.

8th procedure: write the contents of a part of the core store

This debug procedure is able to write on the operator's typewriter the contents of a specified part of the core store (from word No. 8) inone of four modes: text, decimal, binary, or machine instructions.

After an operator termination it is called when the operator types c; the core store area is specified when the procedure writes:

first word addr. =

last word addr. =

respectively, and the mode is selected when the operator types t, d, b, or i, respectively.
In case of d(ecimal) and b(inary) the program waits for input of an integer which specifies the print lay-out:

<answer>	lay-out
(new line)	24 bits
0 < <integer> <= 24	the word is divided into <integer> No. of bits from the left and printed. If d(ecimal) and if 24 mod <integer> ≠ 0 then print the rest word as a binary number.
<integer> = 0	24 bits

The procedure may be called in this way:

```
am          (18)
jl  w2      (+14)
```

9th procedure: administrate the test ('directory')

Immediately after the selection of the device No. (and the interrupt channel No.) for the peripheral device and the output device, a jump from the loader to 'directory' is performed. In this procedure the test program and the number of runs are selected, and furthermore the procedure is able to write on the operator's typewriter a description of the stored test programs.

After the question:

test program:

the operator may type a letter: a, b, c, ... and in this way select the 1st, 2nd, 3rd, ... test program, or he may type <NL>, after which the procedure writes the following directory:

a <description of the 1st test program>

b <description of the 2nd test program>

c <description of the 3rd test program>

.
.
.

<description of the last test program>

These descriptions are fetched from the first words of each program (chapter 1.4.).

When the operator has answered the question:

number of runs =

the test program is called 0th, 1st, 2nd, ..., last time. During call No. 0 the test program is initialized. At each call the return address is placed in w2, while

w1 (22) = last call
w1 (23) = 0th call

involving that run No. 0 and last run may be selected in this way:

```
sz   w1    1
jl          ; run No. 0 (initiate)
```

and

```
sz   w1    2
jl          ; last run (finish).
```

Before some runs 'directory' writes:

run no. <run No.>

that is before

1st, 2nd,..., 9th run, if 1 ≤ No. of runs ≤ 9
1st, 11th, 21st,..., 91st run, if 10 ≤ No. of runs ≤ 99
1st, 101st, 201st,..., 901st run, if 100 ≤ No. of runs ≤ 999

etc., so that a test is always introduced with the message

run no. 1

and so that a message is sent each time such a number of runs are executed that:

this number = the greatest 10-power which is less or equal the specified number of runs.

Having performed the wanted number of runs, the procedure writes:

test end

after which a new test program may be selected.

If the testporgram wants to finish the test before <No. of runs> are exceeded, it may return to the directory with return address:= return address +2. I.e. if w2 contains the return address the <test end> may be executed in the following way:

```
jl      x2 + 2
```

10th procedure: write a binary number part 2

This procedure compares two binary words and writes some of the leftmost bits in the following way: the two words are called 'received pattern' (rec.) and 'expected pattern' (exp.) respectively; the two words are compared bit by bit and if they are equal the value is written else one of the two letters ø or x is written:

ø if the bitvalue in rec. is 0 (a wrong zero),

x if the bitvalue in rec. is 1 (a wrong one).

input	output
(w1) = + or - table address	(w1) = undefined
(w2) = return address	(w2) = unchanged

It may be called in this way:

```
al.  w1    <table addr.>.
am           (18)
jl   w2      (+18)
```

if output on the operators typewriter.

```
ac.  w1    <table addr.>.
am           (18)
jl   w2      (+18)
```

it output on the spec. output device.

```
<table address> + 0:  <blocksize> <12 + <No. of bits>
                + 2:  bit pattern <received>
                + 4:  bit pattern <expected>
```

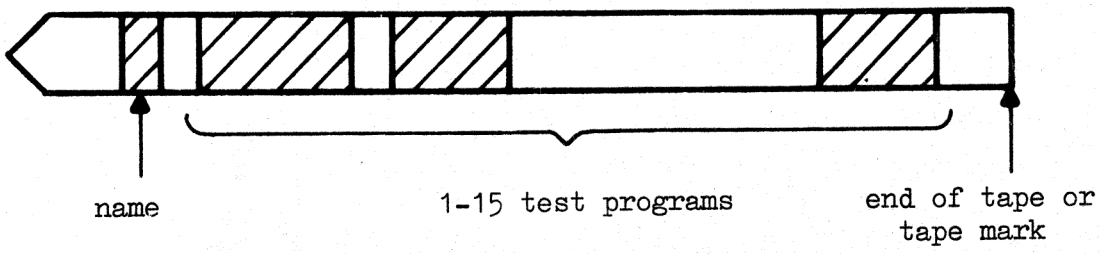
where <blocksize> denotes the number of bits to be printed before a <space>; if <blocksize> <=0 or <blocksize> > = <No. of bits> no <space>'s are printed. <No. of bits> denotes the total number of bits to be printed counted from left to right.

