Schnell-Einstieg in den Mikrocodesimulator MikroSim2000

Die folgenden Seiten sollen einen kurzen Überblick über die Tragweite der Mikrocodesimulation mit Hilfe des vorliegenden Mikrocodesimulators MikroSim2000 geben. An Hand eines ausgewählten Beispiels, in dem das Register A (R1) des Mikrocodesimulators schrittweise um Eins erhöht wird, wird die Funktionsweise des Simulators auf verschiedenen Simulationsebenen vorgestellt. Zunächst wird im „Erkundungsmodus" der 49-Bit lange Mikrocode näher erläutert, der bereits für sich alleine eine Inkrementierung um Eins bewerkstelligen kann. Anschließend wird in den Mikrocode-Simulationsmodus übergewechselt, der mit drei vorgefertigten Beispieldateien betrieben werden kann, die dem Installationsprogramm des Mikrocodesimulators beigelegt sind. In diesen Beispielen kann die Inkrementierung des Registers A entweder in einer Folge von Mikrocodes (Beispiel1), in einer Mikrocode-Programmschleife (Beispiel2) oder in einer Sequenz mikrocodeprogrammierter Maschinensprachebefehle (Beispiel3) durchgeführt werden. Im letzten Beispiel 4 wird ein mikrocodeprogrammiertes Maschinenspracheprogramm zur Berechnung der Fakultät vorgestellt.

Installation des Mikrocodesimulators MikroSim2000

Der Mikrocodesimulator MikroSim2000 wird als Programmpaket mit einem bereitgestellten Setup-Programm installiert. Beim Setup-Start kann als Installationssprache zwischen Deutsch und Englisch gewählt werden (Abb. 1), die gleichzeitig die Startsprache des zweisprachig ausgelegten Mikrocodesimulators ist.

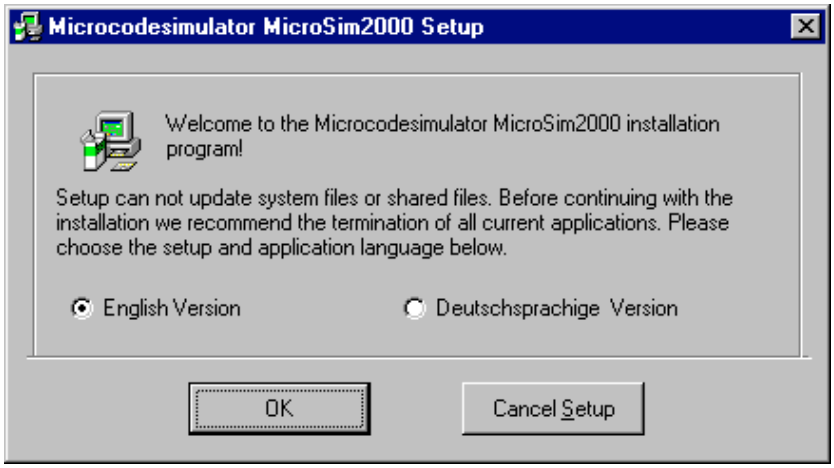

Abb. 1: Setup-Programm mit Sprachauswahl

Mit der Kommandozeilenanweisung „MikroSim.exe **0**“ und „MikroSim.exe **1**“ kann zwischen der englischen (Standard) bzw. der deutschen Sprachvoreinstellung gewählt werden. Ohne Komandozeilenparameter startet der Mikrocodesimulator standardmäßig in der englischen Version. Im Mikrocodeprogramm selbst kann jederzeit zwischen der deutschen und der englischen Version gewechselt werden. Das Setup-Programm fragt bei der Installation nach dem Verzeichnis, in dem das Programm eingerichtet werden soll. Nach dem Bestätigen des Zielverzeichnisses oder der Angabe eines neu zu erstellenden Zielverzeichnisses läuft die Installation von alleine ab. Mit dem Setup wird ein neuer Programmgruppen-Eintrag in die Startleiste eingefügt, in der der Mikrocodesimulator, die Hilfedateien und die Beispieldateien in der entsprechend gewählten Sprache als Programmsymbol abgelegt werden (Abb. 2).

Abb. 2: Bei der Programm-Installation wird der Mikrocodesimulator in einer Programmgruppe und als Applikation in das Startmenü unter MS-Windows95/98/NT/2000 aufgenommen.

Der Mikrocodesimulator wird mit Hilfe des Setup-Programms als 32-Bit-Applikation in die Registry von Windows eingefügt und kann auf Wunsch deinstalliert werden. Daher sollten sowohl Programminstallation als auch -deinstallation über die Systemsteuerung erfolgen. Liegt kein vollständiges Setup-Program vor, kann der Mikrocodesimulator auch ohne große Mühe zum Laufen gebracht werden, wenn das Simulationsprogramm „**MIKROSIM.EXE**“ die Library-Dateien „**VB40032.DLL**“, „**CTL3D32.DLL**“ und „**COMDLG32.OCX**“ in dem Systemverzeichnis von Windows oder dem Programmverzeichnis direkt vorfindet. Oftmals sind diese Dateien bereits von anderen Programminstallationen abgelegt worden und müssen nicht unbedingt nachträglich installiert werden.

