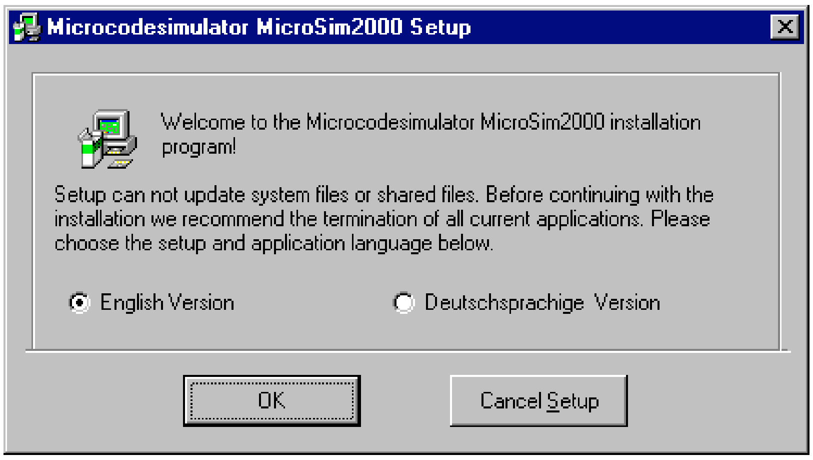
Getting Started
Microcodesimulator MicroSim2000

In the following an overview of the range of microcoding machine codes with the Microcodesimulator MicroSim2000 is being presented. By means of several specially selected examples, i.e. in which simply the register A (R1) of the Microcodesimulator by one, the functionality of the simulator is explained step by step. At first, the so called exploration mode the 49-bit microcode is investigated in its functionality, since only one single microcode itself can already control the execution of the incrementation of a register. In the next example, the microcode simulation mode is explained, that uses already programmed microcode examples provided by the simulator setup routine. The incrementation can be achieved in a sequence of microcode instructions (Example 1), in a looped sequence of microcode instructions (Example 2) or in a looped sequence of microcoded machine codes (Example3). Finally, a microcoded mashine code program is presented which calculates the faculty of an integer value.

Installation of the Microcodesimulator MicroSim2000

The Microcodesimulator MicroSim2000 is distributed with its own installation program that ease you the installation on your MS-Windows system. At the beginning of the installation the user can choose between two installation languages: English and German.


Microcodesimulator MicroSim2000 Setup

Welcome to the Microcodesimulator MicroSim2000 installation program!

Setup can not update system files or shared files. Before continuing with the installation we recommend the termination of all current applications. Please choose the setup and application language below.

[✓ English Version] Deutschsprachige Version

Fig. 1: Selection of the preferred start-up language and for the setup-up.

The choice of the installation language simultaneously determines the Microcodesimulator appearance at program start-up. By the installation the commando line is extended to „MicroSim.exe

0" or „MicroSim.exe **1**" for the English or German installation, respectively, indicating the start-up language. Starting MicroSim without a preferred language selection, English will be used as default. After having selected the start-up language, the setup program asks for the destination path where the simulation program should be installed in. After defining the destination directory the installation carries on without interruption. During the installation, a new program group is created in the start menu where a links to the executable file, the help files and the documentation are created.

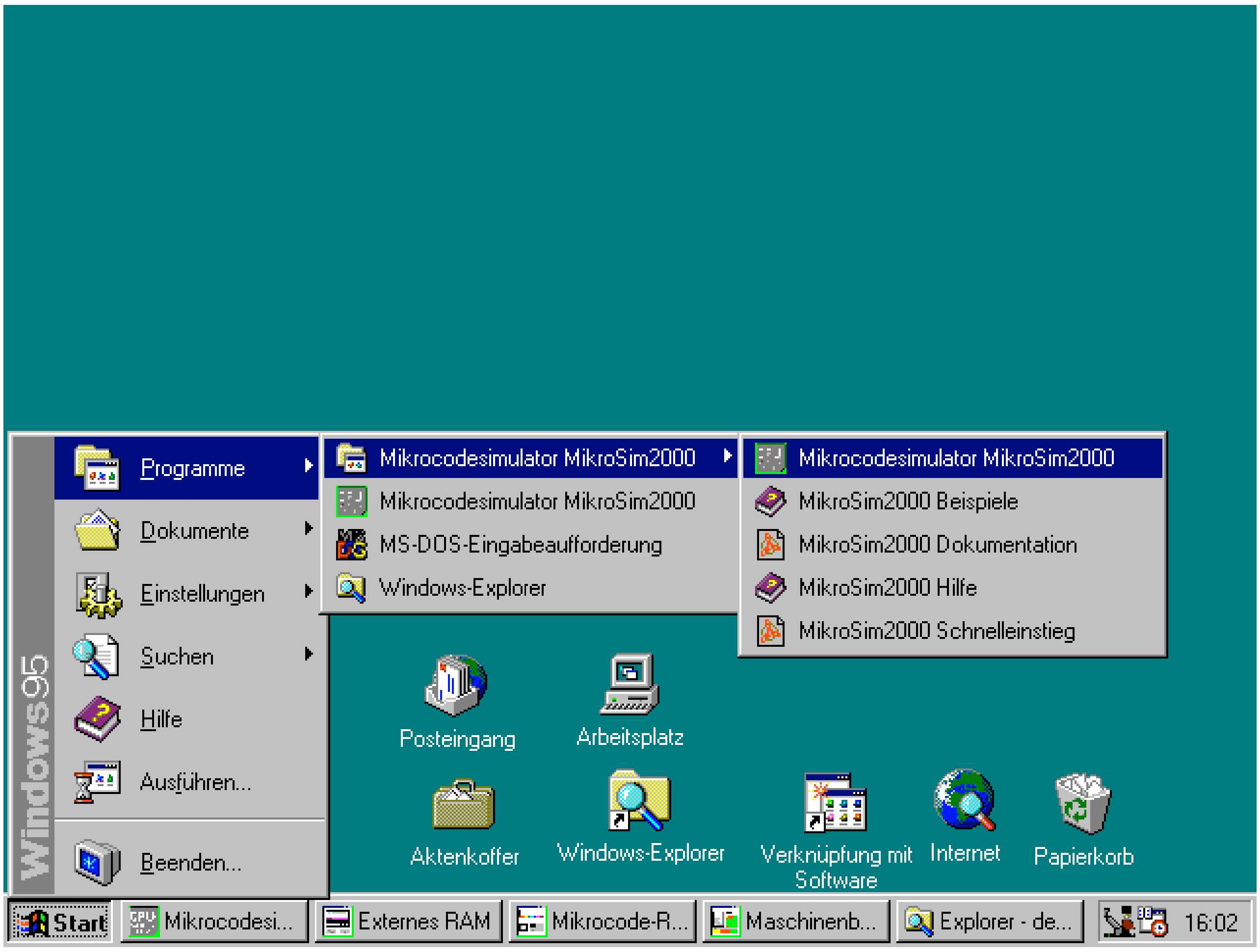


Fig. 2: During the Microcodesimulator program installation a new program group and an application link is created in the start menu in MS-Windows 95/98/NT/2000