1.4. TEST PROGRAMS.

For each kind of peripheral devices mentioned below is made a set of test programs:

	File No.
RC 2000 Paper Tape Reader	1
RC 150 Paper Tape Punch	2
RC 315 Typewriter	3
RC 610 Lineprinter, Data Products	etc.
RC 333 Lineprinter, Anelex	
RC 4191 Plotter	
RC 707 Magnetic Tape Station, 7 tracks	
RC 709 Magnetic Tape Station, 9 tracks	
RC 4415 Drum	
RC 4314 Disc	
Interval Timer	
Teletypewriter	
DPC401 Display	

The test programs in the binary version exist both on paper tapes (so that all the programs for one kind of peripheral devices are punched on one paper tape) and on 7- or 9-track magnetic tape (so that each program forms one block, and so that all the programs for one kind of peripheral devices form one file. In both cases the parity is odd.



This drawing shows a paper tape or a file on magnetic tape containing the binary test programs for one kind of peripheral devices.

For each program (test program or procedure), which is read and stored by the loader, the following rules apply:

1. The first 15 words (that is the first 45 ISO-characters) must contain a text. This text is used by the loader in the message in case of parity error and check sum error, and it is used by the 9th procedure when writing the directory.

2. The 16th word must be the entry point.

3. The program must be finished with a check sum when punched on paper tape.

The paper tape/the file first contains a 15-word program containing the name of the kind of peripheral devices, for example:

<:rc 707 magnetic tape station, 7 tracks<0> :>

This is the text which is written by the loader immediately after the test programs are stored.

After the name follows a number of test programs in arbitrary succession; if the number exceeds 15, only the first 15 are loaded.

At the jump from 'directory' to a test program w2 contains the return address and

if 0th call then w1(23) = 1
if last call then w1(22) = 1

(the other bits are all 0) so that the test program may initiate and finish the test.

The test programs are divided into two groups:

Test programs:

1. Checking programs for critical check of the device.
2. Motion programs for uncritical use of the device.

Within each group for a given kind of peripheral devices the programs are successively numbered: 1.1, 1.2, ..., and 2.1, 2.2, ...

By means of the checking programs the complete, critical test of a peripheral device is performed. If the device does not react in the expected manner, messages mentioned in chapter 3 are given. The test of

sense, control, read, write
exception register
interrupt signals
status
data

is included in these programs. It is a principle that whatever happens, the test is going on. For example, the absence of an interrupt signal or even a disconnected device causes a message to the operator, but the test continues; but the operator may break the run by activating the operatorkey.

The exception register and the interrupt signal are tested as shown on the next page and as explained below:

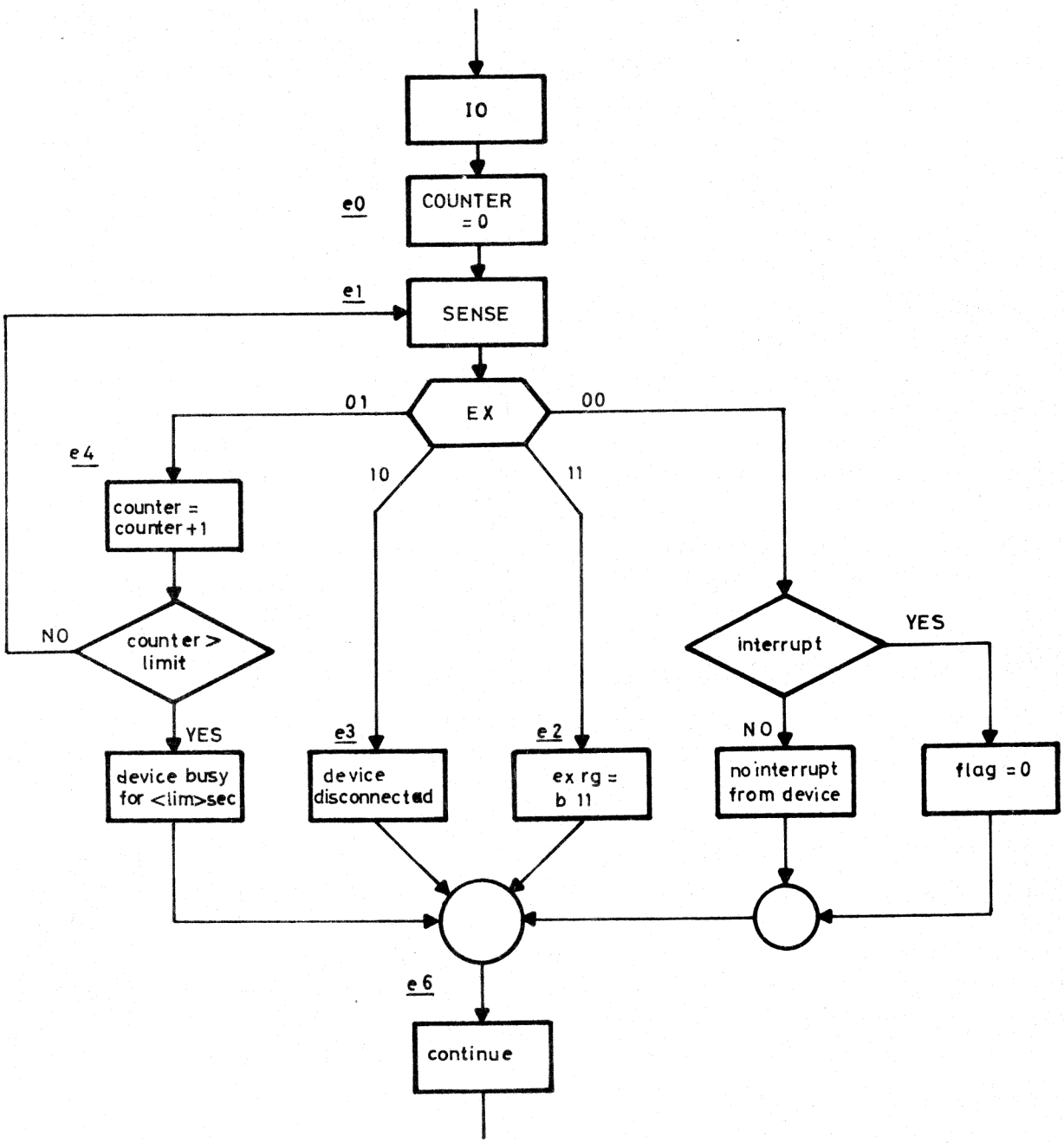
EX = 00: The device is available and, if it has sent an interrupt signal, the test continues; contrary this message is written (and the test continues):

no interrupt from device

EX = 01: The device is busy either because the transmission has not yet finished (especially because the device is in the local state) or because of a hardware error. If the device remains busy for a time depending on the kind of device, the program writes:

device busy for <time> sec.

and the test continues.


IO
e0
COUNTER = 0
e1
SENSE
EX
01
00
10
11
e4
counter = counter+1
NO
counter > limit
YES
device busy for <lim>sec
e3
device disconnected
e2
ex rg = b 11
interrupt
YES
NO
no interrupt from device
flag = 0
e6
continue

EX = 10: The device is disconnected either because of an operator oversight or because of a hardware error. After the message:

device disconnected

the test continues.

EX = 11: This is a hardware error which involves the message:

exception reg. = b11

After this the test continues.

Motion programs are commonly used when checking programs have shown an error. These programs use the peripheral devices in an uncritical way, that is they do not apply interrupt signals, the status word is not examined, and they hardly ever send error messages to the operator. In this way and by selecting a great number of runs it is possible to encircle the error, for example by oscilloscope measurements.

Each kind of peripheral device may be tested by means of a core store of minimum size that is 4096 words, but for high-speed devices it is possible to place the input-output buffer anywhere in the free part of the available core store.