Starten des Simulators im Erkundungsmodus

Der Mikrocodesimulator MikroSim2000 wird durch Klicken auf das graugrünfarbene CPU-Programm-Symbol gestartet. Der Mikrocodesimulator öffnet anfangs ein grünes leeres Fenster, das eine unbestückte Platine symbolisieren soll. Der Benutzer befindet sich in dem sogenannten „*Erkundungsmodus*“. In diesem Modus kann die CPU schrittweise aufgebaut werden, indem unter Zuhilfenahme des Menüpunkts „*Optionen/Simulationsstufe*“ nacheinander die leere Platine mit CPU-Bausteilen wie dem 32-Bit Register, der Recheneinheit (ALU), dem externe RAM, dem Mikrocode-ROM und dem Mikrocode-Adressrechner bestückt wird.

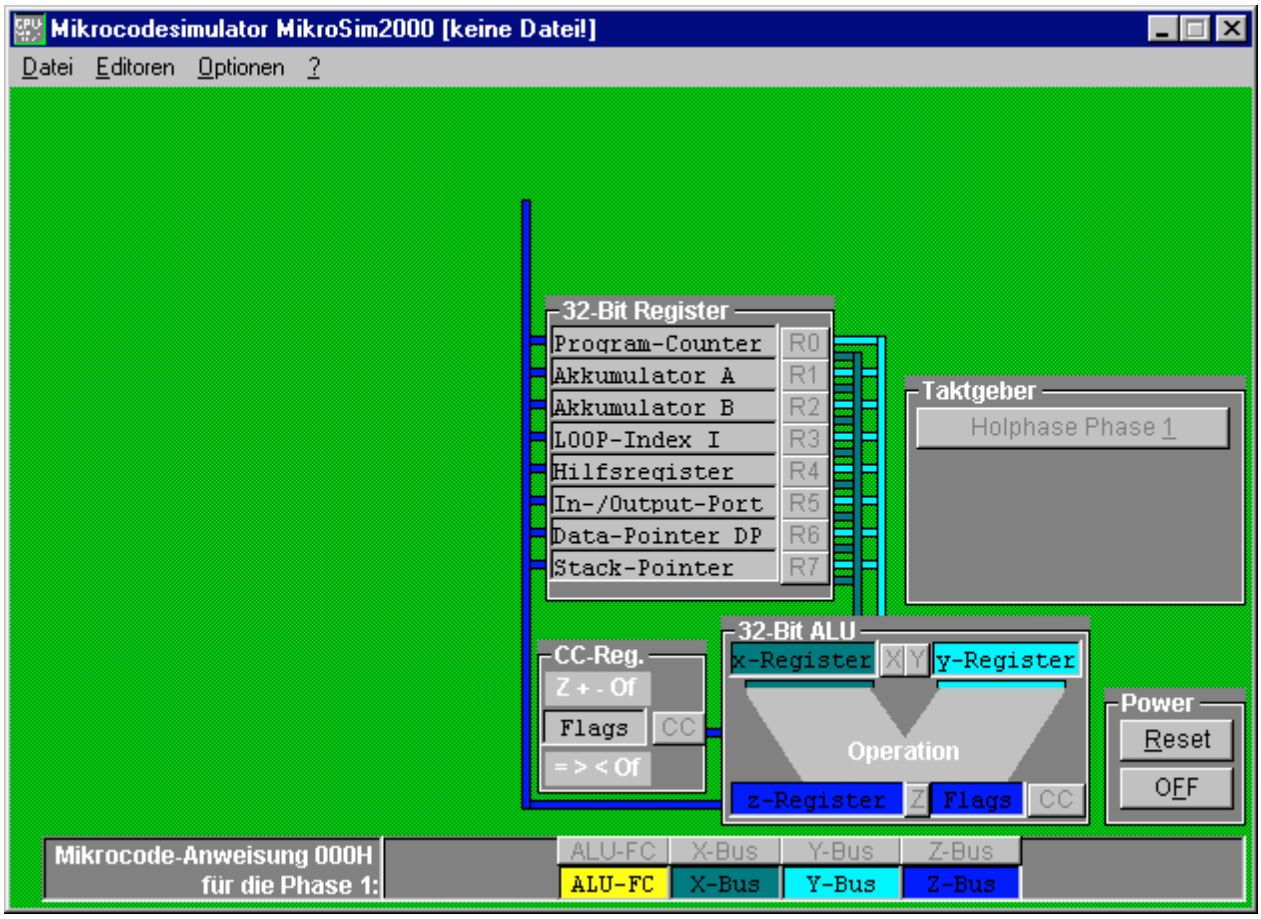

Abb. 3: Schrittweiser Aufbau des Mikrocodesimulators im Erkundungsmodus. Die farblich markierten Register und Datenleitungen werden von den entsprechend gefärbten Bits des Mikrocodes gesteuert.