The Microcodesimulator is registered as a 32-bit application in MS-Windows registry and can be uninstalled on demand. For the case that no installation discs are available, the Microcodesimulator should also get started, if the EXE-file „**MICROSIM.EXE**" can find the library files „**VB40032.DLL**", „**CTL3D32.DLL**", and „**COMDLG32.OCX**" in the system directory of windows. These dynamic link libraries are usually used by other programs, too, and are often already preinstalled.

Starting the Simulator in the Exploration Mode

The microcode simulator MicroSim2000 is started by clicking onto the green-grey coloured CPU icon. The start-up window is in the beginning green which symbolises a non constructed CPU, i.e an unassembled printed circuit board. The user is in the so-called *"exploration mode"*. In this mode the CPU can be built up successively component wise by entering the menu topic "*Options/Simulation Levels*". Successively, the registers, the arithmetic-logical unit, the external RAM, the microcode ROM, and the microcode address-calculator can be added to the CPU. Before resetting the CPU simulator by pressing the "Reset" button in the lower right corner of the main window, the registers of all CPU components are filled with their names that indicate the usage of them and relation to the microcode (Fig. 4).

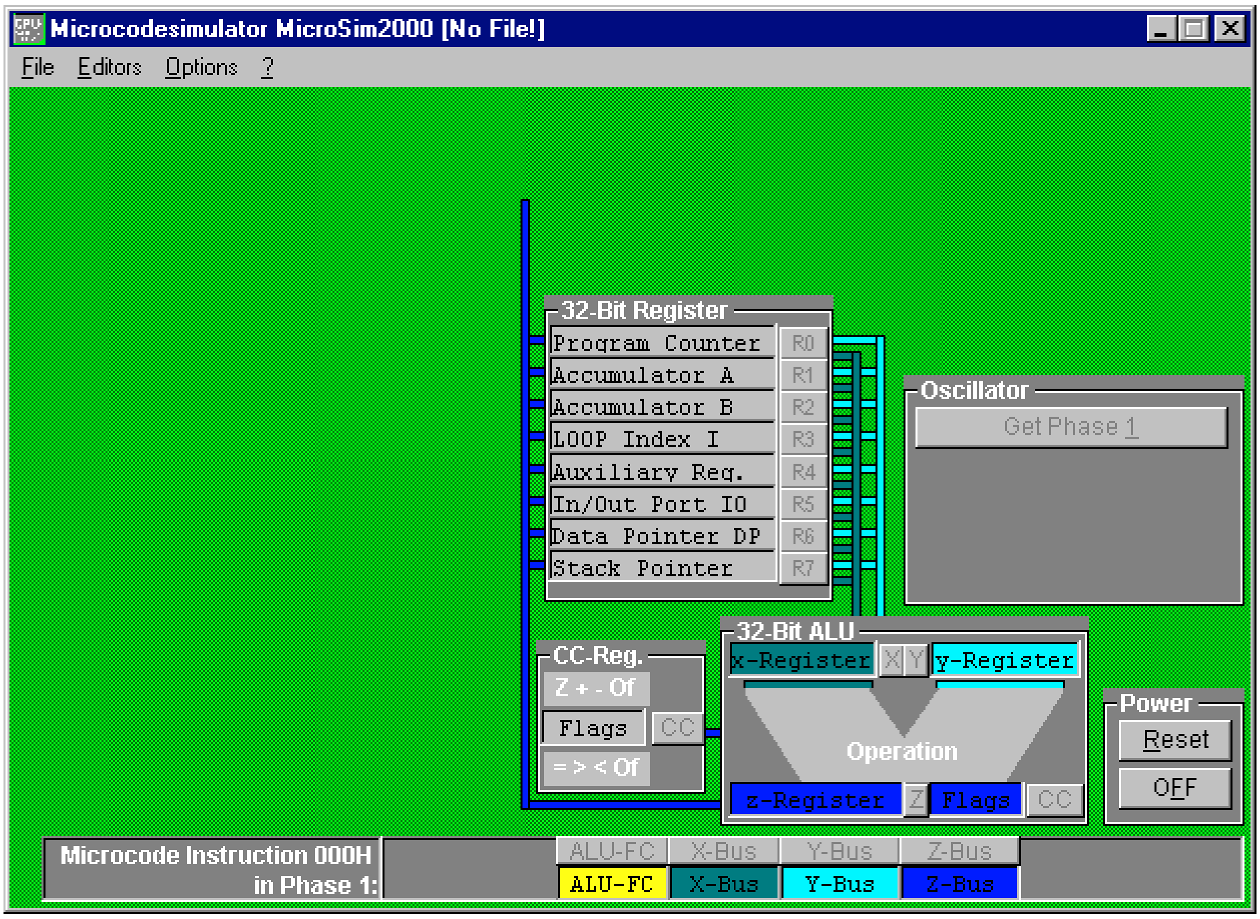


Fig. 3: Stepwise construction of the Microcodesimulator in the exploration mode..

