TECHNICAL MANUAL

CONTENTS:

RCSL: 51-VB431
Author: Jørgen Lindballe
Edited: November 1969

RC 4000
TEST OF PERIPHERAL DEVICES

MAIN CHARACTERISTICS

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

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

A test by means of these programs demands perfectly operating:

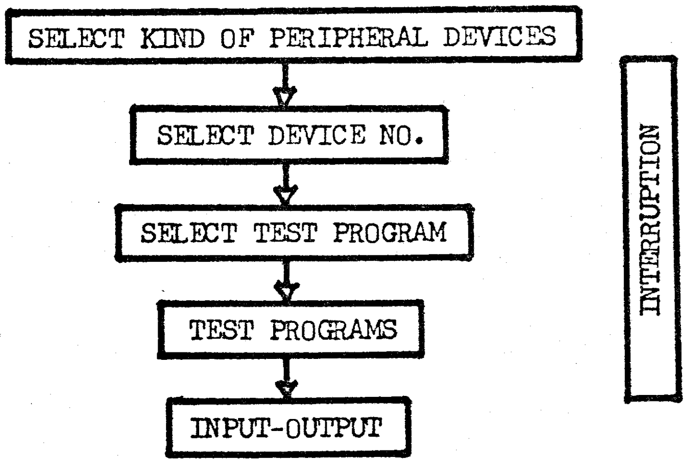
1. a central unit including
2. a core store not less than 4096 words.
3. a paper tape reader (device No. 0).
4. a typewriter (device No. 2).

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 on the typewriter).

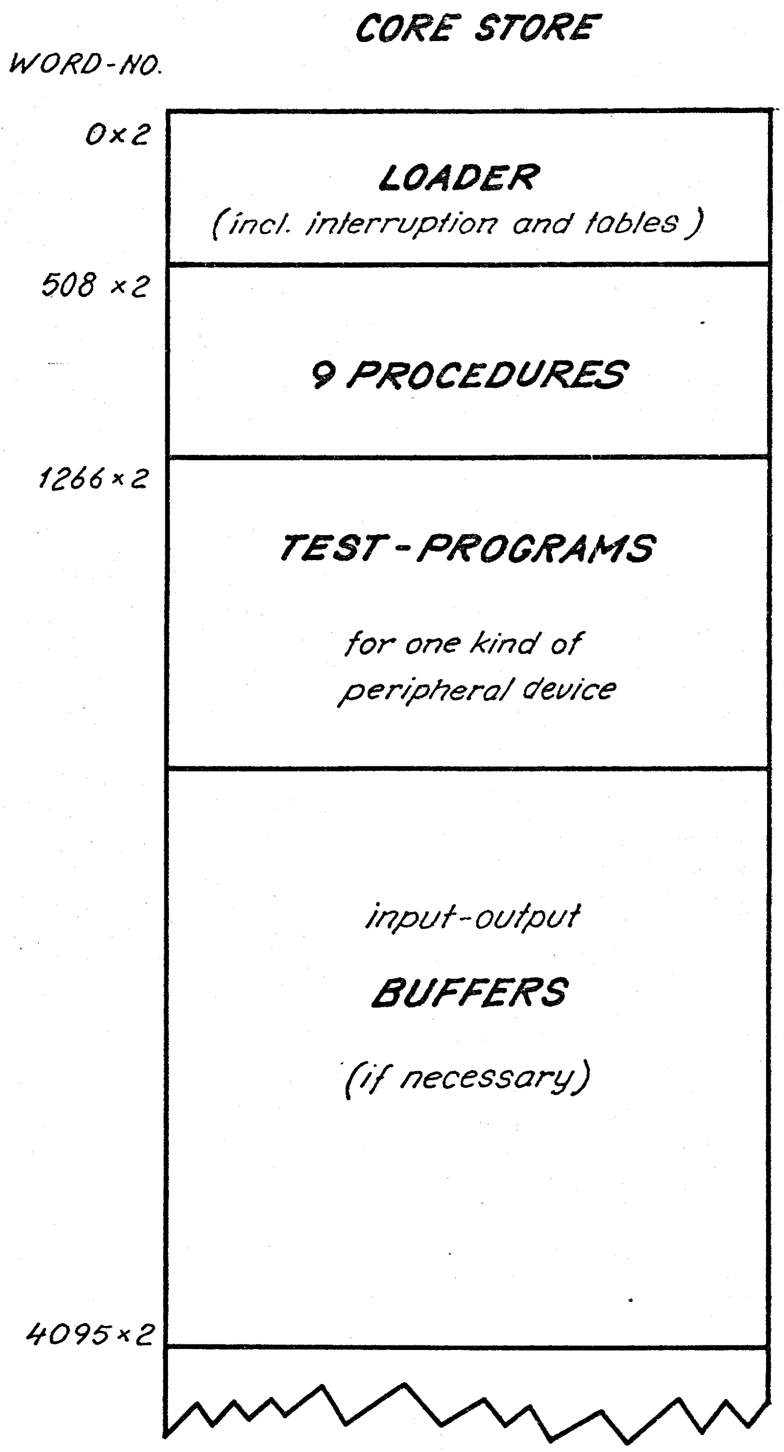
The loader and the procedures are found in a binary version punched on a paper tape which may be read by the paper tape reader after activating the autoload pushbutton.

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>.



Unit:	Designed	CORE STORE LAYOUT	Drawing No V20907
			Drawn by
A/S REGNE CENTRALEN	Approved		Checked
	Checked		____ Sheets / Sheet ____
	Last Revision		

Form. 305-A4-2.2.68

1.2. THE LOADER.

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 508 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.

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
```

VB431

TABLE 2 (startaddress in word 18):

word 1-9: Contain the addresses of the entry points of the 9 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) $\neq (-1)$

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

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.=

it reads the device number(s) and the interrupt channel number(s), 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.

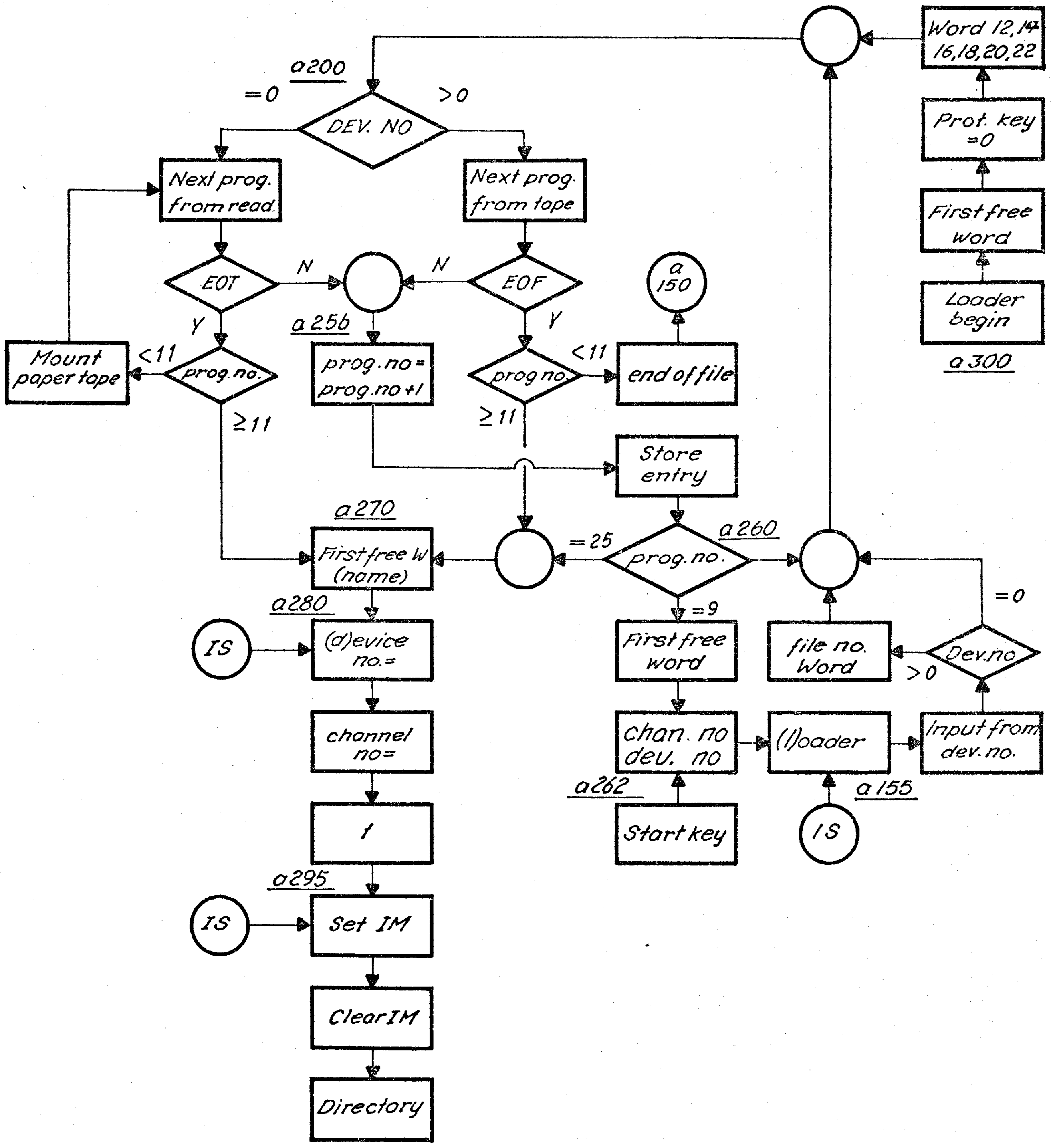
2. The parity and (when punched on paper tape) the check sum are examined, and in case of error, the following messages are given:

   parity error in <name of program>

   check sum error in <name of program>

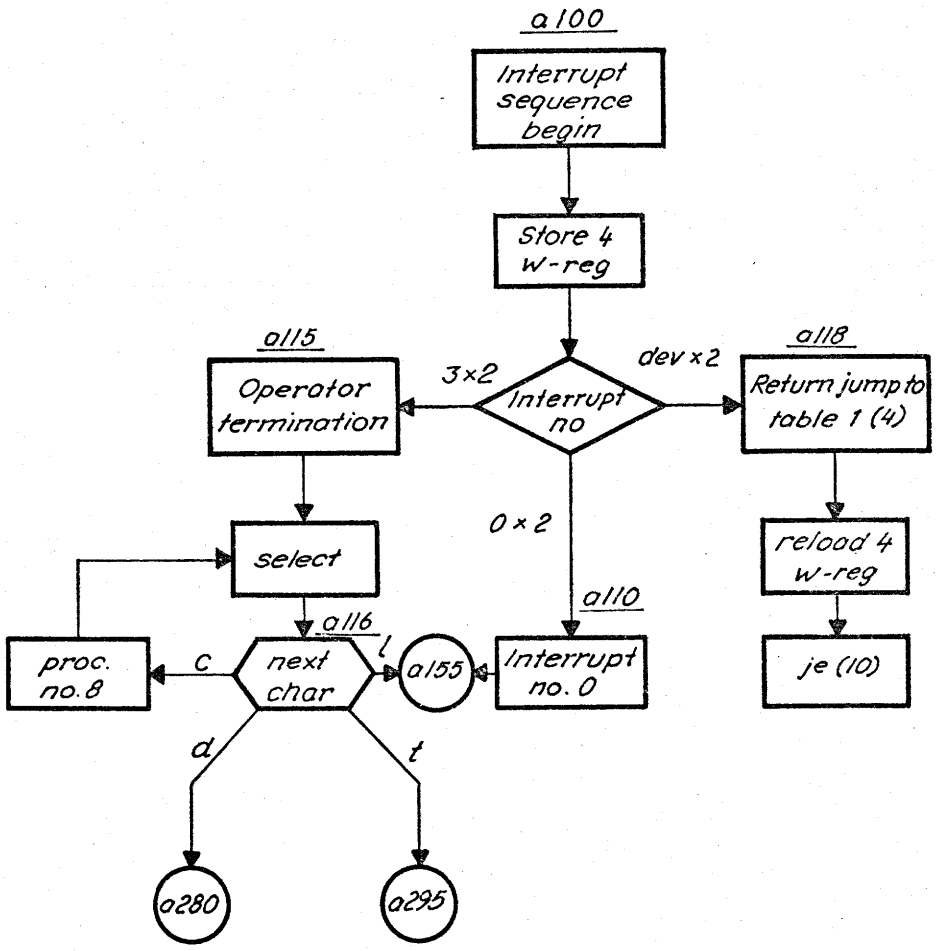
3. The first words of a program must contain a text string finished with the character <0> and giving a description of the program.

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



Unit:	Designed		Drawing No V20908
	Approved	THE LOADER	Drawn by
A/S REGNECENTRALEN	Checked		Checked
	Last Revision		___ Sheets · Sheet ___

Form 305-AA-2-2-68



Unit:	Designed	THE INTERRUPT SEQUENCE	Drawing No
A/S REGNE CENTRALEN	Approved		Drawn by
	Checked		Checked
	Last Revision		_____ Sheets / Sheet _____

1.3. THE PROCEDURES

The first programs which are read (by means of the IO-instruction) and stored by the loader are the belowmentioned 9 procedures which in the binary version are punched on the same paper tape as the binary loader:

1. write a text on the typewriter.
2. write a decimal number on the typewriter.
3. write a binary number on the typewriter.
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.

Each of these procedures, together occupying 758 words, are described in detail one by one on the following pages.

1st procedure: write a text

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

input	output
(w0) = 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)
```

2nd procedure: write a decimal number

This procedure writes in decimal a 24-bit integer on the operator's typewriter. The integer may be negative, zero, or positive.

input	output
(w0) = 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)
```

3rd procedure: write a binary number

This procedure writes on the operator's typewriter the leftmost bits of a word.

input	output
(w0) = 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)
```

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 (greater) start address. 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) in one 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.

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, 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

$$w1\ (22) = \text{last call}$$
$$w1\ (23) = \text{0th call}$$

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	$1 \leq$ No. of runs ≤ 9
1st, 11th, 21st,..., 91st run, if	$10 \leq$ No. of runs ≤ 99
1st, 101st, 201st,..., 901st run, if	$100 \leq$ 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

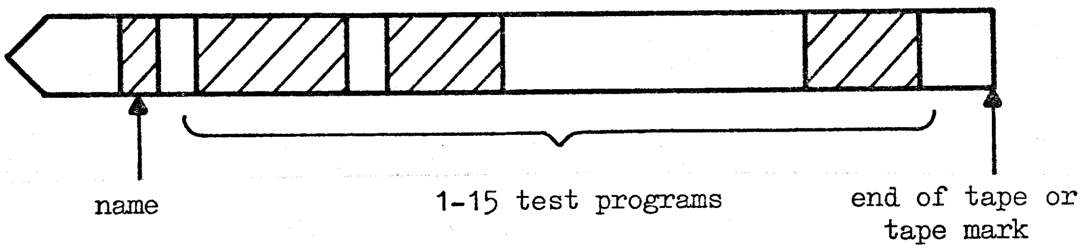
after which a new test program may be selected.