Die Register tragen vor der Initialisierung mittels „Reset“ eine erläuternde Beschriftung, die deren Funktion und Verwendung andeuten. Beim Initialisieren mittels Reset wird die Beschriftung der Register durch zufällige Zahlenwerte überschrieben. Auch nach einem Reset oder während der Simulation können die Registerbezeichnungen eingeblendet werden mittels des Menüs „*Optionen/Sichtbarkeit/Registerbezeichnungen*“. Mit dem Hinzufügen des letzten Bausteins, dem "Mikrocode-Adressrechner", ist die CPU komplett (Abb. 4).

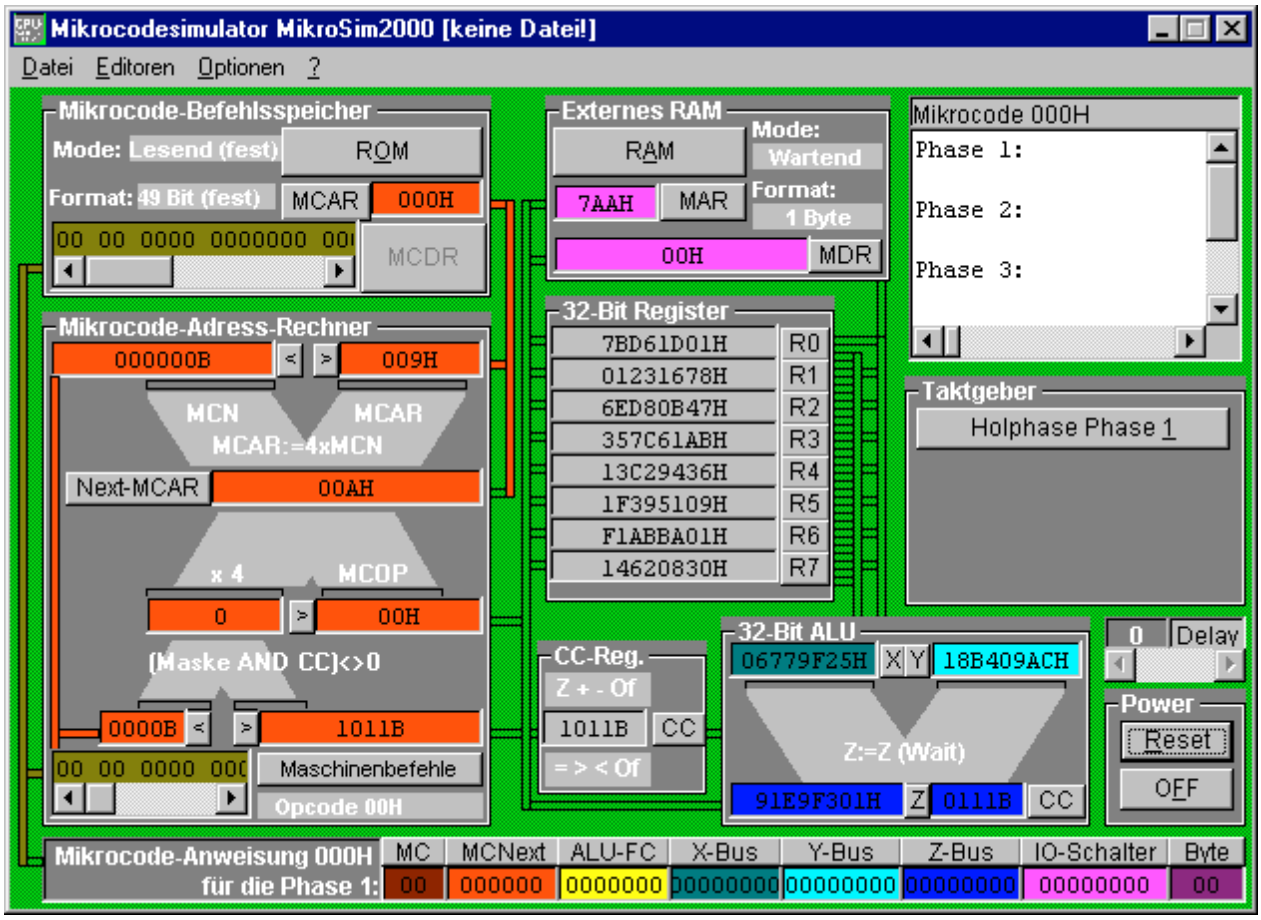

Abb. 4: Vollständig aufgebauter Mikrocodesimulator im Erkundungsmodus. Durch den Reset des Simulators wurden die Register der Modell-CPU mit zufälligen Zahlenwerten gefüllt.