The testprograms for high-speed devices are so designed that they propose the buffer start address by writing:

fbw = <address of first free word>

and wait for input. If the operator types <NL>, he accepts the start address; if he types a slash, the programs write:

fbw =

and he must input another start address. After input from the operator the address of the last buffer word is calculated and written. (ref. 7th procedure: reserve buffer area).

1.5. INTERRUPTION.

Interruption (that is interrupt signals, the interrupt register (IR), the interrupt mask (IM), and interrupt enabled/disabled) is applied in the following way:

When the loader is stored and executed and when the interrupt sequence is executed, interruption is disabled so that only interrupt No. 0 causes interruption. At all other times, that is during the execution of test programs and procedures, interrupt is enabled.

At the jump from the loader to directory the interrupt register is cleared, so that old signals from the operator key or other peripheral devices should not cause interruption. During the execution of directory IM(0) = IM(operator key) = 1, involving that only interrupt No. 0 or the use of the operator key causes interruption.

At the jump to a test program applying interrupt signals from the peripheral device furthermore IM (interrupt channel No.) is set to 1. (This mask is constructed by the test program when initiated; at the same time the test program stores the start address of its own subinterrupt sequence in word No. 4 of table 1 in the loader). So for these test programs interrupt signals from the peripheral device cause interruption. At the return jump from test programs the interrupt mask of directory is reloaded.

The interrupt sequence is placed inside the loader. Its start address is stored in word No. 12 by the loader when initiated. It is shown in the flow chart in chapter 1.2, and is now further described.

At the jump to the interrupt sequence, interrupt is disabled. Such a jump is performed in the following situations:

1. Interrupt No. 0 which may always occur. If the interrupt sequence and the first procedure are not destroyed, this message is written:

   interrupt no. 0

   If this occurs (due to a hardware- or software-error), the loader should be stored again.

2. By activating the operator key during the execution of a test program, directory or one of the other procedures. The interrupt sequence writes:

select

after which the operator may type t, o, d, l, or c. The typewriter now continues to write one of the following underlined texts:

test program:

A jump to directory is performed, and the operator may now select a new test program for the same device No.

output dev. No.=

and the operator may select a new output device number > 0. Hint: when using the cpu-timer as output dev. No. even checking programs may be used as motion porgrams. This can only be used for testprograms newer than medio 1971.

device no.

A jump to the statement in the loader where the device number(-s) is(are) selected. In this way the operator may select a new device No. (and after this a new interrupt channel No.) for the same kind of peripheral device

loader 2

A set of test programs for a new kind of peripheral device may be loaded.

core store contents

A jump to procedure No. 8 (chapter 1.3) is performed.

3. At interrupt signals from the peripheral device during the execution of a test program which applies interrupt. In this case a return jump is performed to the subinterrupt sequence of the test program with interrupt still disabled. w2 contains the return address. The start address of this sequence is stored by the test program when initiated in word No. 4 of table 1 in the loader. The contents of the 4 W-registers and the exception-register are stored by the loader's interrupt sequence, and the registers are reloaded just before a jump with interrupt enabled is performed to the broken test program.

1.6. BITPATTERNS.

In some test programs (for example RC 2000 1.2 and RC 150 1.2) a bitpattern consisting of 304 8-bit characters is generated. The decimal values of these characters are shown in the table on the next page.

The character set has the following proporties:

1. Except for 8 zeroes and 8 ones it contains (one or more times) all characters which are composed by 8 bits.

2. During the generation of the characters row by row each of the 8 bitpositions is activated in a very irregular way.

3. The programming of the generation is rather simple.

1	2	4	8	16	32	64	128
3	6	12	24	48	96	192	129
5	10	20	40	80	160	65	130
9	18	36	72	144	33	66	132
17	34	68	136	17	34	68	136
7	14	28	56	112	224	193	131
13	26	52	104	208	161	67	134
11	22	44	88	176	97	194	133
25	50	100	200	145	35	70	140
19	38	76	152	49	98	196	137
21	42	84	168	81	162	69	138
41	82	164	73	146	37	74	148
15	30	60	120	240	225	195	135
29	58	116	232	209	163	71	142
27	54	108	216	177	99	198	141
53	106	212	169	83	166	77	154
51	102	204	153	51	102	204	153
45	90	180	105	210	165	75	150
85	170	85	170	85	170	85	170
170	85	170	85	170	85	170	85
210	165	75	150	45	90	180	105
204	153	51	102	204	153	51	102
202	149	43	86	172	89	178	101
228	201	147	39	78	156	57	114
226	197	139	23	46	92	184	113
240	225	195	135	15	30	60	120
214	173	91	182	109	218	181	107
234	213	171	87	174	93	186	117
236	217	179	103	206	157	59	118
230	205	155	55	110	220	185	115
244	233	211	167	79	158	61	122
242	229	203	151	47	94	188	121
248	241	227	199	143	31	62	124
238	221	187	119	238	221	187	119
246	237	219	183	111	222	189	123
250	245	235	215	175	95	190	125
252	249	243	231	207	159	63	126
254	253	251	247	239	223	191	127