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:

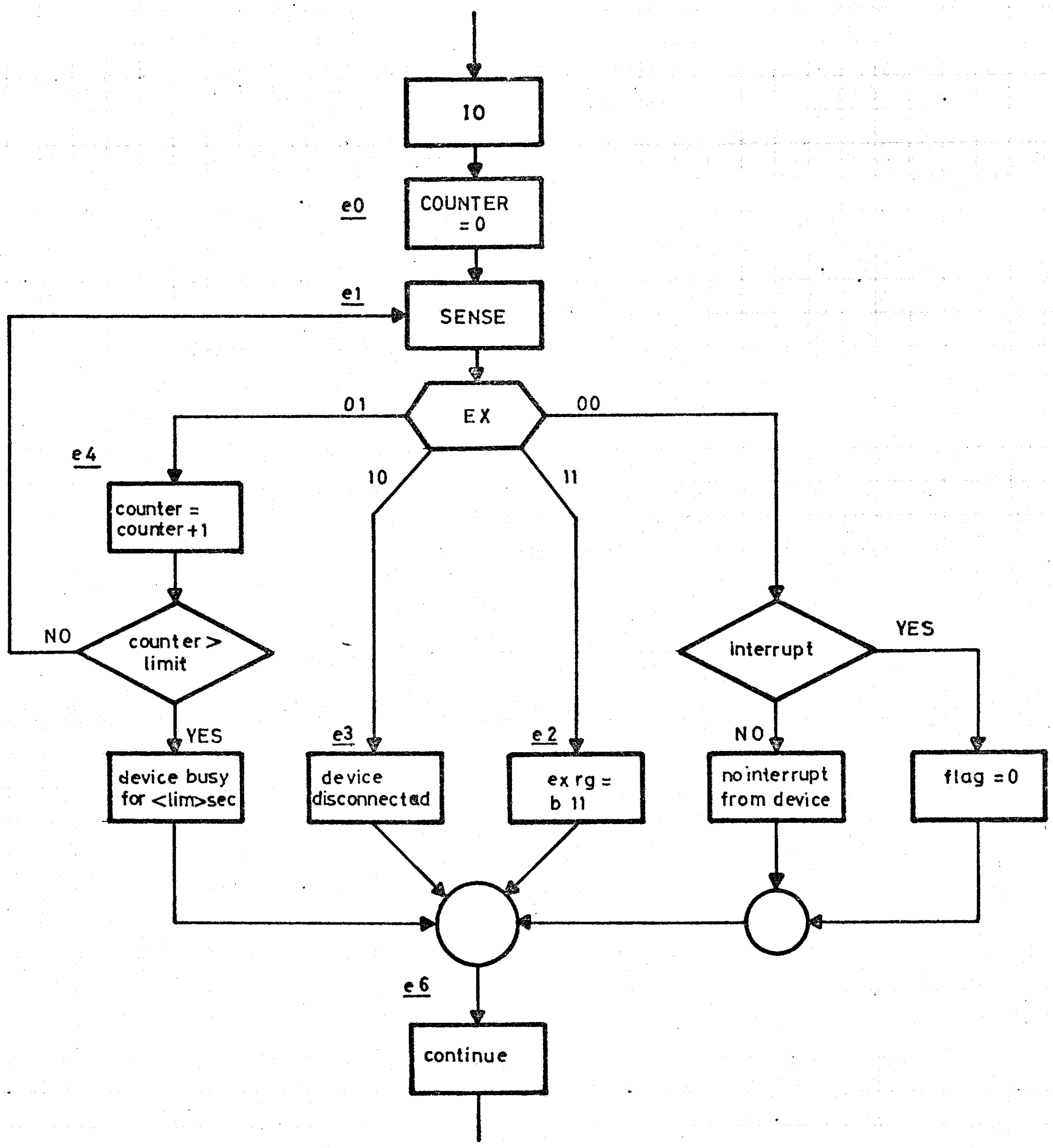
EX = 00: The device is available and, if it has sent an interrupt signal, the test continues; contrary this message is written (and the test continues):

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
counter > limit
NO
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 (higher) start address. After input from the operator the address of the last buffer word is calculated and written.

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:

   operator termination

   select

   after which the operator may type t, 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.

   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

   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. 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

$7*9 = 63$ different characters

are optained.

After complementation of each of these

$63*2 = 126$ 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

The relocatable loader consists of the above-mentioned 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

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

All lengths are measured in No. of bytes. The length of the relocatable loader is 2532 bytes.

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

RCSL: 51-VB473
Author: Jørgen Lindballe
Edited: November 1969

RC 4000

TEST OF PERIPHERAL DEVICES

OPERATOR's MANUAL

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS

CHAPTER 3: OPERATOR's MANUAL

LOADER, PROCEDURES AND INTERRUPTION

The loader (which reads and stores procedures and testprograms) and 9 procedures (used by the programs for output on and input from the operator's typewriter, for reservation of buffer area and for administration of the test) exist in the binary version on one paper tape.

This paper tape is read from the RC 2000 Paper Tape Reader (dev. No. 0) when the operator activates the RESET and then the AUTOLOAD push-button.

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).

The loader now writes:

loader
input from dev.no.:

and then the operator writes the number of the device, from which the binary testprograms shall be read, followed by <NL>. Consequently he must write 0, if input is wanted from the paper tape reader. If input is wanted from a magnetic tape station (dev. No. > 0), the loader asks:

file no.

and the operator writes the number of the file in which the testprograms are placed. These filenumbers appear from this table:

RC NO.	KIND OF PERIPHERAL DEVICE	FILE NO.
2000	Paper Tape Reader	
150	Paper Tape Punch	
315	Typewriter	
610	Lineprinter, Data Products	
333	Lineprinter, Anelex	
4191	Incremental Plotter	
707	Magnetic Tape Station, 7 tracks	
709	Magnetic Tape Station, 9 tracks	
4415	Drum	
4314	Disc	
	Interval Timer	
	Teletypewriter	
	Display	

When all the testprograms (not more than 15) for the kind of device are stored, the loader writes:

```
        <name of the kind of peripheral device>
        <device no. =
```

After this the operator types the device number (or the device numbers, as it is possible to write up to 3 device numbers separated by <comma>; e.g. teletypewriter). When the program hereafter writes:

```
        channel no. =
```

the operator types the interrupt channel number of the device (or the channel numbers, as it is possible to write up to 3 channel numbers separated by <comma>; e.g. teletypewriter).

Next the program writes:

```
        testprogram:
```

and the operator may type a or b or c or ... and in this way select the first, the second, the third, ... testprogram. If he types <NL> the following directory of the stored testprograms is written:

```
        a   <description of   1st testprogram>
        b   <description of   2nd testprogram>
        c   <description of   3rd testprogram>
        .                     .
        .                     .
        .                     .
            <description of the last testprogram>
        testprogram:
```

Having selected the testprogram the operator to the question:

```
        number of runs =
```

must write the number of times the program is wanted to be executed. This number must be chosen so that

$$1 \leq \text{number of runs} \leq 8\ 388\ 607$$

Before the execution of some runs it writes:

run no. <run no.>

namely before the execution of

```
        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.
```

Having executed the specified number of runs, the program writes:

test end

and now it is possible to select a new testprogram.

Using the loader some erroneous situations may occur:

If 'end of tape' appears in the paper tape reader or if 'tape mark' appears on magnetic tape when the loader reads testprograms before the first testprogram has been stored, it writes:

mount paper tape

end of file

respectively. If the operator types <NL> after the first message, the loader continues to read.

The error messages:

parity error in <program description>

and (when loading from the paper tape reader):

checksum error in <program description>

means parity error and checksum error in the program being loaded.

If bit 0, 2, 3, 4, 5 or 6 is set in the statusword when the programs are loaded from tape, the loader writes:

status = <bit 0-9>

and the programs must be loaded again.

It is always possible to break the execution of a testprogram by activating the operator key. After this the interrupt sequence writes:

operator termination

select

Then the operator may type t, d, l or c after which the typewriter continues to write the below-mentioned underlined texts:

testprogram:

The operator may select a new testprogram for the same device number.

device no. =

The operator may select a new device number (and after this as usual a new interrupt channel number) for the same kind of peripheral device.

loader

Now a new set of testprograms may be loaded, and in this way a new kind of peripheral device may be selected.

core store contents

The contents of some part of the core store may be written on the typewriter. When the program writes:

first word addr. =

last word addr. =

the operator specifies the part of the core store; it must be mentioned that for the testprograms which use input-output buffer, the addresses of the first and the last bufferword are written on the typewriter immediately before the execution of the first run.

When the program writes:

mode =

the operator types t, d, b or i after which the typewriter continues to write the below-mentioned underlined texts:

text

The bitpatterns are written as a text.

decimal

The bitpatterns are interpreted as integers (negative, zero or positive) and written in decimal.

binary

The bitpatterns are interpreted as positive integers and they are written in binary.

instruction

The bitpatterns are written as machine instructions including the mnemonic functioncodes.

In case of hardware or software error interrupt No. 0 may occur. This involves an error message:

interrupt no. 0

from the interrupt-sequence. In this case not only the testprograms but even the loader should be stored again.

By activating the RESET and then the START push-button, a jump to the loader is executed, and a new operator-key and -typewriter may be selected.

On the next page is shown some messages to and from the operator during a test.

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

fbw = <address of first free word>

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

fbw =

and he must input another (higher) start address. After input from the operator the address of the last buffer word is calculated and written.

Messages to and from the operator:

```
loader
input from dev. no.: 0
rc teletypewriter
device no.  = 16,17,18
channel no. = 22,23
testprogram:

a   1.2 read (echo)
b   1.3 write key - board
c   1.4 timer
d   2.1 sequence

testprogram: b
number of runs = 1

run no.        1
terminal disconnected

operator termination
select testprogram: c
number of runs = 10
time,expected:3000-6000 msec
time,measured:
run no.        1

operator termination
select loader
input from dev. no.: 0
rc 315 typewriter
device no.  = 2
channel no. = 7
testprogram: d
number of runs = 1

sequence =
abcdefghijklmn

run no.        1
abcdefghijklmn
test end

testprogram:
operator termination
select device no.  = 10
channel no. = 8
testprogram: b
number of runs = 2

run no.        1
run no.        2
test end

testprogram:
```

THE RELOCATABLE LOADER

The relocatable loader consists of the above-mentioned 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 2532 bytes.

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

RCSL: 51-VB432
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

RC 2000 PAPER TAPE READER

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

1.2 READ TEST TAPE

Purpose

In each run this program tests that an 8-track paper tape, punched in accordance with the description in chapter 1.6, is read correctly by the

odd parity -
even parity -
general

read command. Furthermore the exception register and the status word are examined, and it is checked that the reader detects parity errors. A partly check of the interrupt signals is performed by testing that an interrupt signal is delivered not later than 1 sec. after that an empty buffer has been recognized.

The program occupies about 310 words.

Initiation

Before the first run the program writes:

mount test tape

and the operator types <NL> after having mounted the test tape. This tape may contain the 304 characters (mentioned in chapter 1.6) 3 times per run, or it may contain the 304 characters 1 time only, forming a circular tape.

Test

In each run the character set is read first in odd parity, second in even parity, and third in general.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.1 sec., the message is:

device busy for 0.1 sec.

If no interrupt signal has been delivered 1 sec. after an empty buffer was recognized, the program writes:

no interrupt for 1 sec.

If bit No. 5 only is set in the status word, the program writes:

end of tape

and waits until the operator types <NL>, after which a new run is started.

If the status word contains an unexpected bitpattern, this message is given:

```
<parity>
read     <bit 0-23 of the status word, received>
expected <bit 0-23 of the status word, expected>
```

where

<parity>::= odd par.|even par.|general

2.1 READ ARBITRARY TAPE

Purpose

This program reads in an uncritical way (i.e. without testing the interrupt signals or the status word) an arbitrary paper tape.

The reading may be performed by means of the

odd parity -
even parity -
general

read command.

The program occupies about 120 words.

Initiation

When the program writes:

parity =

the operator may select the wanted read mode by typing o, e or g after which the program continues to write dd, ven and eneral, respectively.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.1 sec, the message is:

device busy for 0.1 sec

RCSL: 51-VB433
Author: Jørgen Lindballe
Edited: July 1969

RC 4000
TEST OF PERIPHERAL DEVICES

RC 150 PAPER TAPE PUNCH

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F.

CONTENTS:

RC 150 PAPER TAPE PUNCH

1.2 PUNCH AND READ

Purpose

This program tests the output of all bitpatterns punched on 8-track paper tape. The odd, the even as well as the general write-command are tested. Furthermore the exception register, the interrupt signals and the status-word are examined.

The program, which occupies about 350 words, demands a perfectly operating RC 2000 Paper Tape Reader (dev. No. 0).

Test

In a run 2x304 characters are punched. Each of the 2 character sets are composed as described in chapter 1.6; the first set, however, is punched in general, and the second alternately in odd parity (during the shift of 1, 3, 5 and 7 bits) and in even parity (during the shift of 2, 4 and 6 bits).

After the last run the program writes:

load rc 2000

and the paper tape is placed in the paper tape reader; when the operator types <NL> the bitpatterns are read (in general) and checked.

Error Messages

After a sense instruction to the paper tape punch the exception register is examined.

If EX = b11 this message is given:

exception reg. = b11

If EX = b10 the program writes:

device disconnected

If EX = b01 for more than 2 seconds, the message is:

```
device busy for 2 sec
```

If no interrupt signal has been received when EX = b00 the program writes:

```
no interrupt from device
```

If a bit is set in one or more of the positions 0-4 and 6-23 of the status word, the program writes:

```
status= <bit 0-23 of the status word>
```

In all cases the test continues after the message.

If bit 5 is set in the status word this message is given:

```
load rc 150
```

and the program waits until the operator types <NL>.

If a non-expected character is read by the paper tape reader, the program writes:

```
run no. <run no.>
<punch mode> parity
read        <read character, 8 bits>
expected    <expected character, 8 bits>
```

where

```
<punch mode> ::= odd|even|general
```

1.3 TIMER

Purpose

By means of this program it is examined whether the timer acts correctly.

The program occupies about 210 words.

Initiation

The program first writes:

disconnect rc 150 motor power

The operator types <NL> after having disconnected the power cable and in this way stopped the motor.

Test

In each run the program tries to punch a character, and it measures the time from the output of the character until bit No. 2 is set in the status word and an interrupt signal is received.

The program then writes:

timer: <time, measured> msec

The time, which is about 1 second, has to be at least 10 per cent higher than the start-up time of the motor (which is measured in the program 1.4).

After the last run the program writes:

connect rc 150 motor power

and the test is finished when the operator inputs <NL> after having connected the power cable.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11 this message is given:

exception reg. = b11

If EX = b10 the program writes:

device disconnected

If EX = b01 for more than 2 seconds, the message is:

device busy for 2 sec

If no interrupt signal has been received when EX = b00 the program writes:

no interrupt from device

If other bits than bit No. 2 are set in the status word the program writes:

status= <bit 0-23 of the status word>

In all cases the test continues after the message.

1.4 START-UP TIME

Purpose

This program measures the start-up time of the paper tape punch.

The program occupies about 190 words.

Test

In each run the program first waits for 4 seconds, so that the motor for certain is stopped; after this a character is output, and the program measures the time until the character is punched and an interrupt signal is received. Then the program writes:

```
start-up time = <time, measured> msec
```

The time has to be at least 10 per cent lower than the timer-time (which is measured in the program 1.3).

Error Messages

After a sense instruction the exception register is examined.

If EX = b11 this message is given:

```
exception reg. = b11
```

If EX = b10 the program writes:

```
device disconnected
```

If EX = b01 for more than 1 second, the message is:

```
device busy for 1 sec
```

If no interrupt signal has been received when EX = b00 the program writes:

no interrupt from device

If the status word contains a bit in one or more positions different from bit position No. 5, the program writes:

status = <bit 0-23 of the status word>

In all cases the test continues after the message.

If in the status word bit No. 5 is set, this message is given:

load rc 150

and the program waits until the operator types <NL> after having mounted paper tape.

2.1 PUNCH SEQUENCE

Purpose

This program punches in an uncritical way (that is without testing the interrupt signals or the status word) an arbitrary sequence of (not more than 50) characters.

The program occupies about 190 words.

Initiation

When the program asks:

sequence =

the operator types in decimal the wanted sequence of (non-negative) characters separated by <comma> and terminated by <NL>.

The program is able to punch in:

odd parity
even parity
general

The punch-mode is selected when the program writes:

parity =

by typing o, e or g, after which the program continues to write dd, ven and eneral respectively.

In odd and even parity the numbers are punched modulu 128, and in general they are punched modulu 256.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11 this message is given:

exception reg. = b11

If EX = b10 the program writes:

device disconnected

If EX = b01 for more than 1 second, the message is:

device busy for 1 sec

In all cases the test continues after the message.

If bit 5 is set in the status word the program writes:

load rc 150

and waits until the operator types <NL> after having mounted new paper tape.

RCSL: 51-VB434
Author: Jørgen Lindballe
Edited: July 1969

RC 4000
TEST OF PERIPHERAL DEVICES

RC 315 TYPEWRITER

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

RC 315 TYPEWRITER

1.2 READ (ECHO)

Purpose

This program tests input from the typewriter. Furthermore the exception register, the interrupt signals, and the status word are examined.

The program occupies about 160 words.

Test

When the operator inputs a character, the program outputs this character followed by <NL>.

After each input- and output operation the program tests the interrupt signal and the exception register. After each input operation the status word is also examined.

Error Messages

If a bit is set in the status word in one or more of the positions 0 to 1 and 3 to 16, the program writes:

read-status: <bit 0-23 of the status word>

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

In all cases the test continues after the message.

1.3 WRITE KEY-BOARD

Purpose

This program tests the output of all characters, defined as well as undefined on the typewriter. Furthermore the exception register, the interrupt signals, and the status word are examined.

The program occupies about 310 words.