Zusätzliche Informationen über den Aufbau und die Verwendung der eingesetzten Registern (farbige Textfelder auf grauem Hintergrund) können überall dort erhalten werden, wo der Standardmauszeiger in einen Mauszeiger mit angehängtem Fragezeichen überwechselt. Mit einem Mausklick auf solche Objekte öffnet sich ein Fenster mit wissenswerten Zusatzinformationen (Abb. 5).

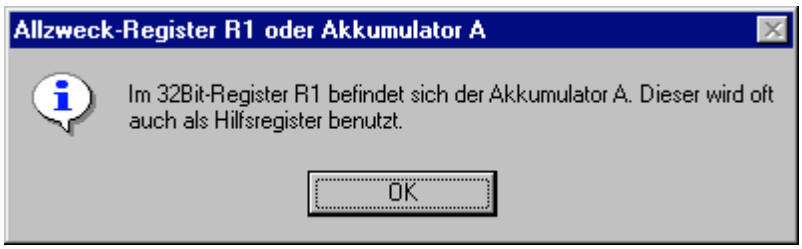

Abb. 5: Informationen zum Register R1. Informationsfenster wie dieses können geöffnet werden, wenn bestimmte Bereiche im Mikrocodesimulator angeklickt werden.

Nach dem Erkunden des Simulators ist der Benutzer näher mit den Bezeichnungen der CPU vertraut. Nun kann sich der Wirkungsweise der einzelnen Bits des am unteren Bildschirmrand

dargestellten Mikrocodes gewidmet werden. Zwecks Übersichtlichkeit entferne man alle Bausteine der CPU bis auf den Registerbaustein und der ALU mit dem Menüpunkt *„Optionen/Simulationsstufe/... und RAM“* in den Hintergrund (Abb. 6).