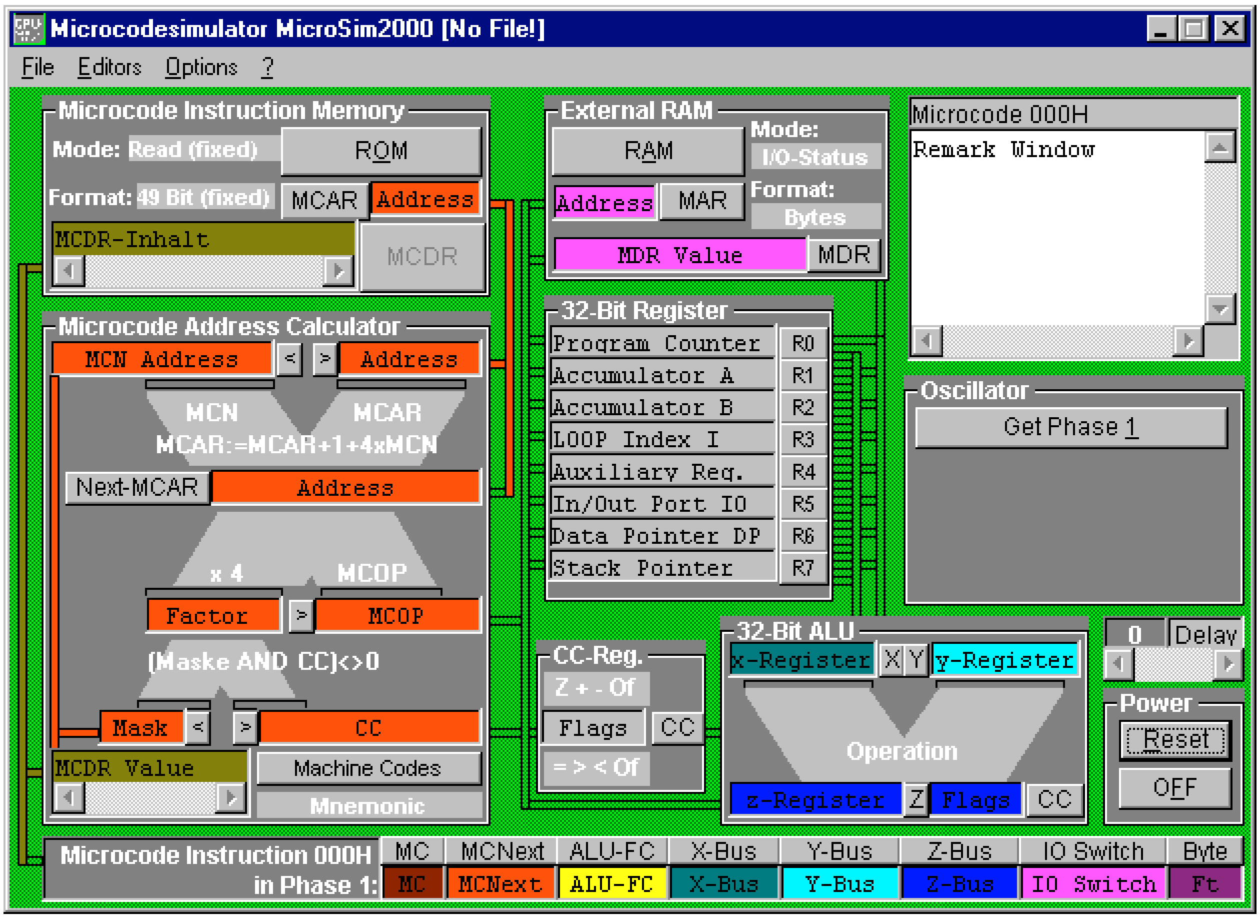
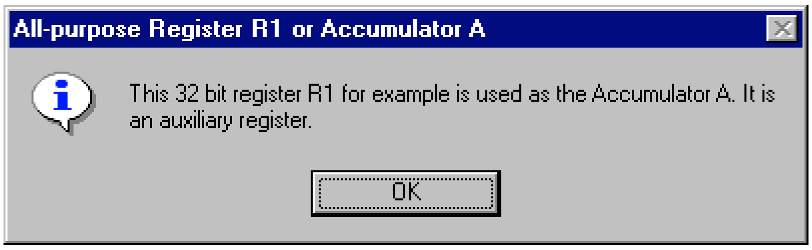


Fig. 4: Completely constructed Microcodesimulator in the exploration mode.

In the Microcodesimulator all registers and CPU components reveal information, when the standard mouse pointer changes its shape to a help mouse pointer . Clicking onto such a marked area, an information window is opened with additional hints or comments (Fig. 5).



All-purpose Register R1 or Accumulator A

This 32 bit register R1 for example is used as the Accumulator A. It is an auxiliary register.

OK

Fig. 5: Here, information to the register R1 and its usage is given.

After the user is got familiar with the CPU, the functionality of the 49-bit microcode that is displayed at the bottom of the main window can be explored in more detail. For simplicity in this step all other confusing elements of the CPU should be hidden by entering the menu item *„Options/Simulation Level/... and RAM“* (Fig. 6).