Test

First the program writes all defined characters in 'upper case' (followed by <HT> and <BS>) and in 'lower case' (followed by <HT> and <BS>). The layout shown below corresponds to the key-board. After each character the program checks that the status word contains the character only.

Second the program outputs all undefined characters which does not cause any printing. After each character the program checks that the status word is 0.

```
!  "  :  +  %  &  ´  (  )  _  =  +
Q  W  E  R  T  Y  U  I  O  P  Å
A  S  D  F  G  H  J  K  L  Æ  Ø
Z  X  C  V  B  N  M  <  >  ?
<HT>     <BS>


1  2  3  4  5  6  7  8  9  0  -  ;
q  w  e  r  t  y  u  i  o  p  å
a  s  d  f  g  h  j  k  l  æ  ø
z  x  c  v  b  n  m  ,  .  /
<HT>     <BS>
```

Error Messages

If the bitpattern of the status word does not correspond to the expected one (that is the output character if this is defined, else all zeroes), the program writes:

```
                                   status-error:
                                   received <bit 0-23, received>
                                   expected <bit 0-23, expected>
```

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

```
                                   exception reg. = b11
```

If EX = b10, the program writes:

```
                                   device disconnected
```

If EX = b01 for more than 4 sec., the message is:

```
                                   device busy for 4 sec.
```

If no interrupt signal has been received when EX = b00, the program writes:

```
                                   no interrupt from device
```

In all cases the test continues after the message.

1.4 TIMER

Purpose

This program tests that the timer acts correctly. The program occupies about 180 words.

Initiation

Before the first run the program writes:

time, expected: <lower limit> - <upper limit> msec

time, measured:

where lower and upper limit are the time limits, that is 2 sec ±10 per cent, or 1800 and 2200 msec, respectively.

Test

In each run the program initiates an input operation and measures the time until an interrupt signal is delivered and timer status is set. Then the program writes:

<measured time> msec

Error Messages

If other bits than bit No. 2 are set in the status word, this message is given:

status = <bit 0-23 of the status word>

After the sense instruction the exception register is examined.

If EX = b11, the message is:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

2.1 SEQUENCE

Purpose

In each run this program writes in an uncritical way an arbitrary sequence of characters followed by <NL>. The program tests neither the interrupt signals nor the status word.

The program occupies about 180 words.

Initiation

When the program writes:

sequence =

the operator inputs the wanted sequence of (not more than 100) characters terminated by <NL>. The characters (exclusive < (iso character No. 60), <NL> (iso character No. 10), <SP> (iso character No. 32), and all undefined characters) may be input directly or they may be input by writing the decimal iso character No. enclosed in < and > ; for example

sequence = <60><0>>

causes the printing of < >.

Error Messages

After a sense instruction the exception register is examined:

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

In all cases the test continues after the message.

RCSL: 51-VB443
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

INTERVAL TIMER

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F.

CONTENTS

INTERVAL TIMER

1.1 TIME-MEASUREMENT

Purpose

By means of this program it is tested that interrupt signals are produced with correct timeintervals and that the binary counter is increased correctly.

The program occupies about 280 words.

Test

In each run all 11 interrupt intervals are tested, the first value being 1.6 msec and the last one 1638.4 msec. For each of these intervals the following happens:

First the program writes:

<interrupt interval, nominal>

expressed in msec, and when the operator types <NL> after having set the interrupt interval manually, the program measures the average time between 2 interrupt signals. Furthermore it measures the increment of the binary counter, measuring for 2xx10, 2xx9, ... , 2xx0 intervals respectively (i.e. for 1.6384 sec).

After this the program writes:

<average interrupt interval, measured ><counter increment +2xx14>

The expected value of (the increment of the binary counter + 2xx14) is 2xx14 = 16384. As shown on the next page this value is written on the typewriter just before the first measurement.

If a minor deviation (max. 0.02 per cent) from the expected interrupt interval is measured it may be due to the difference of the accuracy of the interval timer frequency and the frequency of the primary clock pulse generator.

```
rc interval timer
device no.  = 3
channel no. = 17
testprogram:

a 1.1 time-measurement

testprogram: a
number of runs = 1
expected counter increment = 16384
run no.       1

   interrupt  counts
    (msec.)

       1.6
       1.6    16384

       3.2
       3.2    16384

       6.4
       6.4    16384

      12.8
      12.8    16384

      25.6
      25.6    16384

      51.2
      51.2    16384

     102.4
     102.4    16384

     204.8
     204.8    16384

     409.6
     409.6    16384

     819.2
     819.3    16384

    1638.4
    1638.6    16384
test end
```

Error Messages

After the sense instruction the exception register is examined.

If EX = b11 this message is given:

exception reg. = b11

If EX = b10 the program writes:

device disconnected

If EX = b01 the message is:

device busy

If no interrupt signal is received for 2 seconds, the program writes:

no interrupt from device

In all cases the test continues after the message.

RCSL: 51-VB436
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

RC 610 LINE PRINTER, DATA PRODUCTS

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

RC 610 LINE PRINTER, DATA PRODUCTS

1.2 WRITE (SI AND SO)

Purpose

This program tests the shift-in character set as well as the shift-out character set. Furthermore the exception register, the interrupt signals, and the status word are examined. The controls <HT>, <CAN>, and <ESC> are tested by means of the program 1.3.

The program occupies about 400 words.

Initiation

A format tape containing holes for form feed (channel No. 0) and for vertical tabulation (channel No. 1) is mounted on the printer.

Test

The printer output from the test described below is shown on page 7-11. (This output is from a printer in which the <SO> character is blind).

1. The program first outputs the characters <SO>, <FF>, and <SI>. In this way <FF> is tested in the shift-out mode, and the printer is ready for output of the shift-in character set. (If any characters were stored in the printer buffer, they are printed before the form feed).

The program now prints:

SHIFT-IN CHAR. SET

and a vertical tabulation is performed so that <VT> is tested in the shift-in mode.

After this the program outputs the characters:

48 32 48 32 <CR> 32 49 32 49 <NL>

so that a line containing

0101

is printed, which is a test of <CR> and <SP> in the shift-in mode. Furthermore it is now possible to measure the length of the skip formed by the vertical tabulation mentioned above.

Then 119 lines are printed; this is one line for each character except for the 9 control characters so that a line is printed for each of the characters 0-8, 16-23, 25, 26, and 28-127. In each line the character is output 132 times followed by the number of the character (3 digits) followed by <NL>. So 60 lines each contains 132 equal graphic characters (including the line containing spaces), while the rest of the lines only contains the number of the (blind) character. In this way 1) <NL> and 2) the printing of only the first 132 characters are tested in the shift-in mode. Furthermore it is tested that 3) blind characters are treated correctly, i.e. the line printer's buffer pointer is not moved by a blind character, and a blind character does not destroy a non-blind character following it.

At last the program prints:

SI END

2. After having output the characters <SI> and <FF> (and in this way tested <FF> in the shift-in mode) the program prints:

SHIFT-OUT CHAR. SET

and it outputs the characters <SO> and <VT> so that vertical tabulation is tested in the shift-out mode.

The printer is now ready for printing the shift-out character set, and the program outputs the characters:

48 32 48 32 <CR> 32 49 32 49 <NL>

so that a line containing

0101

is printed, which is a test of <CR>, and <SP> in the shift-out mode. Furthermore it is now possible to measure the length of the skip performed by the vertical tabulation mentioned above.

Then 119 lines are printed; this is one line for each character except for the 9 control characters so that a line is printed for each of the characters 0-8, 16-23, 25, 26, and 28-127. In each line the character is output 132 times followed by the number of the character (3 digits) followed by <NL>. So 19 lines each contains 132 equal graphic characters (including the line containing spaces), while the rest of the lines only contains the number of the (blind) character. In this way 1) <NL> and 2) the printing of only the first 132 characters are tested in the shift-out mode. Furthermore it is tested that 3) blind characters are treated correctly, i.e. the line printer's buffer pointer is not moved by a blind character, and a blind character does not destroy a non-blind character following it.

At last the program prints:

SO END

3. Finally it is tested that in the same line it is possible to change between the shift-in and the shift-out character set. The program outputs the characters:

<SI> 65 <SI> <SI> 65 <SI> <SO> 37 <SO> <SO> 37 <SO> <SI> 66 <SI> <SI> 66 <NL>

causing the printing of:

AAooBB

After the last run <FF> is output.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

lineprinter disconnected

If EX = b01 for more than 0.5 sec after one of the control characters <NL>, <VT>, <FF>, <CR>, the message is:

char. = <char.>, lineprinter busy for 0.5 sec.

If EX = b01 for more than 60 microsec. after a character different from the 4 control characters mentioned above, the message is:

char. = <char.>, lineprinter busy for 60 microsec.

If no interrupt signal has been received when EX = b00 after one of the control characters <NL>, <VT>, <FF>, <CR>, the program writes:

char. = <char.>, no interrupt from lineprinter

```
SHIFT-IN CHAR. SET

0101
000
001
002
003
004
005
006
007
008
016
017
018
019
020
021
022
023
025
026
028
029
030
031

!!!!!!!!!!!!!!!!!!!!!!!!              .!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
""""""""""""""""""""""""              """"""""""""""""""""""""""""""""""

035
036
%%%%%%%%%%%%%%%%%%%                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
&&&&&&&&&&&&&&&&&&                  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''''''''''''''''''                  '''''''''''''''''''''''''''''''''''''
(((((((((((((((((                 .(((((((((((((((((((((((((((((((((((((((
))))))))))))))))                  )))))))))))))))))))))))))))))))))))))))
***************                 ****************************************
+++++++++++++++                 ++++++++++++++++++++++++++++++++++++++++
,,,,,,,,,,,,,,,,,                 ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
-----------------                  --------------------------------------

.................                   .....................................
///////////////////                 '/////////////////////////////////////
000000000000000000000                000000000000000000000000000000000000
111111111111111111111                 11111111111111111111111111111111111
2222222222222222222222                 2222222222222222222222222222222222
33333333333333333333333                 333333333333333333333333333333333
```

```
4444444444444444444444        444444444444444444444444444444
55555555555555555555555        55555555555555555555555555555
666666666666666666666666        6666666666666666666666666666
7777777777777777777777777       7777777777777777777777777777
8888888888888888888888888       88888888888888888888888888888
9999999999999999999999          999999999999999999999999999999
:::::::::::::::::::::           :::::::::::::::::::::::::::::::
;;;;;;;;;;;;;;;;;;;;          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
<<<<<<<<<<<<<<<<<<<<          <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
=====================         ================================
>>>>>>>>>>>>>>>>>>>>>          >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
??????????????????????          ??????????????????????????????
064
AAAAAAAAAAAAAAAAAAAAAAAA        AAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBB       BBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCC        CCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDD        DDDDDDDDDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEEEEEEEE       EEEEEEEEEEEEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFFFFFFFFF        FFFFFFFFFFFFFFFFFFFFFFFFFFFF
GGGGGGGGGGGGGGGGGGGGGGGG        GGGGGGGGGGGGGGGGGGGGGGGGGGGGG
HHHHHHHHHHHHHHHHHHHHHH          HHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
IIIIIIIIIIIIIIIIIIIII          IIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
JJJJJJJJJJJJJJJJJJJJ         JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKKKKK         KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
LLLLLLLLLLLLLLLLLL        LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMM         MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
NNNNNNNNNNNNNNNNNN         NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
OOOOOOOOOOOOOOOOOOO         OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
PPPPPPPPPPPPPPPPPPPP         PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
QQQQQQQQQQQQQQQQQQQQQ         QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
RRRRRRRRRRRRRRRRRRRRRR         RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SSSSSSSSSSSSSSSSSSSSSSS         SSSSSSSSSSSSSSSSSSSSSSSSSSSSS
TTTTTTTTTTTTTTTTTTTTTTTT        TTTTTTTTTTTTTTTTTTTTTTTTTTTT
UUUUUUUUUUUUUUUUUUUUUUUUU        UUUUUUUUUUUUUUUUUUUUUUUUUUU
VVVVVVVVVVVVVVVVVVVVVVVVVV        VVVVVVVVVVVVVVVVVVVVVVVVVVV
WWWWWWWWWWWWWWWWWWWWWWWWWWW        WWWWWWWWWWWWWWWWWWWWWWWWWW
XXXXXXXXXXXXXXXXXXXXXXXXXXXX        XXXXXXXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYYYYYYYYYYYYY        YYYYYYYYYYYYYYYYYYYYYYYY
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZ        ZZZZZZZZZZZZZZZZZZZZZZZZZ
ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ        ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ
ØØØØØØØØØØØØØØØØØØØØØØØØØ        ØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ         ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ
094
______________________          ________________________________
096
aaaaaaaaaaaaaaaaaaaaa          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbb          bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccc          cccccccccccccccccccccccccccccc
ddddddddddddddddddddddd          ddddddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeee         eeeeeeeeeeeeeeeeeeeeeeeeeeeee
ffffffffffffffffffffffff         fffffffffffffffffffffffffffff
gggggggggggggggggggggggg         ggggggggggggggggggggggggggggg
hhhhhhhhhhhhhhhhhhhhhhh          hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
iiiiiiiiiiiiiiiiiiiiiii          iiiiiiiiiiiiiiiiiiiiiiiiiiiiii
```

```
jjjjjjjjjjjjjjjjjjjjjj              jjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
kkkkkkkkkkkkkkkkkkkkkk             kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
lllllllllllllllllllll              lllllllllllllllllllllllllllllll
mmmmmmmmmmmmmmmmmmmmr              mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
nnnnnnnnnnnnnnnnnnnn               nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
ooooooooooooooooooooooo              ooooooooooooooooooooooooooooo
ppppppppppppppppppppppp              ppppppppppppppppppppppppppppp
qqqqqqqqqqqqqqqqqqqqqqqq                qqqqqqqqqqqqqqqqqqqqqqqqqq
rrrrrrrrrrrrrrrrrrrrrrrrrr               rrrrrrrrrrrrrrrrrrrrrrrrrr
sssssssssssssssssssssssssss                sssssssssssssssssssssss
ttttttttttttttttttttttttttt                ttttttttttttttttttttttt
uuuuuuuuuuuuuuuuuuuuuuuuuu                uuuuuuuuuuuuuuuuuuuuuuuu
vvvvvvvvvvvvvvvvvvvvvvvvvv               vvvvvvvvvvvvvvvvvvvvvvvvv
wwwwwwwwwwwwwwwwwwwwwwww                wwwwwwwwwwwwwwwwwwwwwwwwww
xxxxxxxxxxxxxxxxxxxxxxx                xxxxxxxxxxxxxxxxxxxxxxxxxxx
yyyyyyyyyyyyyyyyyyyyyy               yyyyyyyyyyyyyyyyyyyyyyyyyyyyy
zzzzzzzzzzzzzzzzzzzzzz              zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
æææææææææææææææææææææ             æææææææææææææææææææææææææææææææ
øøøøøøøøøøøøøøøøøøøø               øøøøøøøøøøøøøøøøøøøøøøøøøøøøøø
ååååååååååååååååååå              ååååååååååååååååååååååååååååååååå
126
127
SI END
```

```
SHIFT-OUT CHAR. SET

0101
000
001
002
003
004
005
006
007
008
016
017
018
019
020
021
022
023
025
026
028
```