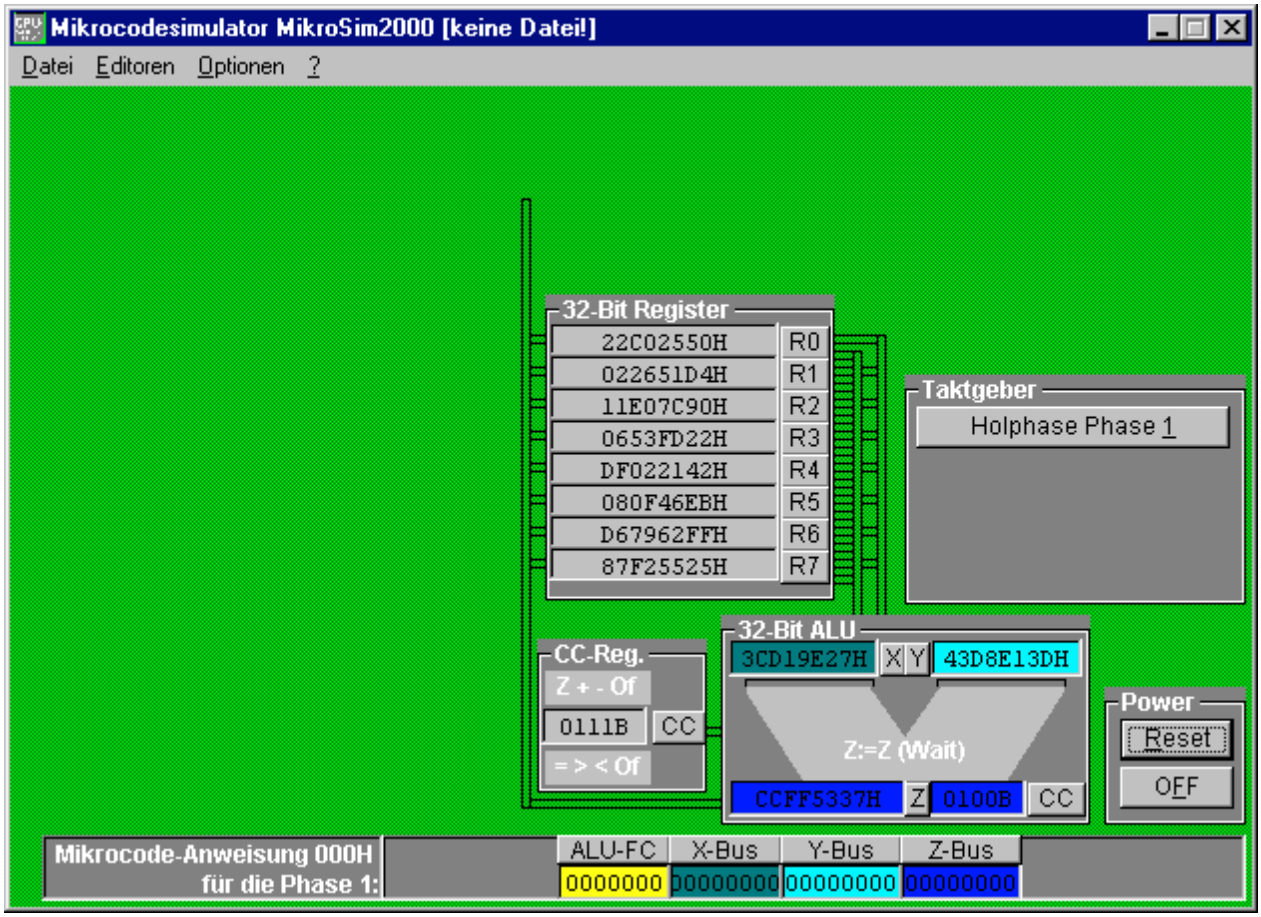

Abb. 6: Initialisierter Mikrocodesimulator im Erkundungsmodus durch Drücken auf den “Reset”-Knopf.

Um die einzelnen Bits der vier ausgewählten Bitgruppen des 49-Bit langen Mikrocodes in Hinblick auf ihre Wirkungweise zu untersuchen, wir der Simulator zunächst mit dem „Reset“-Knopf initialisiert. Wie in Abb. 6 dargestelltt tragen nun alle Register anstelle der erklärende Beschriftung wie in Abb. 4 zu sehen, nun einen Zahlenwert als Registerinhalt. Der Zahlenwert wird entweder im Binär- oder Hexadezimalformat dargestellt, je nach dem ob der Zahlenwert vorzugsweise primär als Schalterstellung oder Flag bzw. als Rechengröße oder Zahlenwert angesehen wird. Alle Registerinhalte können mit Hilfe eines „Systemzahl-Editors“ auf vier Arten dargestellt und verändert werden: In der *binären*, in der *dezimal-vorzeichenlosen*, in der *dezimal-vorzeichenbehafteten* und in der *hexadezimalen Schreibweise* (Abb. 7). Jede Zahleneingabe in einem der Zahlensysteme wird sofort in die übrigen Formate umgerechnet und im Editor dargestellt. Mit dem Verlassen des Editors mit dem „OK“-Knopf wird der Inhalt in dem Zahlenformat an das Register übergeben, welches im „Systemzahl-Editor“ grün hinterlegt ist.

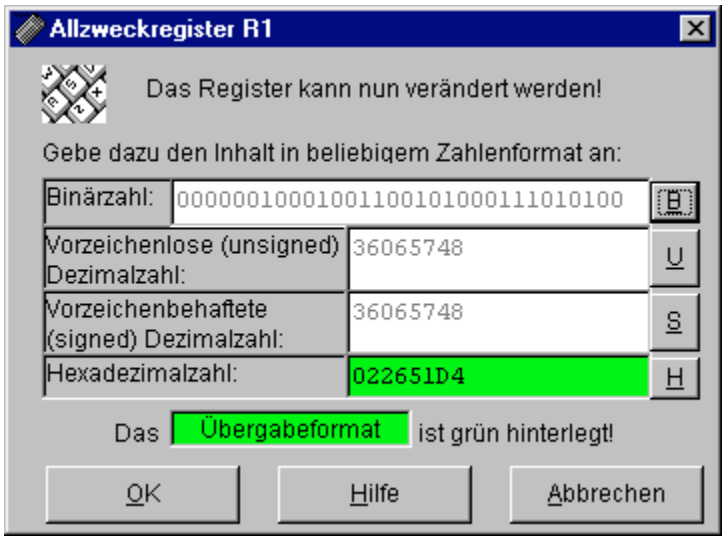

Abb. 7: Im Systemzahlen-Editor können die Registerwerte des Mikrcodesimulators in vier Darstellungsarten verändert werden.

Der Mikrocodesimulator kann im Erkundungsmodus auf diese Weise durch Modifikation von Schalterstellungen und Registerinhalten und deren Wirkungsweise auf den Simulationsablauf erkundet werden. Schreibt man z.B. in das X-Bus- und Z-Bus-Register des aktuellen Mikrocodes den Wert “01000000“ hinein, wie in Abb. 8 zu erkennen, so kann man durch Drücken des Taktgeber-Knopfes sehen, dass sich die Busleitungen in der sogenannten Holphase vom Register R0 zum X-Bus öffnen, um einen Wert in das X-Register der arithmetisch-logischen Recheneinheit ALU zu schreiben. In der Bringphase Phase3 wird über den Z-Bus ein Rechenergebnis der arithmetisch-logischen Recheneinheit ALU in das Register R0 zurückgeschrieben. Setzt man zusätzlich das 7-Bit-Register des ALU-Funktionscodes im Mikrocodesimulator auf den Wert 9, so kann man verfolgen, wie durch Drücken des Taktgebers das Register R0 schrittweise nach jedem Hol-Rechen-Bring-Zyklus um Eins erhöht wird, da nun in der Rechenphase Phase 2 das X-Register der ALU jedes Mal um Eins inkrementiert wird.