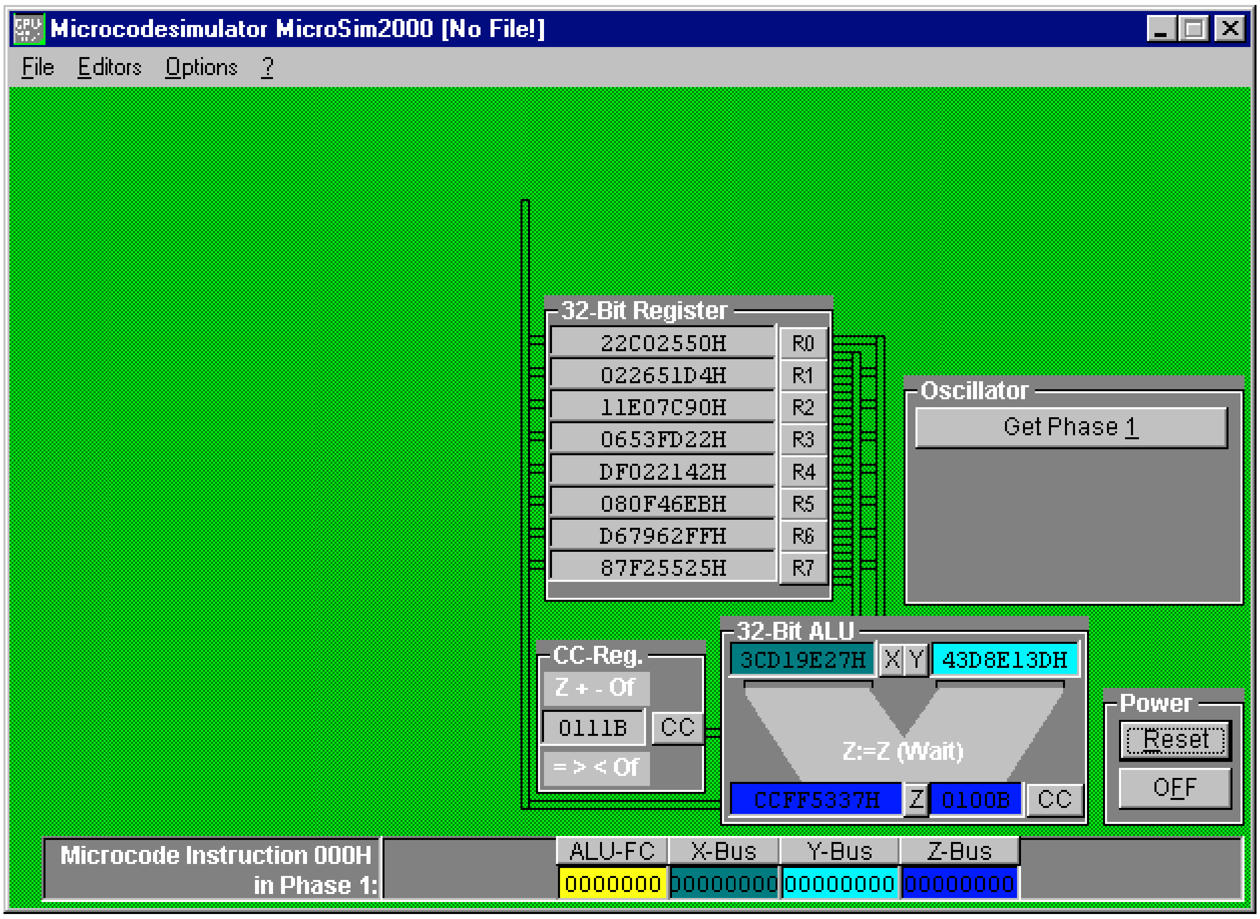
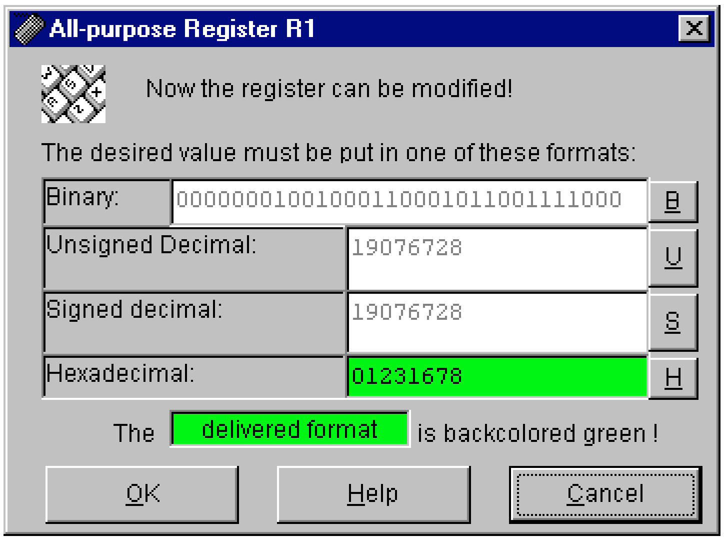


Fig. 6: Initialised Microcodesimulator in the exploration mode after pressing the “Reset” button.

In order to investigate the single bits of the four selected bit groups of the 49 bit long microcode, please initialise the simulator by pressing the reset button first. As shown in Fig. 6, all registers have now binary or hexadecimal values instead of their descriptive labels.


All-purpose Register R1

Now the register can be modified!

The desired value must be put in one of these formats:

Binary:	00000001001000110001011001111000	B
Unsigned Decimal:	19076728	U
Signed decimal:	19076728	S
Hexadecimal:	01231678	H

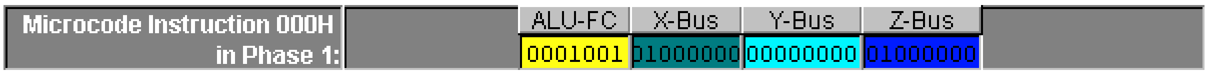
The delivered format is backcolored green !

OK Help Cancel

Fig. 7: By means of this special number editor the values of a selected register (here R1) can be modified.

Values of all registers can be modified by means of a special editor in which the value can be entered in the binary, hexadecimal, the unsigned decimal, and the signed decimal format. The value entered in one of these formats is converted immediately into the other number formats. After entering the values and leaving the editor with „OK“, the value and that format of the modifying register is transferred into the register that was chosen for modification. The number format of the value transferred is back coloured green. Depending on the usage of the register as a switch or as a number the preferred format is binary or hexadecimal, respectively.

In this way the Microcodesimulator can be explored in the exploration mode bit by bit. For example writing “01000000“ into the x-bus and z-bus group of the actual microcode command, one can trace by pressing the oscillator button how the value of the 32-bit register R1 is loaded into the x-register of the ALU and the z-register value is transferred back to the 32-bit register R1. During this operation the responsible circuits for the data transfer are highlighted. Setting additionally the 7-bit value of the ALU function code to binary 9, in each calculate phase the x-register is incremented by one and stored in the z-register of the ALU. So a complete get-calculate-put cycle increments the register R1 by one.