VB436

```
029
030
031

!!!!!!!!!!!!!!!!!!!!       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
""""""""""""""""""""        """"""""""""""""""""""""""""""""""""""""
035
036
%%%%%%%%%%%%%%%%%%%%%%%       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
&&&&&&&&&&&&&&&&&&&&&&&&       &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''''''''''''''''''''''''        ''''''''''''''''''''''''''''''''''''
(((((((((((((((((((((((((        (((((((((((((((((((((((((((((((((((
))))))))))))))))))))))))))        ))))))))))))))))))))))))))))))))))
***************************        **********************************
+++++++++++++++++++++++++++       ++++++++++++++++++++++++++++++++++
,,,,,,,,,,,,,,,,,,,,,,,,,,       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
-------------------------        ------------------------------------
........................        ....................................
///////////////////////        /////////////////////////////////////
0000000000000000000000        00000000000000000000000000000000000000
1111111111111111111111       111111111111111111111111111111111111111
222222222222222222222       2222222222222222222222222222222222222222
33333333333333333333       33333333333333333333333333333333333333333
4444444444444444444        444444444444444444444444444444444444444444
555555555555555555        5555555555555555555555555555555555555555555
666666666666666666       66666666666666666666666666666666666666666666
77777777777777777        77777777777777777777777777777777777777777777
8888888888888888888       8888888888888888888888888888888888888888888
999999999999999999         999999999999999999999999999999999999999999
::::::::::::::::::::        ::::::::::::::::::::::::::::::::::::::::
;;;;;;;;;;;;;;;;;;;;;        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
<<<<<<<<<<<<<<<<<<<<<<       <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
=======================       ======================================
>>>>>>>>>>>>>>>>>>>>>>>>       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
?????????????????????????       ????????????????????????????????????
064
AAAAAAAAAAAAAAAAAAAAAAAAAAA        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBB         BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCC         CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDD        DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEEEEEEEEE       EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFFFFFFFFFF       FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
GGGGGGGGGGGGGGGGGGGGGGGG       GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
HHHHHHHHHHHHHHHHHHHHHHHH       HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
IIIIIIIIIIIIIIIIIIIIIIIII       IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
JJJJJJJJJJJJJJJJJJJJJJJJJJ       JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKKKKKKKKKKKK        KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
LLLLLLLLLLLLLLLLLLLLLLLLLLL        LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMMMMMMMMMMMMM       MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
NNNNNNNNNNNNNNNNNNNNNNNNNNNN        NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
OOOOOOOOOOOOOOOOOOOOOOOOOOOOO        OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPP        PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQ        QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
RRRRRRRRRRRRRRRRRRRRRRRRRRRRR       RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

VB436

```
SSSSSSSSSSSSSSSSSSSSSSSS      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
TTTTTTTTTTTTTTTTTTTTTTTT     TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
UUUUUUUUUUUUUUUUUUUUUU     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
VVVVVVVVVVVVVVVVVVVVV     .VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
WWWWWWWWWWWWWWWWWWWW     WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
XXXXXXXXXXXXXXXXXXX'    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYY      YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
ZZZZZZZZZZZZZZZZZZZ      ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ     `ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ
ØØØØØØØØØØØØØØØØØØØØØ      ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ      `ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ
094
------------------------      ------------------------------
096
aaaaaaaaaaaaaaaaaaaaaaaaa       aaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbb       bbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccc       cccccccccccccccccccccccccc
dddddddddddddddddddddddddddd      'dddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeeeeeeee      ,eeeeeeeeeeeeeeeeeeeeeeeee
ffffffffffffffffffffffffffff'      (ffffffffffffffffffffffffff
gggggggggggggggggggggggggg      .ggggggggggggggggggggggggggg
hhhhhhhhhhhhhhhhhhhhhhhhh      hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
iiiiiiiiiiiiiiiiiiiiiii      iiiiiiiiiiiiiiiiiiiiiiiiiiiiii
jjjjjjjjjjjjjjjjjjjjjj'     jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
kkkkkkkkkkkkkkkkkkkkk:     kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
lllllllllllllllllllll      llllllllllllllllllllllllllllllll
mmmmmmmmmmmmmmmmmmmmmmm     mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
nnnnnnnnnnnnnnnnnnnnnn      nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
oooooooooooooooooooooooo     ooooooooooooooooooooooooooooooo
ppppppppppppppppppppppp      pppppppppppppppppppppppppppppp
qqqqqqqqqqqqqqqqqqqqqqqq     qqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
rrrrrrrrrrrrrrrrrrrrrrrr      rrrrrrrrrrrrrrrrrrrrrrrrrrrrr
ssssssssssssssssssssssss     ,ssssssssssssssssssssssssssssss
tttttttttttttttttttttttt     tttttttttttttttttttttttttttttt
uuuuuuuuuuuuuuuuuuuuuuu     uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
vvvvvvvvvvvvvvvvvvvvvv      vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
wwwwwwwwwwwwwwwwwwwwww     wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
xxxxxxxxxxxxxxxxxxxxx      xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
yyyyyyyyyyyyyyyyyyyyy'    /yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
zzzzzzzzzzzzzzzzzzzzz     `zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
æææææææææææææææææææææææ.     æææææææææææææææææææææææææææææææ
øøøøøøøøøøøøøøøøøøøøøøøø      `øøøøøøøøøøøøøøøøøøøøøøøøøøøøøø
ååååååååååååååååååååååååå.      `ååååååååååååååååååååååååååååå
126
127
SO END
AA%%BB
```

VB436

1.3 HT, CAN, ESC

Purpose

This program tests the control characters <HT>, <CAN>, and <ESC> in the shift-in mode as well as in the shift-out mode. Furthermore the exception register, the interrupt signals, and the status word are examined.

The program occupies about 370 words.

Initiation

A format tape for this test is mounted on the printer.

Test

1. HT (character No. 9)

The program first outputs the characters:

<FF> <SI> H T <NL>

causing a form feed and the printing of

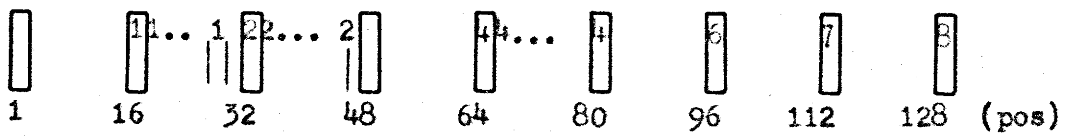
HT

(If any characters are situated in the printer buffer, they are printed before the form feed).

Then the program outputs:

$<HT>\ \underbrace{49...49}_{15\text{ times}}\ <HT>\ \underbrace{50...50}_{16\text{ times}}\ <HT>\ \underbrace{52...52}_{17\text{ times}}\ <HT>\ 54\ <HT>\ 55\ <HT>\ 56\ <HT>\ 57\ <NL>$

causing the printing of



Now the program outputs <SO> followed by the same sequence of characters as mentioned above, and the line is printed again. In this way <HT> is tested after 0, 1, 15, 16, and 17 characters, and <HT> is tested to the right of position No. 128.

2. CAN (character No. 24)

The program first outputs the characters:

```
                    <SI> <NL> C A N <NL>
```

causing the printing of

```
                              CAN
```

Then the program outputs:

```
                                   <CAN>  67  <NL>
              48...48  (132 times) <CAN>  65  <NL>
              48...48  (133 times) <CAN>  78  <NL>
        <SO>                       <CAN>  67  <NL>
        <SO>  48...48  (132 times) <CAN>  65  <NL>
        <SO>  48...48  (133 times) <CAN>  78  <NL>
```

causing the printing of

```
                          C
                          A
                          N
```

twice. In this way <CAN> is tested after 0, 132, and 133 characters, and it is checked that <CAN> involves a change to the shift-in mode.

3. ESC (character No. 27)

The program first outputs the characters:

```
              <SI>  <NL>  S  I  -  E  S  C  <NL>
```

causing the printing of

```
                    SI  -  ESC
```

Characters are now output according to this algorithm:

```
for m = 10 step 1 until 11 do
for n = 48 step 1 until 63 do
output (<CAN><nnn><SP><mmm><ESC> n m )
```

where

```
<nnn>::= <3 digits of the number n>
<mmm>::= <3 digits of the number m>
```

This causes the printing of:

```
048  010  and 049  010
050  010

051  010


052  010
     .
     .
     .
063  010
     .
     .          15 x <NL>
     .
048  011   followed by VT according to channel No. 0
049  011   followed by VT according to channel No. 1
050  011   followed by VT according to channel No. 2
     .
     .
063  011   followed by VT according to channel No. 15
```

The program then prints:

SO-ESC

and after <SO> the abovementioned algorithm is executed again so that <ESC> is tested in the shift-out mode too.

Each run is finished by performing a form feed.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 1 sec, the message is:

device busy for 1 sec.

If no interrupt signal has been received when EX = b00 after one of the characters <NL>, <VT>, <FF>, and <CR>, the program writes:

no interrupt from device

If bit No. 5 only is set in the status word, the message is:

end of tape

and the program waits until the operator types <NL> after having mounted the paper.

If a bit is set in one or more positions different from bit pos. No. 5, the program writes:

status = <bit 0-23 of the status word>

2.1 WRITE CHAR. FROM TAPE

Purpose

In each run this program outputs to the line printer an arbitrary sequence of characters. The program does not test the interrupt signals or the status word.

The program occupies about 160 words. Furthermore it uses a buffer area of 250 words.

Initiation

The wanted sequence of (not more than 500) characters must be punched in binary (i.e. the numbers 0-127 in an arbitrary parity may be used) on a piece of paper tape. (The program 2.1 for the paper tape punch may be used for this purpose).

When the program writes:

load tape

the paper tape is placed in the RC 2000 paper tape reader (dev. No. 0), and the operator types <NL>.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 1 sec, the message is:

device busy for 1 sec.

If bit No. 5 is set in the status word, the program writes:

end of paper

but the program does not wait. If there really is no paper in the line printer, the operator may mount this after having put the printer into the local state.

If it is not possible to reserve the output buffer within the available core store, this message is given:

buffer too big

RCSL: 51-VB437
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

RC 333 LINE PRINTER, ANELEX

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

1.2 WRITE (SI AND SO)

Purpose

This program tests the shift-in character set as well as the shift-out character set. Furthermore the exception register, the interrupt signals, and the status word are examined.

The program occupies about 370 words.

Initiation

A format tape containing holes for form feed (channel No. 8) and for vertical tabulation (channel No. 2) is mounted on the printer.

Test

The printer output from the test described below is shown on page 7-11.

1) The program first outputs the characters <SO>, <FF>, and <SI>. In this way <FF> is tested in the shift-out mode, and the printer is ready for output of the shift-in character set. (If any characters were stored in the printer buffer, they are printed before the form feed).

The program now prints:

SHIFT-IN CHAR. SET

and a vertical tabulation is performed so that <VT> is tested in the shift-in mode.

After this the program outputs the characters:

48 32 48 32 <CR> 32 49 32 49 <NL>

so that a line containing

0101

is printed, which is a test of <CR> and <SP> in the shift-in mode. Furthermore it is now possible to measure the length of the skip performed by the vertical tabulation mentioned above.

Then 122 lines are printed; this is one line for each character except for the 6 control characters so that a line is printed for each of the characters 0 to 9 and 16 to 127. In each line the character is output 160 times followed by the number of the character (3 digits) followed by <NL>. So 89 lines each contains 160 equal graphic characters (including the line containing spaces), while the rest of the lines only contains the number of the (blind) character. In this way 1) <NL> and 2) the printing of only the first 160 characters are tested in the shift-in mode. Furthermore it is tested that 3) blind characters are treated correctly, i.e. the line printer's buffer pointer is not moved by a blind character, and a blind character does not destroy a non-blind character following it.

At last the program prints:

```
SI END
```

2) After having output the characters <SI> and <FF> (and in this way tested <FF> in the shift-in mode) the program prints:

```
SHIFT-OUT CHAR. SET
```

and it outputs the characters <SO> and <VT> so that vertical tabulation is tested in the shift-out mode.

The printer is now ready for printing the shift-out character set, and the program outputs the characters:

48 32 48 32 <CR> 32 49 32 49 <NL>

so that a line containing

```
0101
```

is printed, which is a test of <CR> and <SP> in the shift-out mode. Furthermore it is now possible to measure the length of the skip performed by the vertical tabulation mentioned above.

Then 122 lines are printed; this is one line for each character except for the 6 control characters so that a line is printed for each of the characters 0 to 9 and 16 to 127. In each line the character is output 160 times followed by the number of the character (3 digits) followed by <NL>. So 44 lines each contains 160 equal graphic characters (including the line containing spaces), while the rest of the lines only contains the number of the (blind) character. In this way 1) <NL> and 2) the printing of only the first 160 characters are tested in the shift-out mode. Furthermore it is tested that 3) blind characters are treated correctly, i.e. the line printer's buffer pointer is not moved by a blind character, and a blind character does not destroy a non-blind character following it.

At last the program prints:

```
SO END
```

3) Finally it is tested that in the same line it is possible to change between the shift-in and the shift-out character set. The program outputs the characters:

<SI> 65 <SI> <SI> 65 <SI> <SO> 64 <SO> <SO> 64 <SO> <SI> 66 <SI> <SI> 66 <NL>

causing the printing of:

$\text{AA}_{\text{oo}}\text{BB}$

After the last run <FF> is output.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

line printer disconnected

If EX = b01 for more than 0.5 sec. after one of the control characters <NL>, <VT>, <FF>, <CR>, the message is:

<u>char. = <char.>, line printer busy for 0.5 sec.</u>

If EX = b01 for more than 60 microsec. after a character different from the 4 control characters mentioned above, the message is:

<u>char. = <char.>, line printer busy for 60 microsec.</u>

If no interrupt signal has been received when EX = b00 after one of the control characters <NL>, <VT>, <FF>, <CR>, the program writes:

<u>char. = <char.>, no interrupt from line printer</u>

If an interrupt signal has been received when EX = b00 after a character different from the 4 control characters mentioned above, the program writes:

<u>char. = <char.>, interrupt from line printer</u>

If bit No. 5 only is set in the status word, the message is:

<u>end of paper</u>

and the program waits until the operator types <NL> after having mounted paper.

If a bit is set in the status word in one or more positions different from No. 5, the program writes:

<u>status = <bit 0-23 of the status word></u>

```
SHIFT-IN CHAR. SET



0101
000
001
002
003
004
005
006
007
008
009
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
!!!!!!!!!!!!!!!!!!!!!!!!!.         !!!!!!!!!!!!!!!!!!!!!!!!
""""""""""""""""""""""""""          """""""""""""""""""""""
035
036
%%%%%%%%%%%%%%%%%%%%%%%%%%%%          %%%%%%%%%%%%%%%%%%%%%%%
&&&&&&&&&&&&&&&&&&&&&&&&&&         .&&&&&&&&&&&&&&&&&&&&&&&&
'''''''''''''''''''''''''          ''''''''''''''''''''''''
(((((((((((((((((((((((          ((((((((((((((((((((((((((
)))))))))))))))))))))))          ))))))))))))))))))))))))))
************************          '*************************
+++++++++++++++++++++++++          +++++++++++++++++++++++++
,,,,,,,,,,,,,,,,,,,,,,,,,          ,,,,,,,,,,,,,,,,,,,,,,,,
--------------------------         ------------------------
.......................          .........................
///////////////////////          //////////////////////////
0000000000000000000000        J00000000000000000000000000
111111111111111111111       1111111111111111111111111111
222222222222222222222      22222222222222222222222222222
3333333333333333333      333333333333333333333333333333
44444444444444444      4444444444444444444444444444444
```

VB437

```
5555555555555555'          5555555555555555555555555555555
666666666666666           666666666666666666666666666666666
77777777777777'          7777777777777777777777777777777777
8888888888888888.          `88888888888888888888888888888888
99999999999999999            `9999999999999999999999999999999
::::::::::::::::::::             `::::::::::::::::::::::::::::
;;;;;;;;;;;;;;;;;;;;;;,             `;;;;;;;;;;;;;;;;;;;;;;;;;;
<<<<<<<<<<<<<<<<<<<<<<<<.              <<<<<<<<<<<<<<<<<<<<<<<<<
==========================.              =======================
>>>>>>>>>>>>>>>>>>>>>>>>>                >>>>>>>>>>>>>>>>>>>>>>>
????????????????????????'              ????????????????????????
064
AAAAAAAAAAAAAAAAAAAAAA/              AAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBB.             BBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCC              'CCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDL               `DDDDDDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEEEEEEEL             `EEEEEEEEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFFFFFFFFFF              `FFFFFFFFFFFFFFFFFFFFFFF
GGGGGGGGGGGGGGGGGGGGGGGGr             GGGGGGGGGGGGGGGGGGGGGGGGG
HHHHHHHHHHHHHHHHHHHHHHHP            .HHHHHHHHHHHHHHHHHHHHHHHHHH
IIIIIIIIIIIIIIIIIIIII           .IIIIIIIIIIIIIIIIIIIIIIIIIII
JJJJJJJJJJJJJJJJJJJJ           ,JJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKKKKKK          .KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
LLLLLLLLLLLLLLLL'          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMi         MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
NNNNNNNNNNNNNNNNN.         NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
OOOOOOOOOOOOOOOOOL          OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
PPPPPPPPPPPPPPPPPPP          PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
QQQQQQQQQQQQQQQQQQL          QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
RRRRRRRRRRRRRRRRRRRRK           RRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SSSSSSSSSSSSSSSSSSSSSS            SSSSSSSSSSSSSSSSSSSSSSSSSSSS
TTTTTTTTTTTTTTTTTTTTTT,            TTTTTTTTTTTTTTTTTTTTTTTTTTT
UUUUUUUUUUUUUUUUUUUUUUUUL             UUUUUUUUUUUUUUUUUUUUUUUUU
VVVVVVVVVVVVVVVVVVVVVVVV              'VVVVVVVVVVVVVVVVVVVVVVVV
WWWWWWWWWWWWWWWWWWWWWWWW'              WWWWWWWWWWWWWWWWWWWWWWWW
XXXXXXXXXXXXXXXXXXXXXXXY              XXXXXXXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYYYYYYYY             YYYYYYYYYYYYYYYYYYYYYYYYYY
ZZZZZZZZZZZZZZZZZZZZZZZ'             ZZZZZZZZZZZZZZZZZZZZZZZZZZ
ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ             .ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ
ØØØØØØØØØØØØØØØØØØØØØØL              ØØØØØØØØØØØØØØØØØØØØØØØØØ
ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ.             ÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅÅ
094
095
096
aaaaaaaaaaaaaaaaaaaaaaaaa              .aaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbb             ,bbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccc            cccccccccccccccccccccccccc
ddddddddddddddddddddddd'           dddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeer          eeeeeeeeeeeeeeeeeeeeeeeeeeee
fffffffffffffffffffff'          ffffffffffffffffffffffffffff
ggggggggggggggggggggg'         gggggggggggggggggggggggggggggg
hhhhhhhhhhhhhhhhhhhh'          hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
iiiiiiiiiiiiiiiiii,           'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
jjjjjjjjjjjjjjjjjjj.          jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
```

VB437

