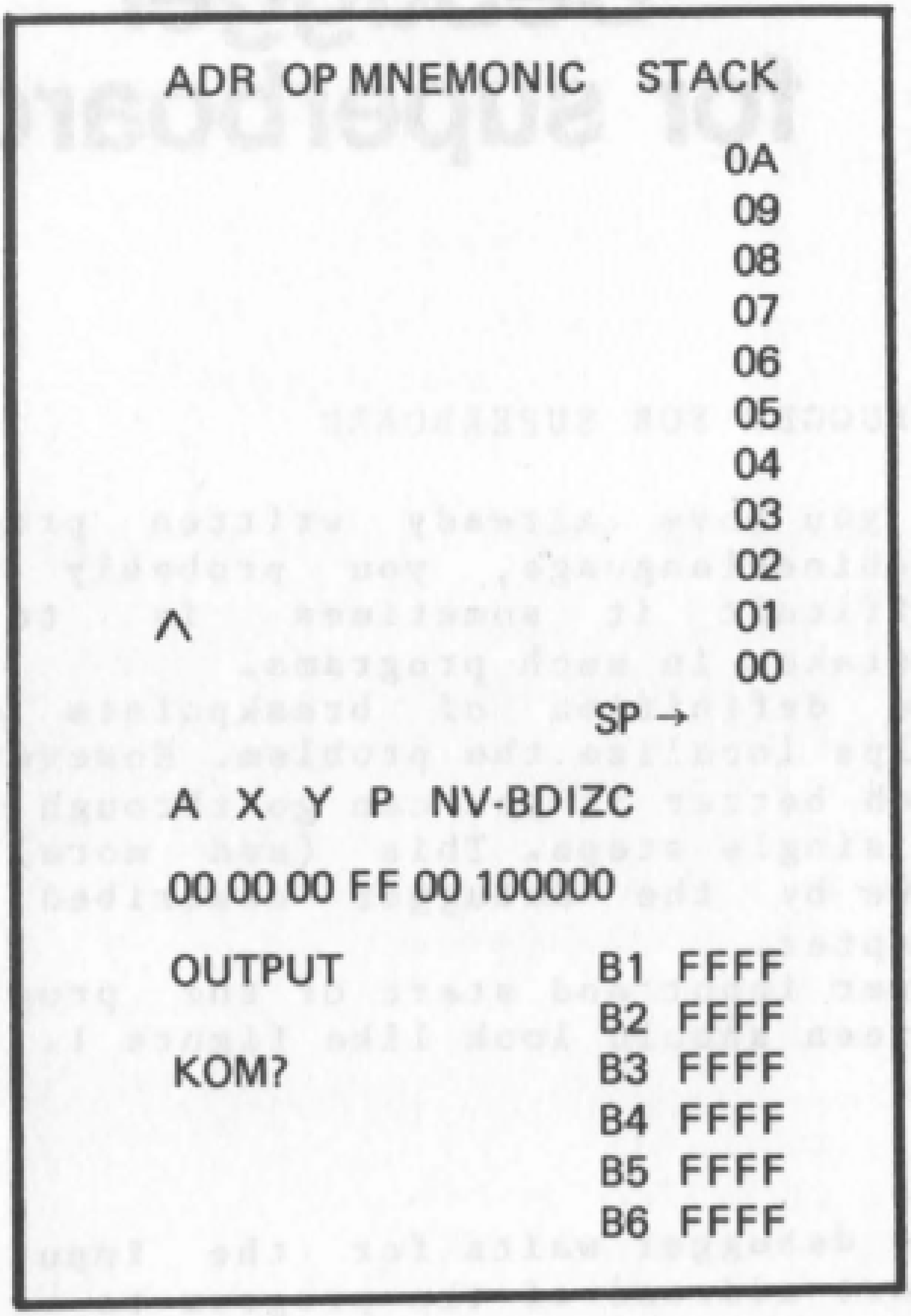
Debugger for superboard

DEBUGGER FOR SUPERBOARD

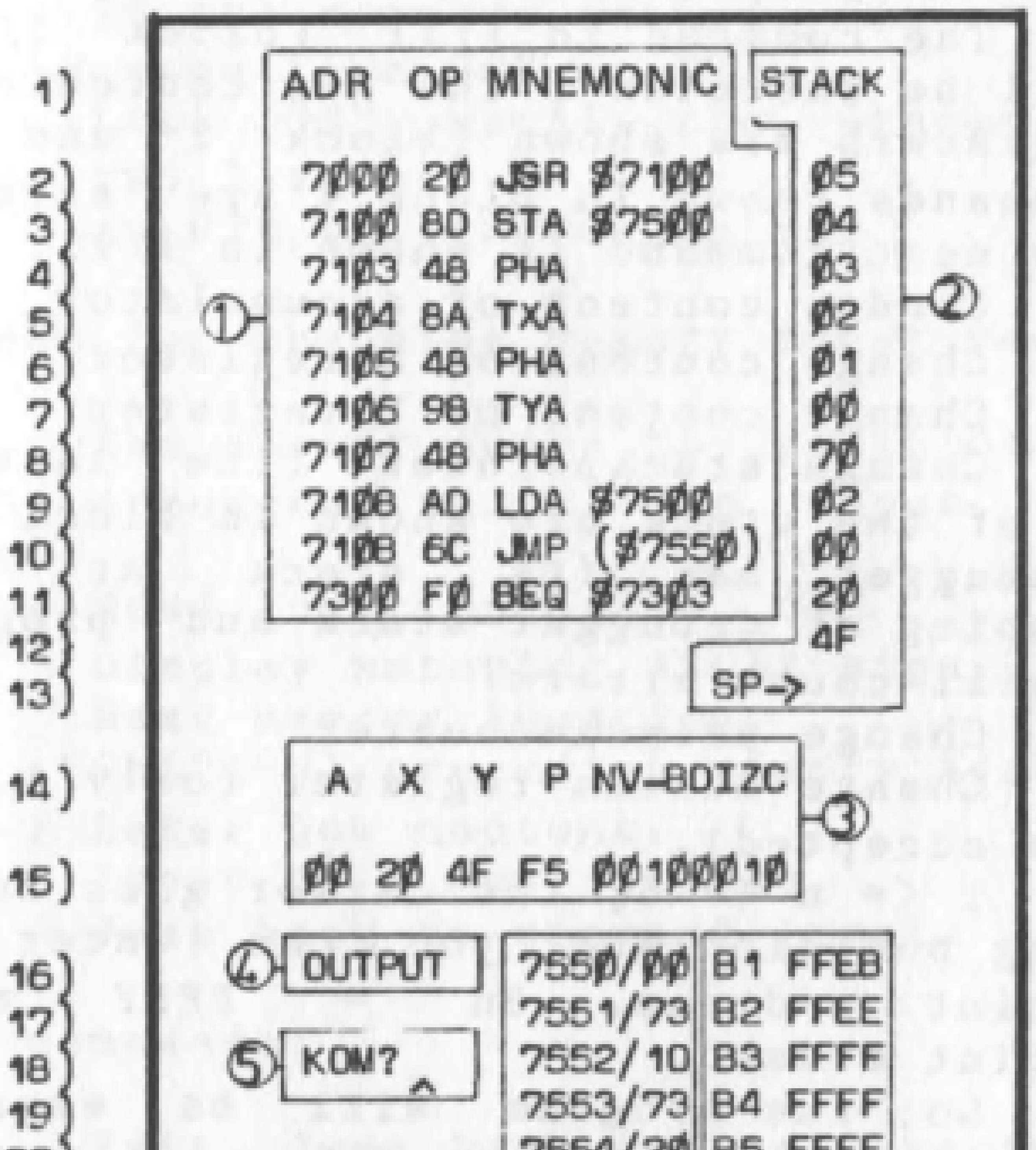
If you have already written programs in machine-language, you probably know how difficult it sometimes is to detect mistakes in such programs.
The definition of breakpoints sometimes helps localize the problem. However it is much better if you can go through a program in single steps. This (and more) can be done by the debugger described in this chapter.
After input and start of the program your screen should look like figure 1.