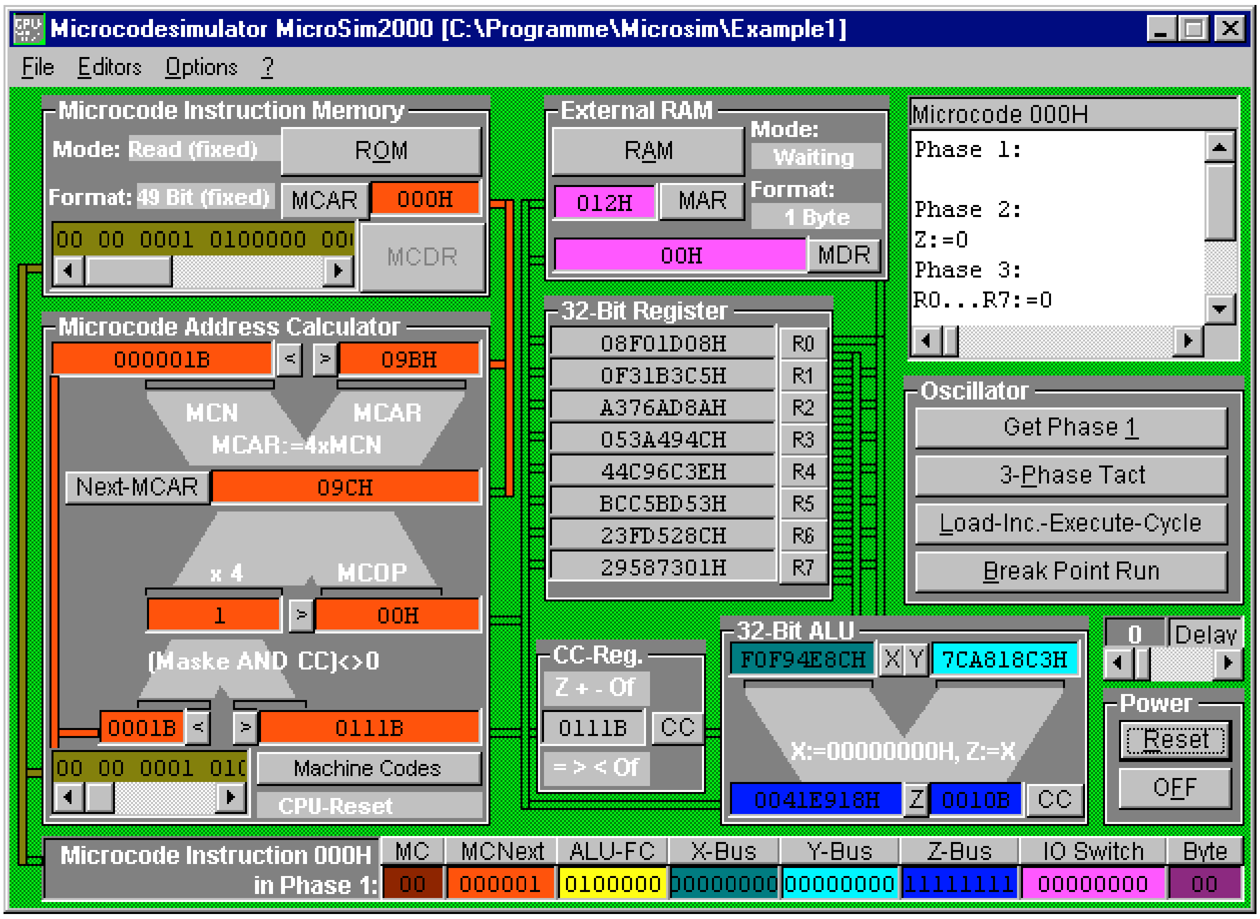
Microcode Instruction 000H in Phase 1:	ALU-FC	X-Bus	Y-Bus	Z-Bus
	0001001	01000000	00000000	01000000

Fig. 8: This microcode instruction in the bottom line of the main window increments the register R1.

Similar as explained for the increment cycle all other bits can be investigated. Several reference tables in the help file gives a detailed overview about the meaning of each bit of the microcode instruction. In order to train the usage of the Microcodesimulator, specially prepared example files are discussed in the following. An example file can be loaded using the menu item.„*File/Open* ...“.

Microcoding Example #1:

As already mentioned in the previous section, in the exploration mode one single microcode instruction is executed, namely that which is displayed in the bottom window area. The next step is now to explore how a sequence of 49-bit long microcode instructions are executed successively. By loading or creating new micrcocodesimulator files the exploration mode is quit. Loading “Example1.ROM” via the menu item „*File/Open* ...“ all other related files of the type “Example1.*” are loaded, too. After loading the files the complete CPU simulator is set up. The exploration mode is taken over from the microcodesimulation mode. Even in that simulation mode one is able to control the execution step by step by changing for individual bits of the microcode or register values.



Mode: Read (fixed)
Format: 49 Bit (fixed)
MCAR 000H
00 00 0001 0100000 00
000001B 09BH
Next-MCAR 09CH
1 00H
0001B 0111B
00 00 0001 010

External RAM – Mode: Waiting; Format: 1 Byte
MAR 012H
MDR 00H

32-Bit Register	
08F01D08H	R0
0F31B3C5H	R1
A376AD8AH	R2
053A494CH	R3
44C96C3EH	R4
BCC5BD53H	R5
23FD528CH	R6
29587301H	R7

CC-Reg.: 0111B
32-Bit ALU: X F0F94E8CH, Y 7CA818C3H, Z 0041E918H, CC 0010B
Delay: 0

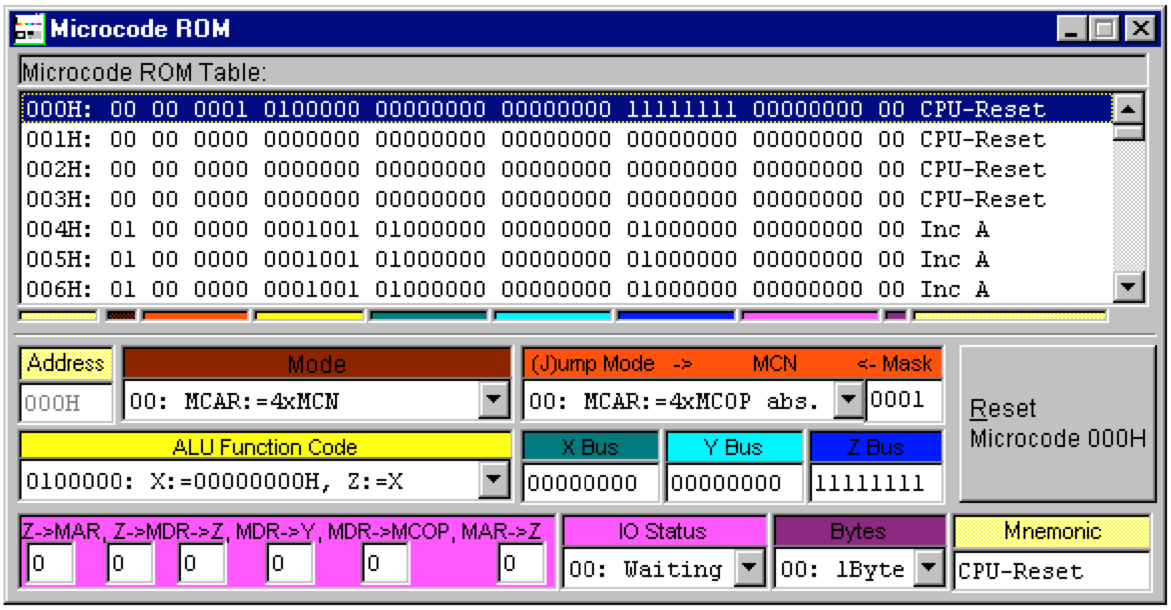
Microcode 000H
```
Phase 1:

Phase 2:
Z:=0
Phase 3:
R0...R7:=0
```