```
kkkkkkkkkkkkkkkkkkkkk             kkkkkkkkkkkkkkkkkkkkkkkkkkk
lllllllllllllllllllll              llllllllllllllllllllllllll
mmmmmmmmmmmmmmmmmmmmmmm             mmmmmmmmmmmmmmmmmmmmmmmmm
nnnnnnnnnnnnnnnnnnnnnnnn             nnnnnnnnnnnnnnnnnnnnnnnn
ooooooooooooooooooooooooo             ooooooooooooooooooooooo
pppppppppppppppppppppppppp             pppppppppppppppppppppp
qqqqqqqqqqqqqqqqqqqqqqqqq             qqqqqqqqqqqqqqqqqqqqqqq
rrrrrrrrrrrrrrrrrrrrrrrr            rrrrrrrrrrrrrrrrrrrrrrrrr
sssssssssssssssssssss             ssssssssssssssssssssssssss
ttttttttttttttttttttttt             ttttttttttttttttttttttttt
uuuuuuuuuuuuuuuuuuuuuuuu             uuuuuuuuuuuuuuuuuuuuuuuu
vvvvvvvvvvvvvvvvvvvvvvvvv             vvvvvvvvvvvvvvvvvvvvvvv
wwwwwwwwwwwwwwwwwwwwwwwwwww             wwwwwwwwwwwwwwwwwwwwww
xxxxxxxxxxxxxxxxxxxxxxxxxxxx             xxxxxxxxxxxxxxxxxxxx
yyyyyyyyyyyyyyyyyyyyyyyyyyyyy             yyyyyyyyyyyyyyyyyyy
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz             zzzzzzzzzzzzzzzzzz
æææææææææææææææææææææææææææææææ             ææææææææææææææææ
øøøøøøøøøøøøøøøøøøøøøøøøøøøøøøø             øøøøøøøøøøøøøøøøø
åååååååååååååååååååååååååååååå             ååååååååååååååååå
126
127
SI END
```

```
SHIFT-OUT CHAR. SET



0101
000
001
002
003
004
005
006
007
008
009
016
017
018
019
020
021
022
023
024
025
```

```
026
027
028
029
030
031

033
034
035
[[[[[[[[[[[[[[[[[[[[[[[[[[[[[      [[[[[[[[[[[[[[[[[[[[[
037
]]]]]]]]]]]]]]]]]]]]]]]]]]      ]]]]]]]]]]]]]]]]]]]]]]]]
039
040
041
042
043
044
045
046
047
0000000000000000000000       000000000000000000000000000
111111111111111111111       1111111111111111111111111111
22222222222222222222       22222222222222222222222222222
3333333333333333333       333333333333333333333333333333
444444444444444444       4444444444444444444444444444444
55555555555555555       55555555555555555555555555555555
6666666666666666       666666666666666666666666666666666
7777777777777777      7777777777777777777777777777777777
88888888888888888       88888888888888888888888888888888
9999999999999999999       999999999999999999999999999999
::::::::::::::::::::        ::::::::::::::::::::::::::::
;;;;;;;;;;;;;;;;;;;;;;        ;;;;;;;;;;;;;;;;;;;;;;;;;;
<<<<<<<<<<<<<<<<<<<<<<<<        <<<<<<<<<<<<<<<<<<<<<<<<
==========================        ======================
>>>>>>>>>>>>>>>>>>>>>>>>>>        >>>>>>>>>>>>>>>>>>>>>>
?????????????????????????        ???????????????????????
00000000000000000000000        000000000000000000000000
1111111111111111111111        1111111111111111111111111
2222222222222222222222          22222222222222222222222
33333333333333333333333          3333333333333333333333
444444444444444444444444          444444444444444444444
5555555555555555555555555          55555555555555555555
66666666666666666666666666          6666666666666666666
7777777777777777777777777          777777777777777777777
88888888888888888888888          8888888888888888888888
999999999999999999999          999999999999999999999999
-------------------          --------------------------
||||||||||||||||||          ||||||||||||||||||||||||||||
10101010101010101010101010101010          101010101010101010101010101010101010101010101010101010101010
vvvvvvvvvvvvvv          vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxx          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
^^^^^^^^^^^^^^^^          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
ooooooooooooooooo                       oooooooooooooooooooooooooooo
081
2222222222222222222.                      22222222222222222222222222
33333333333333333333.                      3333333333333333333333333
084
085
086
087
088
089
\\\\\\\\\\\\\\\\\\\\\\\                          \\\\\\\\\\\\\\\\\\\\\\\
''''''''''''''''''''''                          '''''''''''''''''''''''
¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨                          ¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨
~~~~~~~~~~~~~~~~~~~~~~~~                          ~~~~~~~~~~~~~~~~~~~~~~
££££££££££££££££££££££££££                            £££££££££££££££££££
$$$$$$$$$$$$$$$$$$$$$$$$$$$                             $$$$$$$$$$$$$$$$$$
096
097
098
099
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
SO END
AA°°BB
```

2.1 WRITE CHAR. FROM TAPE

Purpose

In each run this program outputs to the line printer an arbitrary sequence of characters. The program does not test the interrupt signals or the status word.

The program occupies about 160 words. Furthermore it uses a buffer area of 250 words.

Initiation

The wanted sequence of (not more than 500) characters must be punched in binary (i.e. the numbers 0 to 127 in an arbitrary parity may be used) on a piece of paper tape. (The program 2.1 for the paper tape punch may be used for this purpose).

When the program writes:

load tape

the paper tape is placed in the RC 2000 paper tape reader (device No. 0), and the operator types <NL>.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 1 sec., the message is:

device busy for 1 sec.

If bit No. 5 is set in the status word, the program writes:

end of paper

but the program does not wait. If there is really no paper in the line printer, the operator may mount this after having put the printer into the local state.

If it is not possible to reserve the output buffer within the available core store, this message is given:

buffer too big

RCSL: 51-VB438
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

RC 4191 PLOTTER

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

RC 4191, PLOTTER

1.2 OCTAGON

Purpose

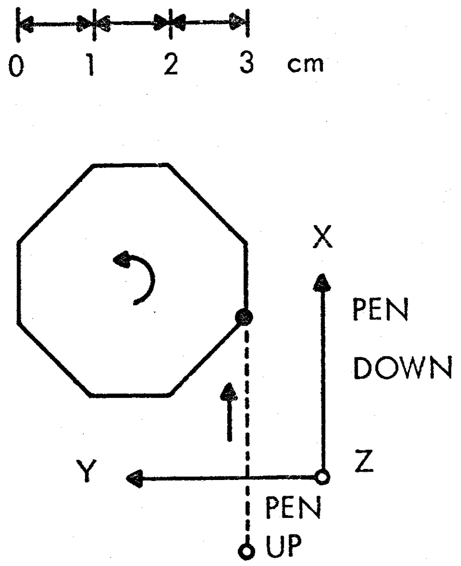
This program tests output of all characters, defined as well as undefined. Furthermore the exception register, the interrupt signals, and the status word are examined.

The program occupies about 220 words.

Test

First the program outputs all undefined characters; this must not move the pen.

After this the program writes all defined characters (shown on the next page) in a way causing the output of this figure:



	STEP	VARIANTS		
	ZXY	ZXY	ZXY	ZXY
	↑ 020	023	320	323
	↖ 022	322		
	← 002	032	302	332
	↙ 012	312		
	↓ 010	013	310	313
	↘ 011	311		
	→ 001	031	301	331
	↗ 021	321		
PEN UP	200	203	230	233
	220	223		
	222			
	202	232		
	212			
	210	213		
	211			
	201	231		
	221			
PEN DOWN	100	103	130	133
	120	123		
	122			
	102	132		
	112			
	110	113		
	111			
	101	131		
	121			
	26 characters	30 characters		

The following 8 characters do not move the pen:

000	003	030	300
	033	303	330
	333		

(Besides these 64 defined characters, 64 undefined characters exist, i.e. 128 7-bit characters).

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 1 sec., the message is:

device busy for 1 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If the status word does not contain the expected bitpattern (that is the output character), this message is given:

received: <received status, bit 0-23>

expected: <received status, bit 0-23>

In all cases the test continues after the message.

2.1 SEQUENCE

Purpose

In each run this program outputs in an uncritical way an arbitrary sequence of characters. The program tests neither the interrupt signals nor the status word.

The program occupies about 140 words.

Initiation

When the program writes:

sequence =

the operator inputs in binary the wanted sequence of (not more than 16) characters separated by <comma> and terminated by <NL>.

Next the program asks:

number of cycles =

and the operator types the number of times each character in the sequence has to be output.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.2 sec., the message is:

device busy for 0.2 sec.

RCSL: 51-VB439
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

RC 707 MAGNETIC TAPE STATION

7 TRACKS

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

RC 707 MAGNETIC TAPE STATION, 7 TRACKS

1.1 BITPATTERN

Purpose

This program writes and reads a block consisting of characters forming an irregular bitpattern. The transmitted data are checked and furthermore the exception register, the interrupt signals, and the status word are tested.

The program occupies about 580 words. In addition an input- outputbuffer is used the length of which is determined by the operator.

Initiation

First the program asks:

blocklength =

and the operator inputs the blocklength measured in words.

Then the program writes:

select density

and after having selected the density (by activating the high-low density push-button) the operator types <NL> and the program answers:

<density> density

Test

In each run the program writes the block in odd parity, backspaces, clears the input buffer, reads the block, and checks that each word is transferred correctly.

The block consists of a number of character sets, each set composed of characters as described in chapter 1.6; (i.e. the rightmost 6 bits of each of the 126 characters plus a parity bit). In this way each of the tracks including the parity track varies in an irregular way.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec. (except for rewind, see below), the message is:

device busy for 4 sec

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If EOT is sensed, the program writes:

end of tape

and the tape is rewound. In this case the tape station may be busy for more than 4 sec. without an error message.

If the status word contains an unexpected bitpattern, the message is:

<mode> status:

received <bit 0-23, received>

expected <bit 0-23, expected>

where

<mode>::= write | read

If a wrong number of characters is written or read, the program writes:

<mode>, no. of char.:

received <received No.>

expected <expected No.>

In case of data error, this message is given:

data-error, word no. <word No.>

received <bit 0-23, received>

expected <bit 0-23, expected>

If it is not possible to reserve an input-output buffer of the wanted size within the available core store, the program writes:

blocklength too big

If a ring is not sensed during the initiation, this message is written:

mount ring

and the program waits until the operator types <NL> after having mounted a ring.

1.2 WRITE AND READ

Purpose

This program tests output on and input from magnetic tape. Blocks of different length, contents and parity are written and read, so that tapemarks, block-length errors and parity errors are tested. Furthermore the status word, the exception register, and the interrupt signals are examined.

The different states of a run has been given a number which is printed below in the left margin.

It is noticed that just before each input instruction, the input buffer is filled with zeroes.

The program occupies about 760 words. Furthermore it uses a buffer of 600 words.

Initiation

A tape with ring is mounted. When the program writes:

select density

the operator selects the density (by activating the high-low density push-button) and types <NL>, after which the program rewinds the tape, senses and writes:

<density> density

where

<density>::= low | high

If a ring is not sensed, the program writes:

mount ring

and waits until the operator has mounted the ring and typed <NL>.

Next the program asks:

contents =

and the operator types (in decimal) an integer used by test II.

<u>Test I</u>

1. First the program writes 3 blocks in odd, even and odd parity respec-
2. tively (state 1, 2, and 3). The block lengths are 1 word (containing
3. the characters 1,2,3,4), 2 words (containing the characters 5,6,7,8, and 9,10,11,12) and 3 words (containing the characters 13,14,15,16, and 17,18,19,20, and 21,22,23,24) respectively.

4. Next the 3 blocks are read (state 4, state 5, and state 6) in even,
5. odd and even parity respectively into a buffer 2 words long. It is
6. tested that

   a. parity error is detected in each block.

   b. the correct number of characters is counted (i.e. 4, 8 and 8 characters respectively).

   c. block length error occurs in state 6 only.

   d. the characters are correctly transferred.

<u>Test II</u>

7. The tape is rewound and 1 block is written with odd parity (state 7). This block consists of 400 words containing the integers:

   the decimal number typed in by the operator plus
   0,1,2,...,398,399, respectively.

8. This block is read 3 times (state 8, state 9, and state 11) into a
9. buffer 400,200, and 600 words long, respectively. Having read the block the second time and having detected block length error, an
10. erase instruction is performed (state 10), which must not destroy
11. the rest of the block. It is tested that the number of input characters is 1600,800, and 1600, respectively.

Test III

12. The tape is backspaced, and the program now writes in even parity a block consisting of 1 word containing the characters 3,2,1,0 (state 12). It is tested that parity error occurs, because it is attempted to output the character 0 in even parity.

13. Next the block is read (state 13), and it is tested that the number of characters is 3.

Test IV

14.
15. The tape is backspaced, and the program writes 2 blocks (state 14 and state 15) in odd and even parity, respectively. Each block consists of 1 word, the first containing the characters 15,15,15,15, the second containing the characters 15,15,0,0 or in other words two tape marks. It is tested that tape mark is sensed and that 4 and 2 characters are written.

16.
17. Finally the two blocks are read (state 16 and state 17), and it is tested that tape mark is sensed and that the number of characters is 4 and 2, respectively.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec, the message is:

device busy for 4 sec

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

After each output operation (especially erase) and input operation (but not after movements, which are tested in program 1.3), the status word is examined; if the contents do not correspond to the expected contents, this message is written:

state no. <state No.>, status:

received: <bit 0-23, received>

expected: <bit 0-23, expected>

If the number of characters transferred is not equal to the expected number of characters, the program writes:

state no. <state No.>, no. of char:

received: <received number>

expected: <expected number>

After an input operation each word is compared with the corresponding output word, and if they are not equal, this message is written:

state no. <state No.>, word no. <word No.>

received: <bit 0-23, received>

expected: <bit 0-23, expected>

It is noticed that before an input operation the input buffer is filled with zeroes.

If a parity error occurs in state 1, 2, 3 or 7, the program writes:

state no. <state No.>, parity error

The block is backspaced, an erase-instruction is performed and the block is written again.

The program uses a buffer of 600 words. If it is not possible to reserve this buffer within the availbale core store, the program writes:

buffer too big

1.3 MOVE TAPE

Purpose

This program tests movements of the tape; at each movement the status word, the exception register, and the interrupt signal are tested.

The different states of a run have been given a number which is printed below in the left margin.

It is noticed that just before an input instruction, the input buffer is filled with zeroes.

The program occupies about 550 words. Furthermore it uses a buffer of 500 words.

Initiation

A tape with ring is mounted. When the program writes:

select density

the operator selects the density (by activating the high-low density pushbutton) and types <NL> after which the program rewinds the tape
0. (state No. 0), senses and writes:

<density> density

where

<density>::= low|high

If a ring is not sensed, the program writes:

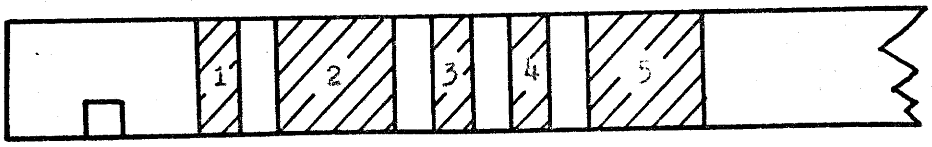
mount ring

and waits until the operator has mounted the ring and types <NL>.