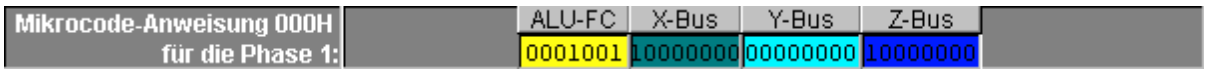

Abb. 8: Diese Mikrocode-Befehlszeile alleine bewirkt im Erkundungsmodus bereits das Inkrementieren des Registers R1.

Auf ähnliche Weise können alle Bits der 49-Bit langen Mikrocodeanweisung untersucht werden. Über den Befehlsumfang und die Abarbeitungsreihenfolge der 49-Bit langen Mikrocode geben die Hilfedateien zum Simulator näher Auskunft.

Um den Umgang mit dem Mikrocodesimulator zu üben, werden vorbereitete und dem Simulator beigelegte Beispieldateien besprochen. Eine Datei des Mikrocodesimulators öffnet man mit Hilfe des Menüpunktes „*Datei/Öffnen* ...“.

Beispielprogramm Beispiel #1:

Im Erkundungsmodus konnte nur ein Mikrocodebefehl in einem Drei-Phasentakt abgearbeitet, nämlich der, der am unteren Bildschirmrand nach Bitgruppen aufgegliedert dargestellt ist und zur Mikrocode-ROM-Adresse 000H gehört. Im Folgenden wird gezeigt, wie eine *Folge* verschiedener 49-Bit Mikrocodes diese Inkrementierung durchführen kann.

Mit dem Laden vorgefertigter Mikrocodesimulator-Programmdateien über den Menüpunkt *„Datei/Öffnen...“* wird der Erkundungsmodus verlassen. Neben der Simulationsdatei "*Beispiel1.ROM*" werden auch alle anderen zu dem Projekt dazugehörigen Dateien des Typs *„Beispiel*.*“* geladen. Mit dem Laden wird gleichzeitig die komplette CPU aufgebaut. Der Erkundungsmodus wird durch den Mikrocode-Simulationsmodus abgelöst. Aber auch in diesem Modus kann immer noch interaktiv in den Programmablauf eingegriffen werden.