Microcode Instruction 000H in Phase 1:	MC	MCNext	ALU-FC	X-Bus	Y-Bus	Z-Bus	IO Switch	Byte
	00	000001	0100000	00000000	00000000	11111111	00000000	00

Fig. 9: Initialised Microcodesimulator that has loaded the microcoding example “Example1”.

Load the example “Example1” into the simulator and perform an initialisation pressing the reset button. All buttons of the simulator are no enabled, such as the ROM button in the upper left corner in the microcode ROM. Pressing this button, the ROM editor opens with the content of the microcode memory. In the example only some microcodes are programmed in the first eight rows.


Microcode ROM

Microcode ROM Table:

```
000H:  00  00  0001  0100000  00000000  00000000  11111111  00000000  00  CPU-Reset
001H:  00  00  0000  0000000  00000000  00000000  00000000  00000000  00  CPU-Reset
002H:  00  00  0000  0000000  00000000  00000000  00000000  00000000  00  CPU-Reset
003H:  00  00  0000  0000000  00000000  00000000  00000000  00000000  00  CPU-Reset
004H:  01  00  0000  0001001  01000000  00000000  01000000  00000000  00  Inc A
005H:  01  00  0000  0001001  01000000  00000000  01000000  00000000  00  Inc A
006H:  01  00  0000  0001001  01000000  00000000  01000000  00000000  00  Inc A
```

Address: 000H
Mode: 00: MCAR:=4xMCN
(J)ump Mode -> MCN <- Mask: 00: MCAR:=4xMCOP abs. | 0001
Reset Microcode 000H
ALU Function Code: 0100000: X:=00000000H, Z:=X
X Bus: 00000000
Y Bus: 00000000
Z Bus: 11111111
Z->MAR, Z->MDR->Z, MDR->Y, MDR->MCOP, MAR->Z: 0 0 0 0 0 0
IO Status: 00: Waiting
Bytes: 00: 1Byte
Mnemonic: CPU-Reset

Fig. 10: Microcode-ROM contents of the microcode editor. Each selected microcode instruction listed in the ROM table can be modified bit group wise.

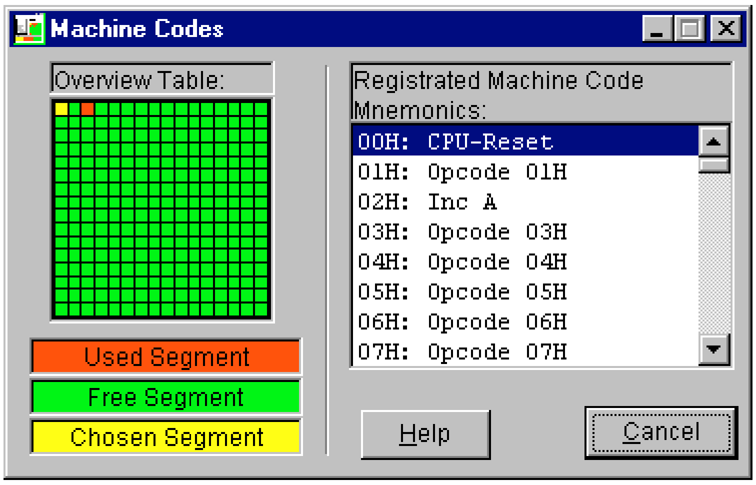
After resetting the simulator the first microcode instruction is always the one at address 000H. So one uses the opportunity to initialise all 32-bit register of the CPU by programming a CPU reset sequence of instructions. In this example all registers set to zero. This is achieved by generating the constant zero in the ALU in the calculate phase and transferring it into all eight registers. Since the MCNext command points to the microcode address 004H, this microcode instruction is being executed next.

The microcode instructions from address 004H to 007H increment the register R1 by one in a 3-phase tact. Since the next microcode address is deduced from the actual address, the next one is calculated by incrementation except the address which is calculated at the microcode 007H. Here, the microcode address calculator determines the address 004H as the address for the next microcode instruction. So, one is entering an endless loop.

Microcoding Example #2:

In this example program “Example2”, a bit complex initialisation is executed that sets first all registers to 0 and then the stack pointer to 800H. That address is the address 000H of the 2 kB external RAM. By decrementing the stack register value by the number of pushed bytes at the end of the external RAM and by incrementing the stack register value by the number of popped bytes one is able to track the state of stored bytes in the stack.

At segment 02H the endless incrementation loop of “Example1” starts. As one can see in the table of segment descriptions, two segments have been already programmed, namely the “CPU-Reset” command and the command “INC A”. Since the segment description is also used as the machine codes’ mnemonic, these two machine codes are implemented. A special editor for machine codes gives a quick overview about segments that are used and those which are unused. As shown below in this example only the segments 00H ("CPU-Reset") and 01H ("INC A") are in use.