The characters are generated by cyclic shifts and complementation of the 19 8-bit characters shown below.

These characters consist of 1, 2, 3, or 4 ones. After 8 cyclic shifts

$$19 \times 8 = 152 \text{ characters}$$

are obtained, among which 6 + 4 + 4 = 14 characters are doublets (from the patterns marked with ⁎ and ⁎⁎), i.e.

$$152 - 14 = 138 \text{ different characters.}$$

By complementation of each of these

$$138 \times 2 = 276 \text{ characters}$$

are achieved, among which 8 + 4 + 8 + 2 = 22 characters are doublets (from the patterns marked with ⁎⁎⁎), so the result is

$$276 - 22 = 254 \text{ different characters}$$

i.e. the characters 1 to 254. It is noticed that the 2 characters 0 and 255 are not included.

1	00000001			
2	00000011			
3	00000101			
4	00001001			
5	00010001		⁎⁎)	
6	00000111			
7	00001101			
8	00001011			
9	00011001			
10	00010011			
11	00010101			
12	00101001			
13	00001111			⁎⁎⁎)
14	00011101			
15	00011011			
16	00110101			
17	00110011		⁎⁎)	⁎⁎⁎)
18	00101101			⁎⁎⁎)
19	01010101	⁎)		

⁎) the pattern is repeated after 2 cyclic shifts.

⁎⁎) the pattern is repeated after 4 cyclic shifts.

⁎⁎⁎) the pattern is repeated after complementation and cyclic shifts.

In the test program RC 707 1.1 the generation of 126 bitpatterns is based upon the shift and complementation of the following 9 7-bit characters:

1	0000001
2	0000011
3	0000101
4	0001001
5	0000111
6	0001101
7	0001011
8	0011001
9	0010101

After 7 cyclic shifts

$$7 \times 9 = 63 \text{ different characters}$$

are optained.

After complementation of each of these

$$63 \times 2 = 126 \text{ different characters}$$

are achieved, i.e. the characters 1 to 126. The 2 characters 0 and 127 are not included.

In the test program RC 709 1.1 and 1.4 the generation of 522 bitpatterns is based upon the shift and complementation of the following 29 9-bit characters:

1	000000001
2	000000011
3	000000101
4	000001001
5	000010001
6	000000111
7	000001101
8	000001011
9	000011001
10	000010011
11	000010101
12	000110001
13	000101001
14	000100101
15	001001001 *)
16	000001111
17	000011101
18	000011011
19	000010111
20	000111001
21	000110011
22	000100111
23	000110101
24	000101101
25	000101011
26	001101001
27	001011001
28	001100101
29	001010101

*) the pattern is repeated after 3 cyclic shifts.

After 9 cyclic shifts

$$29 \times 9 = 261 \text{ characters}$$

are optained, among which 6 are doublets, i.e.

$$261 - 6 = 255 \text{ different characters.}$$

By complementation of each of these

$$255 \times 2 = 510 \text{ different characters}$$

are achieved, i.e. the characters 1 to 510. It is noticed that 2 characters 0 and 511 are not included.

1.7. THE RELOCATABLE LOADER 2

The relocatable loader 2 consists of the abovementioned loader and procedures, however so designed that the relocatable loader and the testprograms may be stored everywhere within the available core store, if the operator before activating the AUTOLOAD push-button puts the start address into w3. This start address must be so chosen that

$$0 \leq w3 \leq \text{length of core store} - (\text{length of relocatable loader} + \text{length of testprograms})$$

All lengths are measured in No. of bytes. The length of the relocatable loader is 2980 bytes.

When w3 = 0, the loader and the testprograms are stored as usual (see Chapter 1.1, page 5).