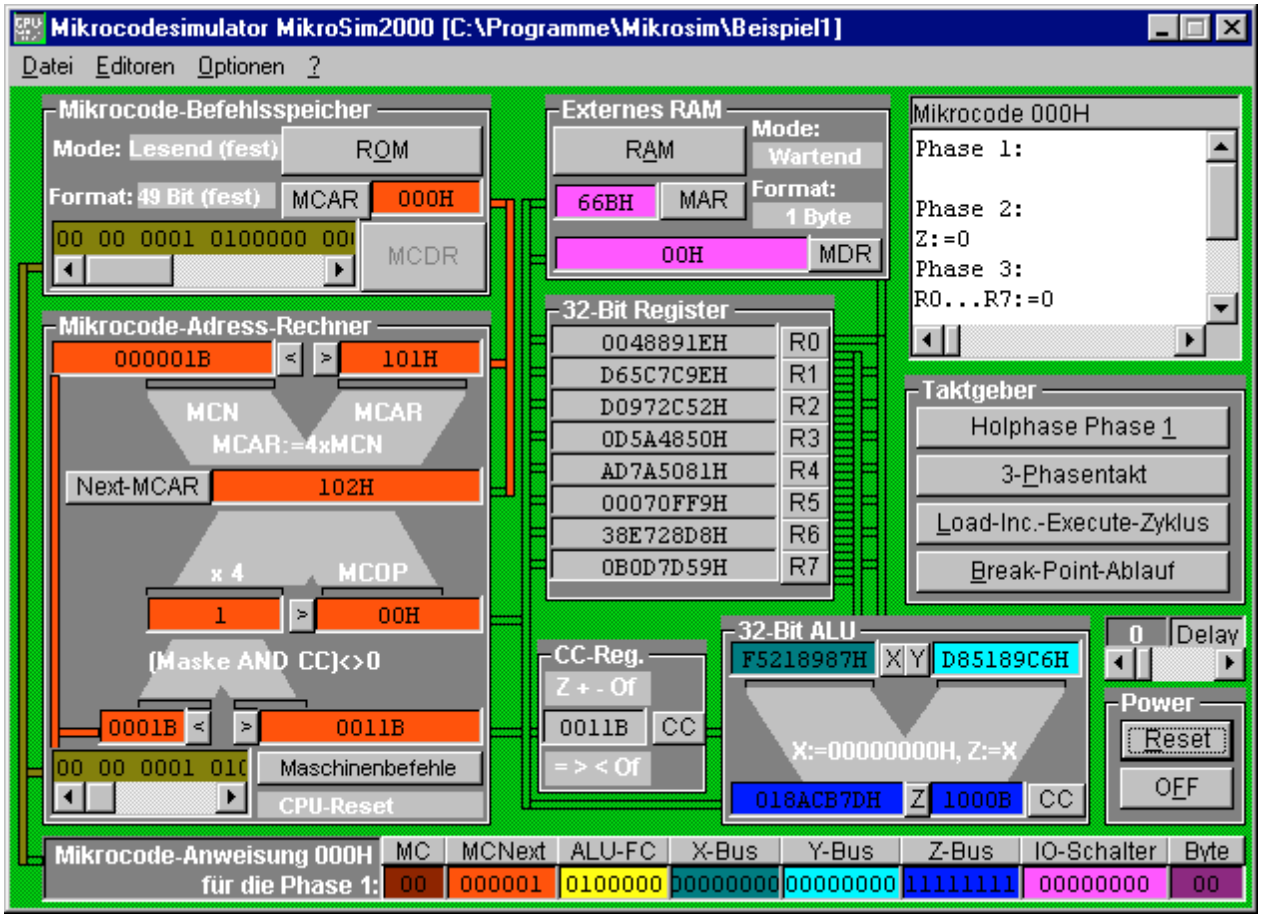

Abb. 9: Initialisierter Mikrocodesimulator mit geladener Beispieldatei "Beispiel1".

Hat man die Datei *„Beispiel1.ROM“* in den Simulator geladen, so kann man durch Drücken des "Reset"-Knopfes (Abb. 9) die CPU für den Simulationsmodus initialisieren. Sämtliche Schaltflächen der Oberfläche sind nun aktivierbar. So auch die ROM-Schaltfläche oben links im CPU-Baustein „Mikrocode-Befehlsspeicher" des Mikrocodesimulators. Mit ihm kann man sich den Inhalt der

Mikrocode-ROM-Beispieldatei "*Beispiel1.ROM*" anschauen. Nur in einige der ersten acht Zeilen sind in diesem Beispiel sind ein paar 49-Bit-Mikrocodes abgelegt (Abb. 10).

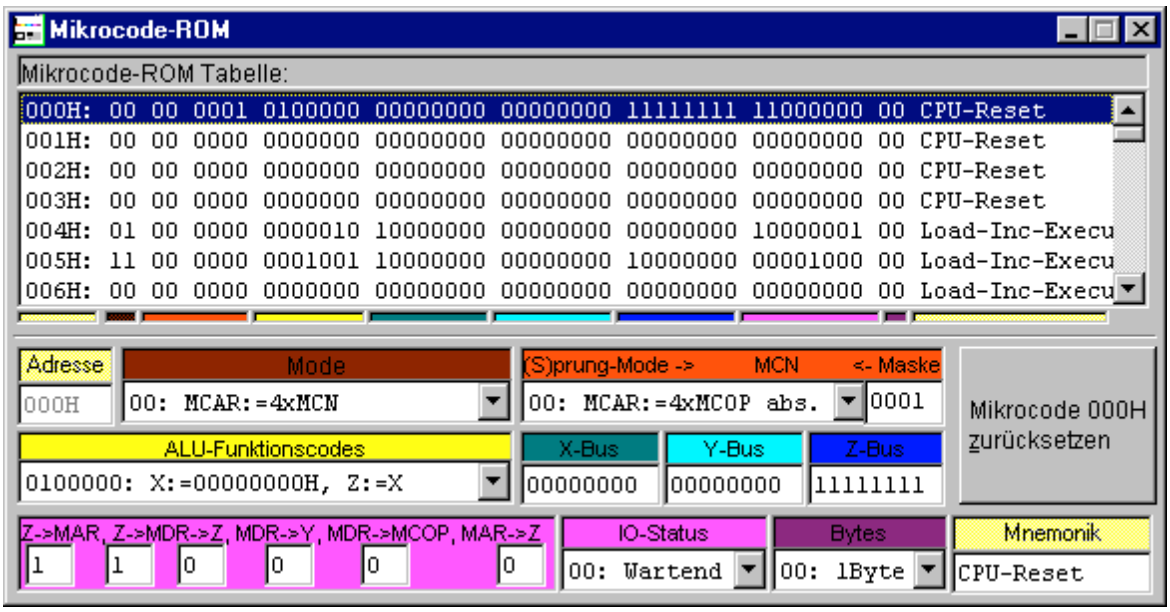

Abb. 10: Mikrocodebefehlsspeicher-Inhalt. Die Mikrocodes können mit dem Editor bitgruppenweise bearbeitet werden.