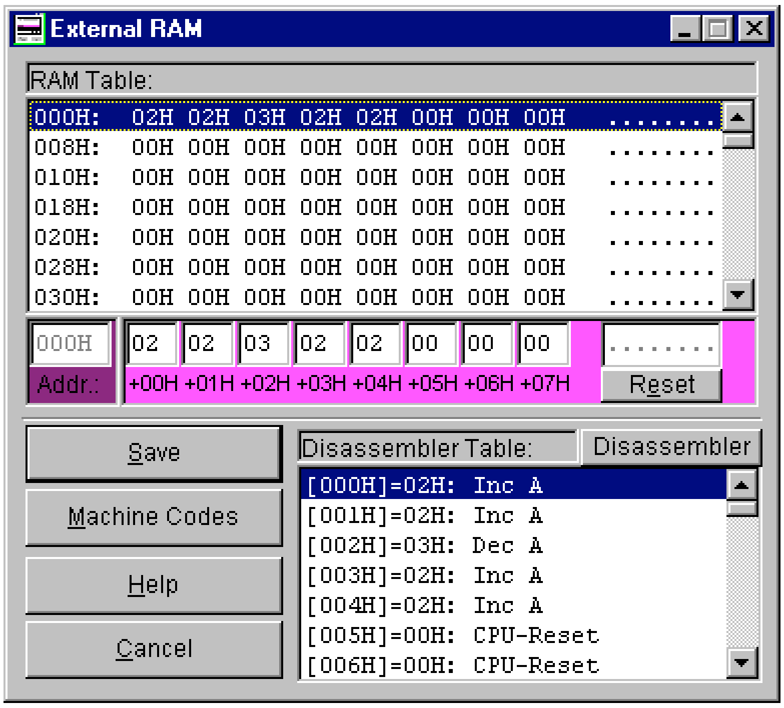
Registrated Machine Code Mnemonics:

```
00H: CPU-Reset
01H: Opcode 01H
02H: Inc A
03H: Opcode 03H
04H: Opcode 04H
05H: Opcode 05H
06H: Opcode 06H
07H: Opcode 07H
```

Fig. 11: Machine code table. This editor gives a quick overview of already used and unused segments in the microcode ROM.

Microcoding Example #3:

In the microcoding example “Example3” the incrementation of the register R1 is done by a machine code program that is situated at RAM address 000H to 004H. The RAM content can be viewed in the RAM editor accessible by the menu item “*Editors/RAM Editor*” or by pressing the RAM button of the external RAM in the main window.



```
RAM Table:
000H:  02H 02H 03H 02H 02H 00H 00H 00H   ........
008H:  00H 00H 00H 00H 00H 00H 00H 00H   ........
010H:  00H 00H 00H 00H 00H 00H 00H 00H   ........
018H:  00H 00H 00H 00H 00H 00H 00H 00H   ........
020H:  00H 00H 00H 00H 00H 00H 00H 00H   ........
028H:  00H 00H 00H 00H 00H 00H 00H 00H   ........
030H:  00H 00H 00H 00H 00H 00H 00H 00H   ........

000H   02  02  03  02  02  00  00  00   ........
Addr.: +00H +01H +02H +03H +04H +05H +06H +07H

Disassembler Table:
[000H]=02H: Inc A
[001H]=02H: Inc A
[002H]=03H: Dec A
[003H]=02H: Inc A
[004H]=02H: Inc A
[005H]=00H: CPU-Reset
[006H]=00H: CPU-Reset
```

Fig. 12: External RAM content of the microcoding example "Example3". The machine code sequence 02H-02H-03H-02H-02H causes a incrementation and decrementation of the register R1 (Accumulator A).

The machine code sequence 02H-02H-03H-02H-02H on the first addresses of the 2 kB large RAM causes two incrementations by “02H”, than a decrementation by “02H” and after this again two incrementations by “02H”. After the last incrementation the next machine code “00H” is the CPU-reset command that initialises the program counter and all other registers and the incrementation starts again. Here an incrementation endless loop has been programmed on a machine code level, too. The reason, why this kind of machine code sequence causes an endless loop is discussed next.

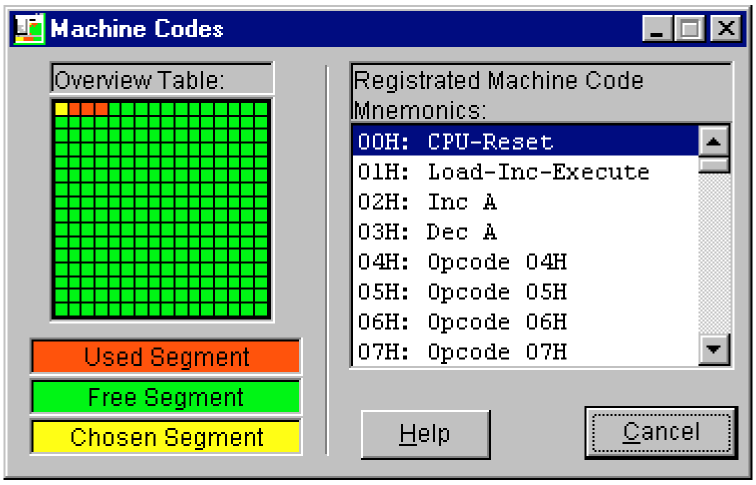
The machine codes „CPU-Reset“, „INC A“, and „DEC A“ are situated at the microcode ROM segments „00H“, „02H“, and „03H“ as shown in the figure below. Additionally one very important machine code s implemented too at segment 01H. This machine code is used as a machine code load command from the external RAM. This machine code load command “01H” is called „LOAD-INCREMENT-EXECUTE“ command. The microcode sequence of the „LOAD-INCREMENT-EXECUTE“ command has to:

1. *Load* the first byte of the external RAM, on which the the program counter value of register R0 points at and to interpret that byte as microcode address of the microcode ROM (MCOPx4).

2. *Increment* the program counter value of register R0 in order to point on to the next machine code after microcode instruction execution. (Additional incrementations of R0 has to be programmed for machine codes commands larger than 1 byte)

3. Execute the microcode sequence beginning at MCOPx4. After execution of the microcode sequence one has to assure that the last microcode address calculation command generates the address 004H which is the first microcode of the „LOAD-INCREMENT-EXECUTE“ command.