The program now writes 5 blocks:

BLOCK NO.	NO. OF WORDS	WORD CONTENTS	PARITY
1	1	1, 1, 1, 1	odd
2	400	2, 2, 2, 2	even
3	2	3, 3, 3, 3	odd
4	1	15,15,15,15	even
5	500	5, 5, 5, 5	even

It is noticed that block No. 4 is a tapemark.



Test

In a run the following movements are performed:

1. Rewind.
2. Backspace 1 block.
3. Upspace 1 block, read 1 block and check that this was block No. 2.
4. Upspace 1 file, check that tape mark is sensed, read 1 block and check that this was block No. 5.
5. Backspace 1 file and check that tape mark is sensed.
6. Backspace 1 block, read 1 block and check that this was block No. 3.
7. Upspace 1 block and check that tape mark is sensed.
8. Backspace 1 block and check that tape mark is sensed.
9. Backspace 1 file.

When all runs are executed, an unload instruction is performed by which the magnetic tape station is set in the local state.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

```
                        exception reg. = b11
```

If EX = b10, the program writes:

```
                        device disconnected
```

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If a parity error occurs when one of the five blocks are written (when the program is initiated), the program writes:

parity error in block no. <block No.>

The block is backspaced, an erase instruction is performed and the block is written again.

After each move-operation the status word is examined; if the contents do not correspond to the expected contents, this message is written:

state no. <state No.> status error

received <bit 0-23, received>

expected <bit 0-23, expected>

If the tape is not correctly moved, the program writes:

state no. <state No.> move error

In all cases the test continues after the message.

The program uses a buffer of 500 words. If it is not possible to reserve this buffer within the available core store, the program writes:

buffer too big

2.1 READ, WRITE, MOVE

Purpose

An arbitrary sequence of output on, input from and movement of a magnetic tape may be performed by means of this program.

The program occupies about 340 words. Furthermore it uses a buffer, the length of which depends of the specified block length (see below).

Initiation

A tape is mounted, provided with a ring if necessary. When the program writes:

block length =

the operator types the length (number of words) of a block. Next the program asks:

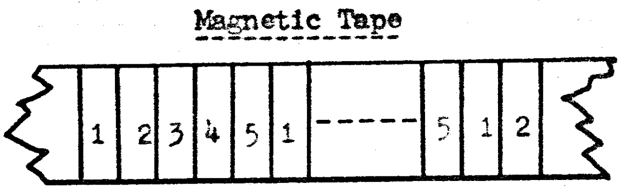
characters =

and the operator types (in decimal) 1-16 integers, separated by <comma> and finished by <NL>. These integers (modulu 64) are filled into the output buffer as shown below (where block length = 3, characters = 1,2,3,4,5).

Output Buffer

word 1	1	2	3	4
word 2	5	1	2	3
word 3	4	5	1	2

Magnetic Tape



After this the program writes:

parity =

and the operator selects the output-input parity by typing o or e, after which the program continues to write dd and ven, respectively.

Finally the program asks:

sequence =

and the operator specifies the wanted tape operations by typing 1-5 characters followed by <NL>. The characters must be selected among those shown below:

i	input	(read 1 block)
o	output	(write 1 block)
t	write tape mark	
e	erase	
0		upspace 1 file
1		upspace 1 block
2	move	backspace 1 file
3		backspace 1 block
4		rewind
5		unload

Test

In each run the specified tape operations are performed in an uncritical way, that is without testing interrupt and status, and without testing the transferred data.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If EOT is sensed, the program writes:

end of tape

and the tape is rewound. In this case the tape station may be busy for more than 4 sec. without message.

The program uses a buffer the length of which is 2 * the specified block length (1 output buffer and 1 input buffer; the latter is filled with zeroes before an input instruction). If it is not possible to reserve the buffer within the available core store, the program writes:

block length too big

RCSL: 51-VB440
Author: Jørgen Lindballe
Edited: November 1969

RC 4000
TEST OF PERIPHERAL DEVICES

RC 709 MAGNETIC TAPE STATION
9 TRACKS

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS

RC 709 MAGNETIC TAPE STATION, 9 TRACKS

1.1. BITPATTERN

Purpose

This program writes and reads a block consisting of characters forming an irregular bitpattern. The transmitted data are checked and furthermore the exception register, the interrupt signals, and the status word are tested.

The program occupies about 720 words. In addition, an input-output buffer is used, the length of which is determined by the operator.

Initiation

First the program asks:

blocklength =

and the operator inputs the blocklength measured in number of words.

Then the program writes:

select density

and after having selected the density (by activating the high-low density push-button), the operator types <NL>, and the program answers:

<density> density

Test

In each run the program writes the block in odd parity, backspaces, clears the input buffer, reads the block, and checks that each word is transferred correctly.

The block consists of a number of character sets, each set composed of characters as described in chapter 1.6; (i.e. the rightmost 8 bits of each of the 522 characters + a parity bit). In this way each of the tracks including the parity track varies in an irregular way.

<u>Error Messages</u>

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

<u>exception reg. = b11</u>

If EX = b10, the program writes:

<u>device disconnected</u>

If EX = b01 for more than 4 sec. (except for rewind, see below), the message is:

<u>device busy for 4 sec</u>

If no interrupt signal has been received when EX = b00, the program writes:

<u>no interrupt from device</u>

If EOT is sensed, the program writes:

<u>end of tape</u>

and the tape is rewound. In this case the tape station may be busy for more than 4 sec. without an error message.

If the status word contains an unexpected bitpattern, the message is:

<u><mode> status:</u>

<u>received <bit 0-23, received></u>

<u>expected <bit 0-23, expected></u>

where

<mode>::= write | read

If a wrong number of characters is written or read, the program writes:

<u><mode>, no. of char.:</u>

<u>received <received No.></u>

<u>expected <expected No.></u>

In case of data error, this message is given:

data error, word no. <word No.>

received <bit 0-23, received>

expected <bit 0-23, expected>

If it is not possible to reserve an input-output buffer of the wanted size within the available core store, the program writes:

blocklength too big

If a ring is not sensed during the initiation, this message is written:

mount ring

and the program waits until the operator types <NL> after having mounted a ring.

1.2. WRITE AND READ

Purpose

This program tests output on and input from magnetic tape. Blocks of different length, contents and parity are written and read, so that tape marks, block length errors, and parity errors are tested. Furthermore the status word, the exception register, and the interrupt signals are examined.

The different states of a run has been given a number which is printed below in the left margin.

It is noticed that just before each input instruction, the input buffer is filled with zeroes.

The program occupies about 760 words. Furthermore it uses a buffer of 600 words.

Initiation

A tape with ring is mounted. When the program writes:

select density

the operator selects the density (by activating the high-low density push-button) and types <NL>, after which the program rewinds the tape, senses and writes:

<density> density

where

<density>::= low | high

If a ring is not sensed, the program writes:

mount ring

and waits until the operator has mounted the ring and types <NL>.

Next the program asks:

contents =

and the operator types (in decimal) an integer used by test II.

Test I

1. First the program writes 3 blocks in odd, even and odd parity respec-
2. tively (state 1,2, and 3). The block lengths are 1 word (containing
3. the characters 1,2,3), 2 words (containing the characters 4,5,6 and
7,8,9) and 3 words (containing the characters 10,11,12 and 13,14,15
and 16,17,18) respectively.

4. Next the 3 blocks are read (state 4, state 5, and state 6) in even,
5. odd and even parity respectively into a buffer 2 words long. It is
6. tested that

a. parity error is detected in each block.

b. the correct number of characters is counted (i.e. 3,6, and 6 characters respectively).

c. block length error occurs in state 6 only.

d. the characters are correctly transferred.

Test II

7. The tape is rewound and 1 block is written with odd parity (state 7).
This block consists of 400 words containing the integers:

the decimal number typed in by the operator plus
0,1,2,...,398,399, respectively.

8. This block is read 3 times (state 8, state 9, and state 11) into a
9. buffer 400,200, and 600 words long, respectively. Having read the
block the second time and having detected block length error, an
10. erase instruction is performed (state 10), which must not destroy
11. the rest of the block. It is tested that the number of input charac-
ters is 1200, 600, and 1200, respectively.

Test III

12. The tape is rewound, and the program now writes in even parity a block consisting of 1 word containing the characters 2,1,0 (state 12). It is tested that parity error occurs, because it is attempted to output the character 0 in even parity.

13. Next the block is read (state 13), and it is tested that the number of characters is 2, and that parity error occurs.

Test IV

14. The tape is rewound, and the program writes a tape mark (state 14). It is tested that tape mark is sensed.

15. Finally the tape mark is read (state 15), and it is tested that tape mark is sensed and that the number of characters is 1.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

```
exception reg. = b11
```

If EX = b10, the program writes:

```
device disconnected
```

If EX = b01 for more than 4 sec, the message is:

```
device busy for 4 sec
```

If no interrupt signal has been received when EX = b00, the program writes:

```
no interrupt from device
```

After each output operation (especially erase) and input operation (but not after movements which are tested in program 1.3), the status word is examined; if the contents do not correspond to the expected contents, this message is written:

```
state no. <state no.>, status:
received: <bit 0-23, received>
expected: <bit 0-23>, expected>
```

If the number of characters transferred is not equal to the expected number of characters, the program writes:

```
state no. <state no.>, no. of char.:
received: <received number>
expected: <expected number>
```

After an input operation each word is compared with the corresponding output word, and if they are not equal, this message is written:

```
state no. <state no.>, word no. <word no.>
received: <bit 0-23, received>
expected: <bit 0-23, expected>
```

It is noticed that before an input operation, the input buffer is filled with zeroes.

If a parity error occurs in state 1,2,3 or 7, the program writes:

```
state no. <state no.>, parity error
```

The block is backspaced, an erase instruction is performed and the block is written again.

The program uses a buffer of 600 words. If it is not possible to reserve this buffer within the available core store, the program writes:

```
buffer too big
```

1.3. MOVE TAPE

Purpose

This program tests movements of the tape; at each movement the status word, the exception register, and the interrupt signal are tested.

The different states of a run have been given a number which is printed below in the left margin.

It is noticed that just before an input instruction, the input buffer is filled with zeroes.

The program occupies about 550 words. Furthermore is uses a buffer of 500 words.

Initiation

A tape with ring is mounted. When the program writes:

select density

the operator selects the density (by activating the high-low density push-button) and types <NL> after which the program rewinds the tape

0. (state No. 0), senses and writes:

<density> density

where

<density>::= low | high

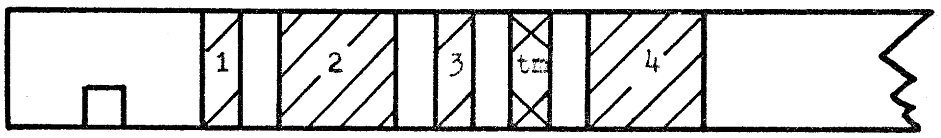
If a ring is not sensed, the program writes:

mount ring

and waits until the operator has mounted the ring and types <NL>.

The program now writes:

BLOCK NO.	NO. OF WORDS	WORD CONTENTS	PARITY
1	1	1, 1, 1	odd
2	400	2, 2, 2	even
3	2	3, 3, 3	odd
tapemark	-	100, 0, 0	even
4	500	4, 4, 4	even



Test

In a run the following movements are performed:

1. Rewind.

2. Backspace 1 block.

3. Upspace 1 block, read 1 block and check that this was block No. 2.

4. Upspace 1 file, check that tape mark is sensed, read 1 block and check that this was block No. 4.

5. Backspace 1 file and check that tape mark is sensed.

6. Backspace 1 block, read 1 block and check that this was block No. 3.

7. Upspace 1 block and check that tape mark is sensed.

8. Backspace 1 block and check that tape mark is sensed.

9. Backspace 1 file.

When all runs are executed, an unload instruction is performed by which the magnetic tape station is set in the local state.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If a parity error occurs when one of the five blocks are written (when the program is initiated), the program writes:

parity error in block no. <block no.>

The block is backspaced, an erase instruction is performed and the block is written again.

After each move operation the status word is examined; if the contents do not correspond to the expected contents, this message is written:

state no. <state no.> status error

received <bit 0-23, received>

expected <bit 0-23, expected>

If the tape is not correctly moved, the program writes:

state no. <state no.> move-error

In all cases the test continues after the message.

The program uses a buffer of 500 words. If it is not possible to reserve this buffer within the available core store, the program writes:

buffer too big

1.4. CRC - TEST

Purpose

This program tests the CRC-circuit by writing a number of blocks, and then by reading each block two times after the removal of a read-amplifier, examining that parity error occurs by the first read operation only.

The transmitted data are checked, and furthermore the exception register, and the status word are tested.

The program occupies about 850 words. In addition, an input-output buffer is used, the length of which is determined by the operator.

Initiation

First the program asks:

no. of blocks =

after which the operator types the number of blocks written and read in each run.

The program then writes:

blocklength =

and the operator inputs the blocklength measured in number of words.

Then the program writes:

select density

and after having selected the density (by activating the high-low density push-button) the operator types <NL> and the program answers:

<density> density

Finally the program asks:

error messages wanted:

and the operator types y or n after which the program continues to write es and o respectively.

Test

In each run the program first writes the wanted number of blocks in even parity. Each block consists of a number of character sets composed of characters as described in chapter 1.6 for the testprogram 1.1 (i.e. the rightmost 8 bits of each of the 522 characters plus a parity bit). In this way each of the tracks including the parity track varies in an irregular way.

Second the tape is rewound, and the program writes:

remove one read-amplifier

and the operator types <NL> after having removed one of the 9 read-amplifiers.

Third each block is read two times:

After having cleared the input buffer a block is read first time and it is checked that parity error occurs.

Then the block is backspaced and after having cleared the input buffer, it is read the second time, and it is checked that no parity error occurs, and that the data are read correctly.

When all blocks are read, the tape is rewound, and the program writes:

mount read-amplifier

After the last run the program writes this table containing the number of errors detected during the test:

no. of errors after:	write	read 1	read 2
status:	<No.>	<No.>	<No.>
data :	<No.>	-	<No.>

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec. (except for rewind, see below), the message is:

device busy for 4 sec.

If EOT is sensed, the program writes:

end of tape

and the tape is rewound. In this case the tape station may be busy for more than 4 sec. without an error message.

If parity error occurs during the write operation, the program writes:

parity error

the block is backspaced, an erase operation is performed and the block is written again.

If the status word contains an unexpected bitpattern after the write or read operation, the message is:

status error after <operation>

received <bit 0-23, received>

expected <bit 0-23, expected>

where

<operation>::= write | read 1 | read 2

In case of data error this message is given:

data error after <read operation>

word no. <word No.>

received <bit 0-23, received>

expected <bit 0-23, expected>

where

<read operation>::= write | read 2

If it is not possible to reserve an input-output buffer of the wanted size within the available core store, the program writes:

block length too big

If a ring is not sensed during the initiation, this message is written:

mount ring

and the program waits until the operator types <NL> after having mounted the ring.

2.1. READ, WRITE, MOVE

Purpose

An arbitrary sequence of output on, input from and movement of a magnetic tape may be performed by means of this program.

The program occupies about 420 words. Furthermore it uses a buffer, the length of which depends on the specified block length (see below).

Initiation

A tape is mounted, provided with a ring if necessary. When the program writes:

blocklength =

the operator types the length (number of words) of a block. Next the program asks:

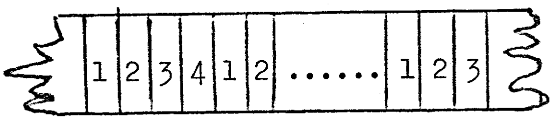
characters =

and the operator types (in decimal) 1-16 integers, separated by <comma> and finished by <NL>. These integers (modulu 256) are filled into the output buffer as shown below (where blocklength = 5, characters = 1,2,3,4).
Af

Output Buffer

word 1	1	2	3
word 2	4	1	2
word 3	3	4	1
word 4	2	3	4
word 5	1	2	3

Magnetic Tape



After this the program writes:

parity =

and the operator selects the output-input parity by typing o or e, after which the program continues to write dd and ven respectively.