The debugger waits for the input of the start-address of the program to be tested. The cursor always shows you what you have to enter next.



```
    ADR  OP MNEMONIC    STACK

                                   0A
                                   09
                                   08
                                   07
                                   06
                                   05
                                   04
                                   03
                                   02
    ^                              01
                                   00
                           SP →

    A   X   Y   P   NV-BDIZC

    00 00 00 FF 00 100000

    OUTPUT                 B1  FFFF
                           B2  FFFF
    KOM?                   B3  FFFF
                           B4  FFFF
                           B5  FFFF
                           B6  FFFF
```

FIGURE 1



```
 1)      ADR  OP MNEMONIC      STACK

 2)      7ØØØ 2Ø JSR $71ØØ        Ø5
 3)      71ØØ 8D STA $75ØØ        Ø4
 4)      71Ø3 48 PHA              Ø3
 5)      71Ø4 8A TXA              Ø2
 6)      71Ø5 48 PHA              Ø1
 7)      71Ø6 98 TYA              ØØ
 8)      71Ø7 48 PHA              7Ø
 9)      71Ø8 AD LDA $75ØØ        Ø2
10)      71Ø8 6C JMP ($755Ø)      ØØ
11)      73ØØ FØ BEQ $73Ø3        2Ø
12)                               4F
13)                          SP->

14)       A  X  Y  P  NV-BDIZC

15)      ØØ 2Ø 4F F5  ØØ1ØØØ1Ø

16)  OUTPUT       755Ø/ØØ   B1 FFEB
17)               7551/73   B2 FFEE
18)  KOM?         7552/10   B3 FFFF
19)        ^      7553/73   B4 FFFF
20)               7554/2Ø   B5 FFFF
21)               7555/73   B6 FFFF
```

FIGURE 2

When the cursor is beside "KOM?" you have the following options:

BLANK The command in 1/11 (block 1/line 11) will be executed ; the new contents of the registers are shown (block 2 and 3); the commands shown in block 1 are scrolled up; the next command is shown in 1/11.

CA	Change content of accumulator
CX	Change content of X-register
CY	Change content of Y-register

CP Change stackpointer (the last 11 values of the stock are shown in block 2). The debugger has its stack at $28. Overlapping of debugger-stack and program-stack will cause errors!

CC Change programcounter

CS Change status-register (only 1 or 0 will be accepted).

Bn 1 <= n <= 6; the cursor goes to the matching position and you can enter the breakpoint address. Bn = FFFF means breakpoint erased.

G Go; the program will be executed under control of the debugger albeit slightly slower; if a breakpoint addres is hit or you press CTRL C (only if enabled) the program stops and the next command will be shown in 1/11.

Example: You want to test a program beginning at $1000 that works well to $1200. Set the programcounter using the CC command to $1000. Stepping through the program by using the space key up to $1200 would take a very long time. It is easier to define a breakpoint at $1200 and then enter the "G"-command.

K This command is usefull if you want to jump (not execute) one instruction.

Commands for subroutines:

W	If the command in 1/11 is a JSR, then the same than "G" to RTS.
S	Like "G" until the stackpointer becomes two larger.
R	An RTS will be executed.

Commands for changing memory locations:

M	The cursor jumps to position 6/21 and the debugger waits for input of an address.
/	Read location again.
"	Display matching ASCII sign.
+	Next memory location.
^	(Shift N) preceding memory location.
0-9,A-F	Enter new content.
CR	Input of a new address.
else	Jump back to command level.

Other commands:

E	Exit. Jump back to reset-vector.
N	New-start of program.
0	Disable CTRL
1	Enable CTRL
CTRL A	Show command being executed.
CTRL C	Like CTRL A then break.

All commands are echoed in block 5. Invalid commands cause a question mark as an echo. The commands CTRL A and CTRL C only have effect if CTRL has been enabled previously. If you try to go through the routine which reads the keyboard in single steps, it will not work correctly.

Special features of the debugger:
The debugger uses no zero-page address. The output-vector is changed by the debugger so that the characters to be displayed appear

beside the text OUTPUT. This prevents the screen from being overwritten.
If the debugger finds a non-existant OP-code, the address and the contents are displayed and the program branches to the "CC" command.
The command JMP ($XXFF) (where XX = any byte), which is handled wrong by the CPU, is handled right by the debugger.
Another CPU-fault: the PHP command always puts the I-flag = 1 to the stack. For that reason the I-flag always, even after BRK, is given out = 0.