If a „LOAD-INCREMENT-EXECUTE“ command is implemented as shown in this example, the execution of microcode instructions can be done single stepwise and 3-phase tact wise as well as in a whole Load-Increment-Execute cycle. One can start that cycle with the oscillator button with the caption “Load-Inc.-Excecute-Cycle”. Which of the machine codes are performed can be seen in right below the machine code button in the microcode address calculator component.

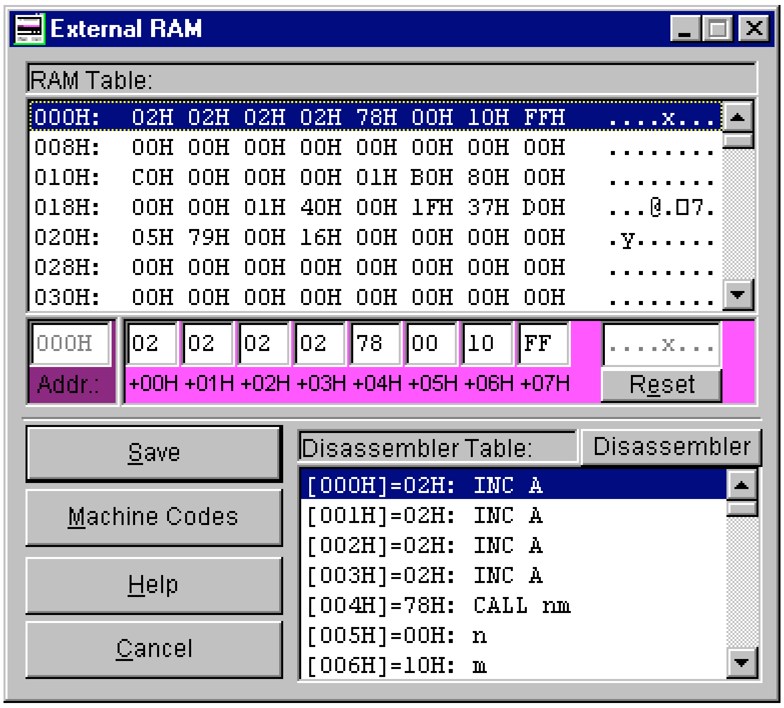


```
00H:  CPU-Reset
01H:  Load-Inc-Execute
02H:  Inc A
03H:  Dec A
04H:  Opcode 04H
05H:  Opcode 05H
06H:  Opcode 06H
07H:  Opcode 07H
```

Fig. 13: The machine code editor showing the machine codes of Example3.

Microcoding Example #4:

In the micrcoding example “Example4” a machine code program is written that calculates the falculty of the number entered in register R1. As an example R1 is loaded here by 4 by incrementation as mentioned in example 3. After incrementation the machine code program is executed by calling address „00 10H“ with the command „CALL 00 10“. From address 10H on the faculty actually is calculated. The result is given in register R2. This register is initialised by 1 since the faculty of 0 is 1. The value of register R2 is multiplied by R1 and R1 is decremented by one as long as R1>0. If R1=0 the RETURN command loads the program address from the stack. The next command that will be executed is the Break-Point command FFH that stops the machine code run. In the following Fig. 14 the RAM content of example4 is shown.



```
000H:   02H 02H 02H 02H 78H 00H 10H FFH   ....x...
008H:   00H 00H 00H 00H 00H 00H 00H 00H   ........
010H:   C0H 00H 00H 00H 01H B0H 80H 00H   ........
018H:   00H 00H 01H 40H 00H 1FH 37H D0H   ...@.□7.
020H:   05H 79H 00H 16H 00H 00H 00H 00H   .y......
028H:   00H 00H 00H 00H 00H 00H 00H 00H   ........
030H:   00H 00H 00H 00H 00H 00H 00H 00H   ........
```

```
000H   02   02   02   02   78   00   10   FF    ....x...
     +00H +01H +02H +03H +04H +05H +06H +07H
```

```
[000H]=02H:  INC  A
[001H]=02H:  INC  A
[002H]=02H:  INC  A
[003H]=02H:  INC  A
[004H]=78H:  CALL nm
[005H]=00H:  n
[006H]=10H:  m
```

Fig. 14: External RAM shows the machine code program that calculates the faculty of a number (here 4) that is hold in register R1. The result of the calculation is put displayed in register R2.

The machine code program is started by pressing the reset button first and the oscillator button “Break-Point-Run”. At the end of the simulation, in register R2 the hexadecimal value 18H (decimal 24) for the faculty of 4 (4!=1*2*3*4=24) is displayed. The simulation stops automatically, since the 1-byte Break-Point command “FFH” is detected.

The simulation of the machine code program calculation can be individually delayed or stepwise executed. In the following a printout of the assembler program of the RAM is listed:

Content of the External RAM:

```
000H: 02H 02H 02H 02H 78H 00H 10H FFH 00H 00H 00H 00H 00H 00H 00H 00H
010H: C0H 00H 00H 00H 01H B0H 80H 00H 00H 00H 01H 40H 00H 1FH 37H D0H
020H: 05H 79H 00H 16H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
030H: 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
```

Disassembler Table:

```
[000H]=02H: INC A
[001H]=02H: INC A
[002H]=02H: INC A
[003H]=02H: INC A
[004H]=78H: CALL nm
[005H]=00H: n
[006H]=10H: m
[007H]=FFH: Break-Point, Halt
[008H]=00H: CPU-Reset
[009H]=00H: CPU-Reset
[00AH]=00H: CPU-Reset
[00BH]=00H: CPU-Reset
[00CH]=00H: CPU-Reset
[00DH]=00H: CPU-Reset
[00EH]=00H: CPU-Reset
[00FH]=00H: CPU-Reset
[010H]=C0H: MOVB, nmlk
[011H]=00H: n
[012H]=00H: m
[013H]=00H: l
[014H]=01H: k
[015H]=B0H: MOVAux, A
[016H]=80H: CMPAux, nmlk
[017H]=00H: n
[018H]=00H: m
[019H]=00H: l
[01AH]=01H: k
[01BH]=40H: JG nm  (Jump)
[01CH]=00H: n
[01DH]=1FH: m
[01EH]=37H: RETURN
[01FH]=D0H: MULB, Aux
[020H]=05H: DEC Aux
[021H]=79H: JP nm
[022H]=00H: n
[023H]=16H: m
```