Then the program calculates the cyclic redundance check character and writes:

crcc = <9-bit crcc>

Finally the program asks:

sequence =

and the operator specifies the wanted tape operations by typing 1-5 characters followed by <NL>. The characters must be selected among those shown below:

i	input	(read 1 block)
o	output	(write 1 block)
t	write tape mark	
e	erase	
l	sense block length	
0		upspace 1 file
1		upspace 1 block
2	move	backspace 1 file
3		backspace 1 block
4		rewind
5		unload

Test

In each run the specified tape operations are performed in an uncritical way, that is without testing interrupt and status, and without testing the transferred data.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 4 sec., the message is:

device busy for 4 sec.

If EOT is sensed, the program writes:

end of tape

and the tape is rewound. In this case the tape station may be busy for more than 4 sec. without message.

The program uses a buffer the length of which is 2 * the specified block length (1 output buffer and 1 input buffer; the latter is filled with zeroes before an input instruction). If it is not possible to reserve the buffer within the available core store, the program writes:

block length too big

RCSL: 51-VB441
Author: Jørgen Lindballe
Edited: July 1969

RC 4000
TEST OF PERIPHERAL DEVICES

RC 4415 DRUM

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

RC 4415 DRUM

1.1 TEST ALL SEGMENTS

Purpose

This program tests output on and input from one or more modules of the drum. Furthermore the exception register, the interrupt signals, and the status word are examined.

This program delivers error messages on segment level. For error messages on word level use test program 1.2.

The program occupies about 670 words.

Initiation

First the program asks:

first module =

and the operator inputs the number ($1 \leq$ number ≤ 8) of the first drum module he wants to test. The program then writes:

number of modules =

and the operator types in the number of modules (within 4096 words the buffer contains about 5 modules; the core store utilization is shown on page 5).

Finally the typewriter writes:

contents =

after which the operator types an integer (in decimal). In each drum word tested is written this integer plus the word number (word No. 0 is the first drum word of the test). In this way it is checked that the output addressing equals the input addressing.

Test

First the abovementioned contents are written in all the drum modules which are tested. Second all these modules are read. In this way it is tested that all the modules are used (i.e. no folding occurs).

The output is performed by transferring 3 segments at a time. The transfer time is:

$$\frac{1.06}{50}\ \frac{\text{sec.}}{\text{rev.}} \times \frac{7}{4}\ \frac{\text{rev.}}{\text{transmission}} \times \frac{256}{3}\ \frac{\text{transmissions}}{\text{module}}$$

or

$$\underline{3.2 \text{ sec. per module}}$$

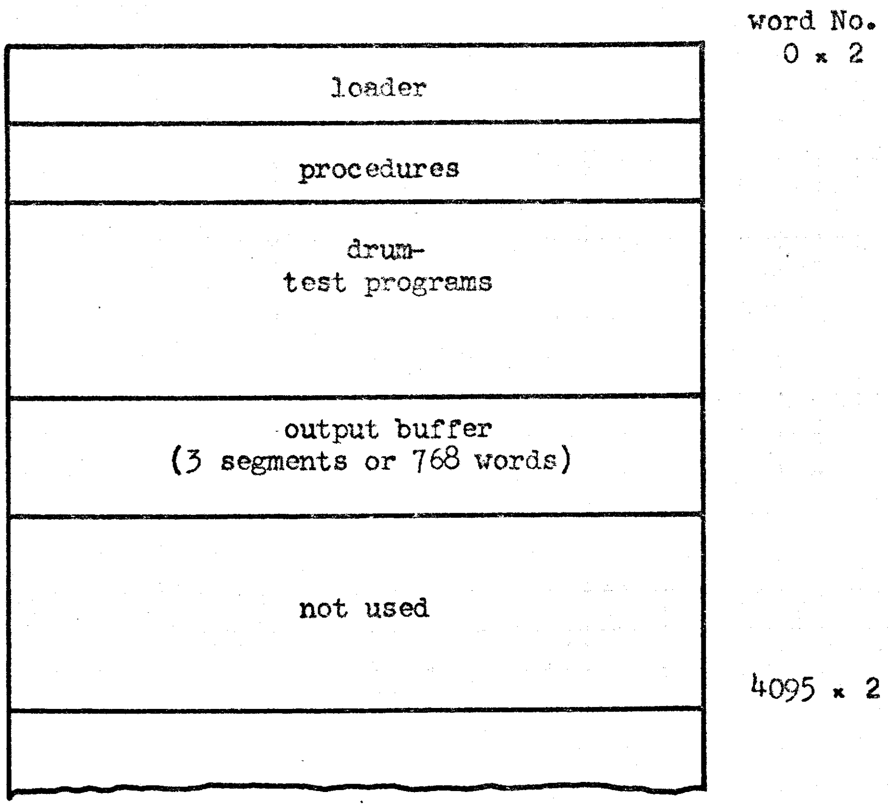
The input is performed by transferring 1 segment at a time. The transfer time is:

$$\frac{1.06}{50}\ \frac{\text{sec.}}{\text{rev.}} \times \frac{5}{4}\ \frac{\text{rev.}}{\text{transmission}} \times 256\ \frac{\text{transmissions}}{\text{module}}$$

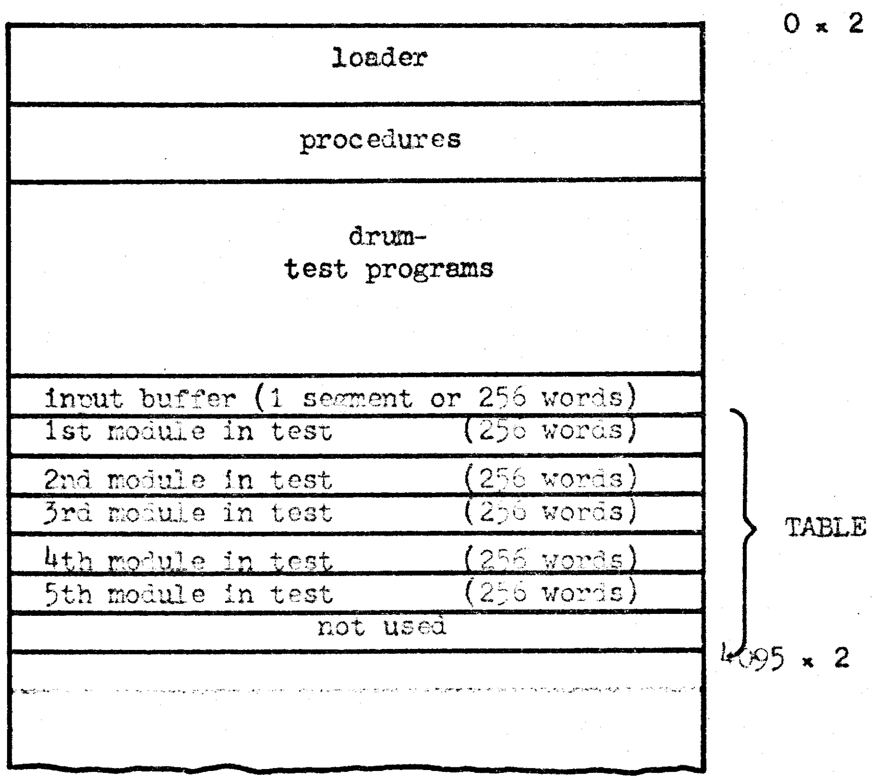
or

$$\underline{6.8 \text{ sec. per module}}$$

CORE STORE UTILIZATION



output on the drum


0 × 2
loader
procedures
drum-
test programs
input buffer (1 segment or 256 words)
1st module in test (256 words)
2nd module in test (256 words)
3rd module in test (256 words)
4th module in test (256 words)
5th module in test (256 words)
not used
TABLE
4095 × 2

input from the drum

Error Messages

If one or more errors are detected during the input from the drum, a scheme (an example of this is shown on the next page) is delivered on the typewriter. For every segment the program distinguishes between status error (that is bit 1, 2, or/and 3 is/are set in the status word) and data error (that is the received contents of one or more words of the segment do not equal the expected (output-) contents.

In case of status error in a segment the value of bit 1, 2, and 3 that is a digit: 1, 2,..., or 7 is printed; in case of data error in a segment the digit (which is 0 if no status error occurs) is underlined. If in a segment neither status error nor data error occurs, a <minus> is printed. For example:

STATUS ERROR	DATA ERROR	OUTPUT
no	no	-
no	yes	$\underline{0}$
yes	no	4 (parity error)
yes	yes	$\underline{4}$

If the status word is not equal to 0 after the output of 3 segments, this message is given:

segment no. <1.segm. no.>, <2.segm. no.>, <3.segm. no.>

output-status = <status bit 0-23>

If one or more bits different from bit No. 1, 2, and 3 are set it the status word after the input of a segment, the program writes:

segment no. <segment no.>

input-status = <status bit 0-23>

	SEGMENT	0- 3	4- 7	8-11	12-15	16-19	20-23	24-27	28-31	32-35	36-39	40-43	44-47	48-51	52-55	56-59	60-63
A2	0- 63	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
A3	64- 127	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
A4	128- 191	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
A5	192- 255	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
B1	256- 319	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
B2	320- 383	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
B3	384- 447	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
B4	448- 511	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
C2	512- 575	4444	4444														
C3	576- 639																
C4	640- 703																
C5	704- 767																
D1	768- 831																
D2	832- 895																
D3	896- 959																
D4	960-1023																
E2	1024-1087																
E3	1088-1151																
E4	1152-1215																
E5	1216-1279																
F1	1280-1343																
F2	1344-1407																
F3	1408-1471																
F4	1472-1535																
G2	1536-1599																
G3	1600-1663																
G4	1664-1727																
G5	1728-1791																
H1	1792-1855																
H2	1856-1919																
H3	1920-1983																
H4	1984-2047																

VB441

After the sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.1 sec., the message is:

device busy for 0.1 sec.

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If it is not possible to reserve the necessary table area for the specified number of modules, this message is given:

too many modules

1.2 TEST SOME SEGMENTS

Purpose

This program tests output on and input from one or more segments of the drum. Furthermore the exception register, the interrupt signals, and the status word are examined.

This program delivers error messages on word level. For error messages on segment level use test program 1.1.

The program occupies about 420 words.

Initiation

First the program asks:

first segment =

and the operator inputs the number of the first drum segment he wants to use for output and input. The program then writes:

number of segments =

and the operator types in the number of segments (within 4096 words the buffer contains about 6 segments).

When the program asks:

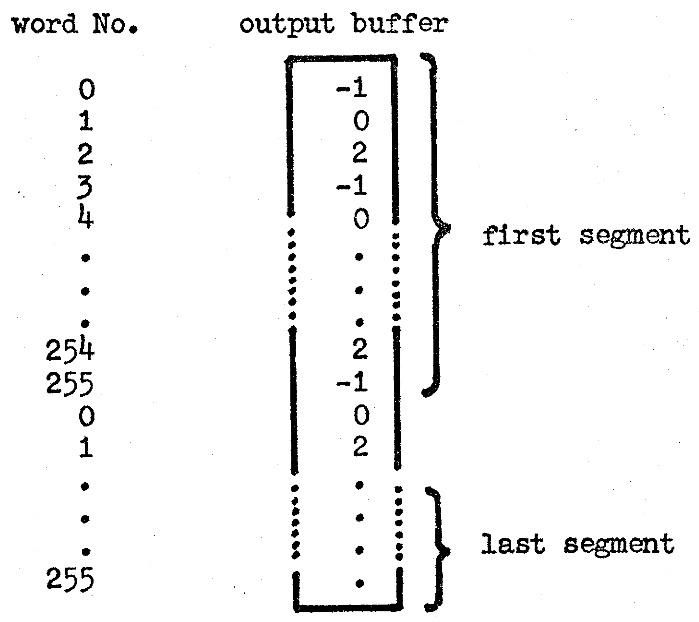
```
check from segment no.
               word no.
```

the operator writes the number of the 1st segment and the number of the first word within this segment from which examination for data error must take place.

Finally the program writes:

contents =

and the operator types (in decimal) from 1 up to a maximum of 16 integers separated by <comma> and terminated by <NL>. These integers are filled into the output buffer as shown below (in this example the integers are: -1, 0, 2):



Error Messages

In each run the program examines the input words specified for examination. In case of data error the program writes:

segment no. <segment no.>, word no. <word no.>

received <received contents, bits 0-23>

expected <expected contents, bits 0-23>

If the status word is different from 0 after output or input, the message is:

<mode> status = <status word bits 0-23>

where

<mode>::= write | read

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for a time greater than:

50 + 5 × (No. of segments)msec

the message is:

device busy

If no interrupt signal has been received when EX = b00, the program writes:

no interrupt from device

If the operator specifies a number of segments which exceed the available buffer area, the program writes:

too many segments

2.1 READ SEGMENTS

Purpose

This program performs input from one or more segments in an uncritical way, that is without examining the status word or the interrupt signals.

The program occupies about 150 words.

Initiation

First the program asks:

first segment =

and the operator inputs the number of the first segment from which input is wanted, and then the program asks:

number of segments =

and the operator writes the number of segments. Within 4096 words a buffer area for about 6 segments is available.

If wanted the segments may be filled with a well-defined contents by means of the program 2.2.

Error Messages

After a sense instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for a time greater than:

50 + 5 × (No. of segments)msec

the message is:

device busy

If the operator specifies a number of segments which exceeds the available buffer area, the program writes:

too many segments

2.2 WRITE SEGMENTS

Purpose

This program performs output on some segments in an uncritical way, that is without examining the status word or the interrupt signals.

The program occupies about 200 words.

Initiation

First the program asks:

first segment =

and the operator writes the number of the first segment he wants to write on. Then the program writes:

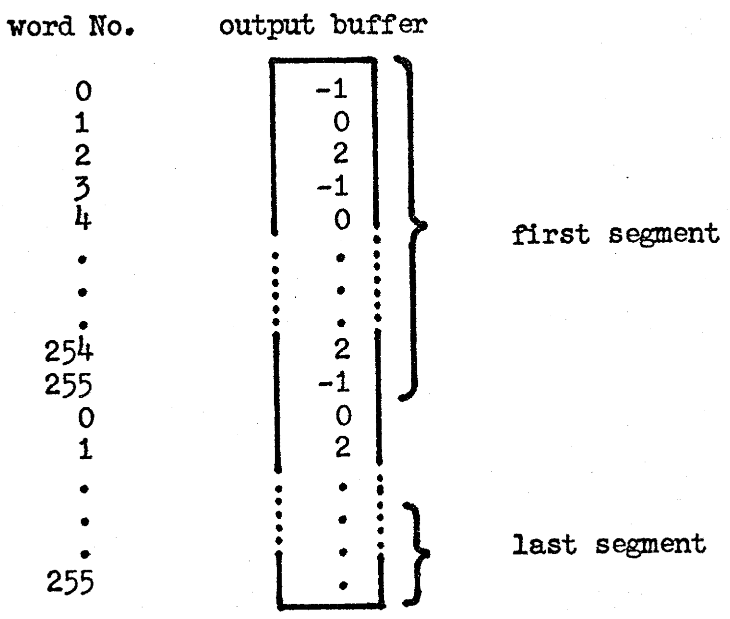
number of segments =

and the operator inputs the number of segments. Within 4096 words a buffer area for about 6 segments is available.

Finally the program asks:

contents =

and the operator types (in decimal) from 1 up to a maximum of 16 integers separated by <comma> and terminated by <NL>. These integers are filled into the output buffer as shown below (in this example the integers are: -1, 0, 2):



Error Messages

After a sense instruction the exception register is examined.

If EX = b11 this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for a time greater than:

$$50 + 5 \times (\text{No. of segments})\text{msec}$$

the message is:

device busy

If the operator specifies a number of segments which exceeds the available buffer area, the program writes:

too many segments

RCSL: 51-VB444
Author: Jørgen Lindballe
Edited: July 1969

RC 4000

TEST OF PERIPHERAL DEVICES

TELETYPEWRITER

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

1.2 READ (ECHO)

Purpose

This program tests input from the teletypewriter. Furthermore the exception register, the IO-interrupt signals, the IO-interrupt register, and the status word are examined.

The program occupies about 230 words.

Initiation

It is noticed that, when testing the teletypewriter, the operator must input 3 device numbers (separated by <comma> and terminated by <NL>), i.e. for the IO-interrupt register, for the CON-interrupt register, and for the terminal, and he must input 2 interrupt channel numbers (separated by <comma> and terminated by <NL>), i.e. for interrupt signals from the IO-interrupt register and from the CON-interrupt register.

Test

Each time the operator inputs a character, the program outputs this character followed by <CR> and <LF>.

After an input- or output operation the exception register, the interrupt signal, and the IO-interrupt register are tested, and after the input operation also the status word is examined.

Error Messages

After a sense instruction to the terminal the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

terminal disconnected

If EX = b01 for more than 8 sec, the message is:

terminal busy for 8 sec

If no IO-interrupt signal has been received when EX = b00, the program writes:

no io-interrupt signal

If the IO-interrupt register is not correctly set, the program writes:

io-int.reg., received: <bit 0-23, received>

io-int.reg., expected: <bit 0-23, expected>

If a bit is set in one or more of the positions 0-1 and 3-16 in the status word of the terminal after an input operation, this message is given:

read-status: <bit 0-23 of the status word>

In all cases the program continues after the message.

1.3 WRITE KEYBOARD

Purpose

This program tests the output of all characters, defined as well as undefined on the teletypewriter. Furthermore the exception register, the IO-interrupt signal, the IO-interrupt register, and the status word are examined.

The program occupies about 350 words.

Initiation

It is noticed that, when testing the teletypewriter, the operator must input 3 device numbers (separated by <comma> and terminated by <NL>), i.e. for the IO-interrupt register, for the CON-interrupt register, and for the terminal, and he must input 2 interrupt channel numbers (separated by <comma> and terminated by <NL>), i.e. for interrupt signals from the IO-interrupt register and from the CON-interrupt register.

Test

First the program writes all defined characters in 'upper case' (followed by <BEL>) and in 'lower case'. The layout shown below corresponds to the keyboard.

Second the program outputs all undefined characters which does not cause any printing.

After each character the program checks that the status word is 0.

```
! @ # $ % ` & * ( ) _ + ‾
Q W E R T Y U I O P ~ [
A S D F G H J K L : " {
Z X C V B N M < > ?
<BEL>
```

```
1 2 3 4 5 6 7 8 9 0 - = |
q w e r t y u i o p ^ ]
a s d f g h j k l ; ' }
z x c v b n m , . /
```

Error Messages

After a sense instruction to the terminal the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

terminal disconnected

If EX = b01 for more than 8 sec, the message is:

terminal busy for 8 sec

If no IO-interrupt signal has been received when EX = b00, the program writes:

no io-interrupt signal

If the IO-interrupt register is not correctly set, the program writes:

io-int.reg., received: <bit 0-23, received>

io-int.reg., expected: <bit 0-23, expected>

If the device buffer of the terminal contains a bit pattern different from 0 after an output operation, this message is given:

device buffer = <bit 0-23>

In all cases the program continues after the message.

1.4 TIMER

Purpose

This program tests that the timer acts correctly.

The program occupies about 190 words.

Initiation

It is noticed that, when testing the teletypewriter, the operator must input 3 device numbers (separated by <comma> and terminated by <NL>), i.e. for the IO-interrupt register, for the CON-interrupt register, and for the terminal, and he must input 2 interrupt channel numbers (separated by <comma> and terminated by <NL>), i.e. for interrupt signals from the IO-interrupt register and from the CON-interrupt register.

Before the first run the program writes:

```
time, expected: <lower limit>-<upper limit>msec
time, measured:
```

where lower and upper limit are the time limits, i.e. 3000 and 6000 msec, respectively.

Test

In each run the program initiates an input operation and measures the time until an interrupt signal is delivered and timer status is set. Then the program writes:

```
<measured time>msec
```

Error Messages

After a sense instruction to the terminal the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

terminal disconnected

If EX = b01 for more than 8 sec, the message is:

terminal busy for 8 sec

If no IO-interrupt signal has been received when EX = b00, the program writes:

no io-interrupt signal

If other bits than bit No. 2 is set in the status word of the terminal, this message is given:

status = <bit 0-23 of the status word>

In all cases the program continues after the message.

2.1 SEQUENCE

Purpose

In each run this program writes in an uncritical way an arbitrary sequence of characters followed by <LF> and <CR>. The program tests neither the interrupt signals, the IO-interrupt register nor the status word.

The program occupies about 180 words.

Initiation

It is noticed that, when testing the teletypewriter, the operator must input 3 device numbers (separated by <comma> and terminated by <NL>), i.e. for the IO-interrupt register, for the CON-interrupt register, and for the terminal, and he must input 2 interrupt channel numbers (separated by <comma> and terminated by <NL>), i.e. for interrupt signals from the IO-interrupt register and from the CON-interrupt register.

When the program writes:

sequence =

the operator inputs the wanted sequence of (not more than 100) characters terminated by <NL>. The characters (exclusive of < (iso character No. 60), <NL> (iso character No. 10), <SP> (iso character No. 32), and all characters undefined on the operator's typewriter) may be input directly or they may be input by writing the decimal iso character No. enclosed in < and > ; for example

sequence = <60><0>>

causes the printing of < > .

Error Messages

After a sense instruction to the terminal the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

terminal disconnected

If EX = b01 for more than 8 sec, the message is:

terminal busy for 8 sec

In all cases the program continues after the message.

RCSL: 51-VB445
Author: Jørgen Lindballe
Edited: July 1969

RC 4000
TEST OF PERIPHERAL DEVICES

DPC 401 DISPLAY

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

1.2 TEST SEQUENCE

Purpose

In each run this program outputs an arbitrary sequence of characters.

The program occupies about 260 words.

Test

In each run the program first writes:

sequence =

and the operator inputs a sequence of (not more than 100) display-characters terminated by <NL>. A character may be input in one of three ways:

1. The character (different from <NL> <SP> c w b d /) is input directly.
2. b followed by a binary integer.
3. d followed by a decimal integer.

A set of control-characters must be preceded by c, and a set of write-characters must be preceded by w, for example:

sequence = wHd10cb0000010wE

A slash deletes the last display-character, 2 slashes delete the 2 last display-characters etc. Initially the sequence is set in the write mode.

Error Messages

After a control- or write-instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.05 sec., the message is:

device busy for 0.05 sec.

2.1 OUTPUT SEQUENCE

Purpose

In each run this program outputs an arbitrary sequence of characters.

The program occupies about 260 words.

Initiation

Thr program writes:

sequence =

and the operator inputs a sequence of (not more than 100) display-characters terminated by <NL>. A character may be input in one of three ways:

1. The character (different from <NL> <SP> c w b d /) is input directly.
2. b followed by a binary integer.
3. d followed by a decimal integer.

A set of control-characters must be preceded by c, and a set of write-characters must be preceded by w, for example:

sequence = wHd10cb0000010wE

A slash deletes the last display-character, 2 slashes delete the 2 last display-characters etc. Initially the sequence is set in the write mode.

Error Messages

After a control- or write-instruction the exception register is examined.

If EX = b11, this message is given:

exception reg. = b11

If EX = b10, the program writes:

device disconnected

If EX = b01 for more than 0.05 sec., the message is:

device busy for 0.05 sec.

RCSL: 51-VB620
Author: Allan Giese
Edited: November 1969

RC 4000

CORE STORE TEST

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

CONTENTS:

1. INITIATION OF PROGRAMS

The core store test program is loaded by means of either the fixed or the relocatable loader* in exactly the same way as the testprograms for the peripheral devices.

When the program writes:

device no. =

and later channel no. =

the operator should respond by typing a <NL> character. He may also type a number terminated by <NL>, but the number has no influence at all on the core store programs.

When the program asks:

error messages:

the operator types y or n for yes and no, respectively, whereafter the program supplies the text with the missing characters es (yes) or o (no).

The next message informs the operator about the free areas of the core store (i.e. the areas not occupied by the program) by writing:

free areas: <area specification>

where <area specification>::= <k number>-<k number> |
<k number>-<k number>,<k number>-<k number>

<k number>::= <positive number>k<positive number>

As an example, the k number 2k15 denotes word number 2 × 1024 + 15 = 2063. The corresponding byte numbers are, of course, 4126 and 4127.

The program writes now:

test areas: <area specification>

and the operator can type the area specification he wants within the stated free areas. The specification is terminated by a <NL> character. If the test areas are outside the free areas, the program will once again write test areas and this is repeated until agreement has been achieved.

Remember to switch off the core store parity control, otherwise the program is stopped whenever a parity error occurs.

2. ADDRESS SELECTION TEST

Purpose

This program tests that every word in the test areas of the core store can be selected. In other words, the test guarantees that two different addresses will not select the same word.

Test

The program works by loading into each word the number of the word, i.e. core store (n):= n. When this is accomplished for every word, the program checks whether the core store contents are correct or not.

Error Messages

An error causes the program to issue the following message:

```
address   <k number>
received  <received data, 24 bits>
expected  <expected data, 24 bits>
```

3. BIT SELECTION TEST

Purpose

This program tests that every bit in the test areas of the core store can be set and reset.

Test

The program works by loading into each word the following bitpatterns consisting of 27 bits:

```
000 ... 000 001
000 ... 000 010

100 ... 000 000
000 ... 000 000
```

When a bitpattern is stored in core store, it is read out again to be checked against the original bitpattern.

Error Messages

An error causes the program to issue the following message:

```
address   <k number>
received  <received data, 27 bits>
expected  <expected data, 27 bits>
```

4. WORST CASE TEST

Purpose

This program checks that the contents of the test areas of the core store are not changed due to noise introduced by the worst case pattern.

Test

The program starts by loading into each word the worst case word, belonging to the worst case pattern; this is done only in run No. 1. Then it continues to read those words repeatedly.

The worst case pattern is an array of worst case words and the contents of those words are chosen so that maximum resultant noise will be produced at the output of the sense windings. The contents of the worst case word are either all ones or all zeroes depending on the address bits 15 (core address 128), 16 (core address 64), and 21 (core address 2) as seen below:

Contents:= all ones	for	odd Address (15,16,21) = 0
Contents:= all zeroes	for	odd Address (15,16,21) = 1

Error Messages

An error causes the program to issue the following message:

address <k number>

received <received data, 27 bits>

expected <0 or 1>

where <0 or 1>::= 0|1

0 stands for all zeroes and 1 for all ones.

5. WORST CASE COMPLEMENT TEST

Purpose

This program checks that the contents of the test areas of the core store are not changed due to noise introduced by the worst case complement pattern.

Test

The program works as the worst case test program, the only exception being that the contents of the words under test are complemented, i.e.

```
Contents:= all ones     for     odd Address (15,16,21) = 1
Contents:= all zeroes   for     odd Address (15,16,21) = 0
```

Error Messages

The error messages are identical to those of the worst case test.

RCSL: 51-VB335
Author: Allan Giese
Edited: February 1969

PROGRAM DESCRIPTION FOR CLEARING THE CORE STORE FOR PARITY ERRORS

ABSTRACT.

The purpose of this program is to cancel the stop of the micro-program caused by a parity error and to re-enter the NORMAL operation mode.

A/S REGNECENTRALEN
Falkoneralle 1
2000 Copenhagen F

The operator must carry out the following procedure:

PAPER TAPE READER:

1. Insert the program in the paper tape reader and press the RESET button

CENTRAL PROCESSOR, TECHNICAL PANEL:

2. Switch the MODE SELECTOR key to TECHNICAL position
3. Deactivate CORE STORE CONTROL
4. Press the button named: MAR MANUAL CONTROLLED and insert in MAR the address x4y2
5. Press the button named: MAR COMPUTER CONTROLLED
6. Press the button named: CONTINUE
7. Press the button named: MAR MANUAL CONTROLLED and insert in MAR the address x20y4
8. Press the button named: MAR COMPUTER CONTROLLED
9. Press the button named: CONTINUE

   When the typewriter writes:

   switch to NORMAL position

   then

10. Switch the MODE SELECTOR key to NORMAL position

INPUT/OUTPUT TYPEWRITER:

11. Press the New Line (NL) key on the typewriter

    When the typewriter writes:

    core store cleared for parity error

    then the computer is ready for normal program execution.

PROGRAM EXPLANATION.

When the program issues its first message, the program has stored in each word of the core store its own word number; - the only exception being the words occupied by the program itself. After the NL key has been depressed, the program will once more activate all words of the store, but now under supervision of the parity control circuit.

Register w1 equals the address of the core store word to be tested.

```
s. a100, b100,

;   procedure Standard heading for autoload key;
                                    ; begin
w.      aw            2             ;   comment         w0: aw            2
        aw            4             ;                   w1: aw            4
        jl            0             ;                   w2: jl            0
        aw            94            ;                   w1: aw            94
        aw        x1  2             ;                  a94: aw       x1   2
        aw            96            ;                   w1: aw            96
        al   w1   x1  2             ;                  a96: al   w1  x1   2
        aw            98            ;                   w1: aw            98
        jl.          -4             ;                  a98: jl.          -4
        jl   w1       94            ;                   w1: jl   w1       94
                                    ;   w1:= 4;
                                    ; end Standard heading for autoload key;

                      0             ;   w3:= 0;
                      0             ;   ST[8]:= 0;
                      0             ;   ST[10]:= 0;
                      24            ;   Service addr:= 24;
                                    ; Start location 1:
                                    ;   w1:= 94;
 a14:   al   w2      -1             ;
        pl            5             ;   PR:= b 1111 1111;
 a18:   al   w1   x1  2             ;   for w1:= 96 step 2 until store limit
        rs   w1   x1  0             ;     do ST[w3]:= w3;
        jl.          -4             ;


;   procedure Type text 1;
                                    ; begin
 a24: al. w2          a42.
 a26: bz  w0   x2     0             ;   char:= w0;
      sn  w0          0             ;   if char = 0
      jl.             a70.          ;     then goto Wait for NL;
 a32: io  w0          2<6+3         ;   io( w0, IO tpw, write);
      sx  w1          3             ;   if busy ∨ disconnected
      jl.            -4             ;     then goto a32;
      al  w2   x2     1             ;   pointer:= pointer + 1;
      jl.             a26.


;  Data for text 1;
h.a42:           10 , 115           ;   comment
                 119, 105           ;   < switch to NORMAL position>
                 116, 99
                 104, 32
                 116, 111
                 32 , 78
                 79 , 82
                 77 , 65
                 76 , 32
                 112, 111
                 115, 105
                 116, 105
                 111, 110
                 0  , 0             ;    the endmark is the character 0;
                                    ; end Type text 1;
```

```
                                        ; Wait for NL:
w.a70: io  w0       2<6+2               ;   io( w0, IO tpw, read);
  a72: io  w0       2<6+0               ;   io( w0, IO tpw, sense);
       sx           3                   ;   if busy ∨ disconnected
       jl.         -4                   ;     then goto a72;
       se  w0       10                  ;   if char < > NL
       jl.          a70.                ;    then goto a70;
       al  w1       6                   ;   w1:= 6;
       al  w1  x1   2                   ;   for w1:= 8 step 2 until 72
       aw      x1   0                   ;     do ST[w1]:= 4 characters;
       se  w1       72                  ;
       jl.         -6
       jl           14                  ;   goto Start location 2;
  a94: jl.          a14.                ;   goto Start location 1;
; a96:
; a98:

                    0                   ;   ST[8]:= 0;
                    0                   ;   ST[10]:= 0;
                    20                  ;   Service addr:= 20;
                                        ; Start location 2:
  b14: al  w1  x1   2                   ;   for w1:= 74 step 2 until store limit;
       rs  w1  x1   0                   ;     do ST[w1]:= w1;
       jl.         -4                   ;


;   procedure Type text 2;
                                        ; begin
  b20: al. w2       b38.
  b22: bz  w0  x2   0                   ;   char:= w0;
       sn  w0       0                   ;   if char = 0
  b26: jl.          0                   ;     then goto b26;
  b28: io  w0       2<6+3               ;   io( w0, IO tpw, write);
       sx           3                   ;   if busy ∨ disconnected
       jl.         -4                   ;     then goto b28;
       al  w2  x2   1                   ;   pointer:= pointer + 1;
       jl.          b22.                ;

;  Data for text 2;
h.b38:            99 , 111              ;   comment
                  114, 101              ;   < core store cleared for
                  32 , 115              ;    parity error>
                  116, 111
                  114, 101
                  32 , 99
                  108, 101
                  97 , 114
                  101, 100
                  32 , 102
                  111, 114
                  32 , 112
                  97 , 114
                  105, 116
                  121, 32
                  101, 114
                  114, 111
  b72:            114, 0                ;  the endmark is the character 0;
                                        ; end Type text 2;
e.
```

VB335

A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

HEADQUARTERS: FALKONER ALLÉ 1 . DK-2000 COPENHAGEN F . DENMARK
TELEPHONE: (01) 105366 . TELEX: 6282 RCHQ DK . CABLES: REGNECENTRALEN