Der erste abzuarbeitende Mikrocode ist der mit der Adresse 000H. Nach einem Reset startet der Simulator stets bei der Anfangsadresse „000H“. Daher wird hier gleich die Gelegenheit genutzt, den Simulator mit einer Folge von Reset-Mikrocodes zu initialisieren und die Registerwerte auf Null zu setzen. Dies geschieht, indem in der ALU in der Rechenphase die Konstante 0 erzeugt wird und diese anschließend in der Bringphase in alle Register R0-7 geschrieben wird. Der nächste auszuführende Mikrocodebefehl steht an der Adresse 004H. Dorthin springt der Simulator, da in der Rechenphase die nächste Mikrocodeadresse die Adresse „MCNx4=4“ vorgegeben wird. Ab dort wird nun schrittweise ein Drei-Phasentakt abgearbeitet, der nach jeder Ausführung eines Mikrocodebefehls das Register A um Eins erhöht hat und den Mikrcodeprogrammzeiger auf den nächsten Befehl gesetzt hat. Bei der Mikrocodeadresse 007H generiert der Mikrocode-Adressrechner als neues Sprungziel die Adresse 004H, womit sich der Simulator in der gewünschten Endlosschleife befindet, in der das Register A nach jedem Drei-Phasentakt um Ein erhöht wird. Das geladene Mikrocodeprogramm kann entweder schrittweise in den drei Einzeltakten untersucht werden oder dreiphasentaktweise abgearbeitet werden.

Beispielprogramm Beispiel #2:

Im Beispielprogramm "Beispiel2" wird vor der Endlosaddition eine ausführlichere Initialisierung beim "Booten" vorgenommen. Neben dem Beschreiben der Register R0-7 wird der Stack-Pointer im Register R7 auf den Hexadezimalwert 800H gesetzt, was der Adresse 000H des 2 kB großen externen RAMs entspricht. Würden beispielsweise Stack-Befehle implementiert, die Daten auf den Stack legen, so müsste zuerst der Stack-Pointer um eine entsprechende Bytelänge erniedrigt werden, bevor Daten mit dieser Adresse beginnend aufsteigend an das Ende des RAMs geschrieben würde. Ab dem Segment 02H beginnt dann die Endlosschleife der Inkrementierung des vorherigen Beispiels.

In diesem Beispiel sind nun schon zwei Maschinensprachebefehle in der CPU implementiert worden, nämlich der Befehl „CPU-Reset“ und der Befehl „Inkrementiere Register A“. Aus diesem Grund ist den Mikrocode-ROM-Segmenten die Bezeichnung „CPU-Reset“ und „INC A“ gegeben worden. Eine Übersicht über die Anzahl und Lage belegter und freier Segmente im Mikrocode-ROM erhält man, wenn der „Maschinensprachebefehls-Editor“ im Menüpunkt „*Editoren*“ geöffnet wird, wie in Abb. 11 gezeigt. Nur das Segment 00H und 02H sind belegt und sind mit einem roten Feld anstelle eines grünen Feldes in der Übersichtstafel markiert. Die Segmente tragen die Bezeichnung "CPU-Reset" und "INC A".

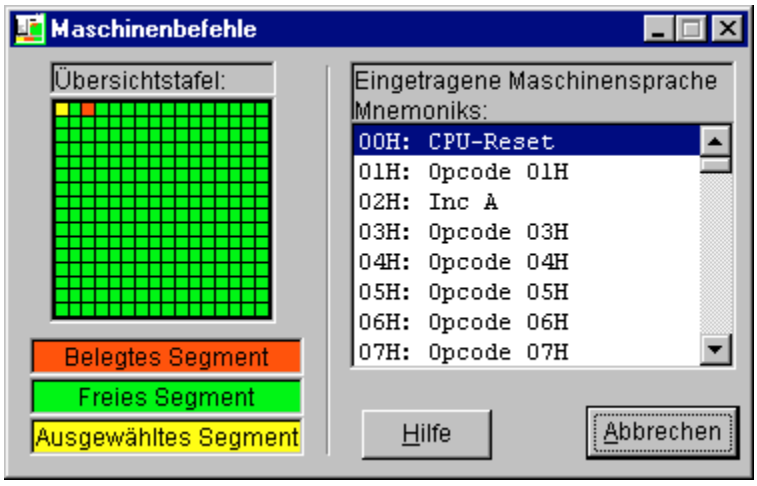

Abb. 11: Maschinensprachebefehlstabelle. Mit diesem Monitor hat man den Überblick über alle bereits verwendeten Mikrocode-ROM-Segmente.

Beispielprogramm Beispiel #3:

Im Beispielprogramm "Beispiel3" wird eine Inkrementierung und Dekrementierung des Registers R1 durch ein Maschinenspracheprogramm erledigt, welches im externen RAM auf den Adressen 000H bis 004H abgelegt ist. Das RAM kann man sich durch Drücken auf die RAM-Schaltfläche im CPU-Baustein des externen RAMs anzeigen lassen (Abb. 12).

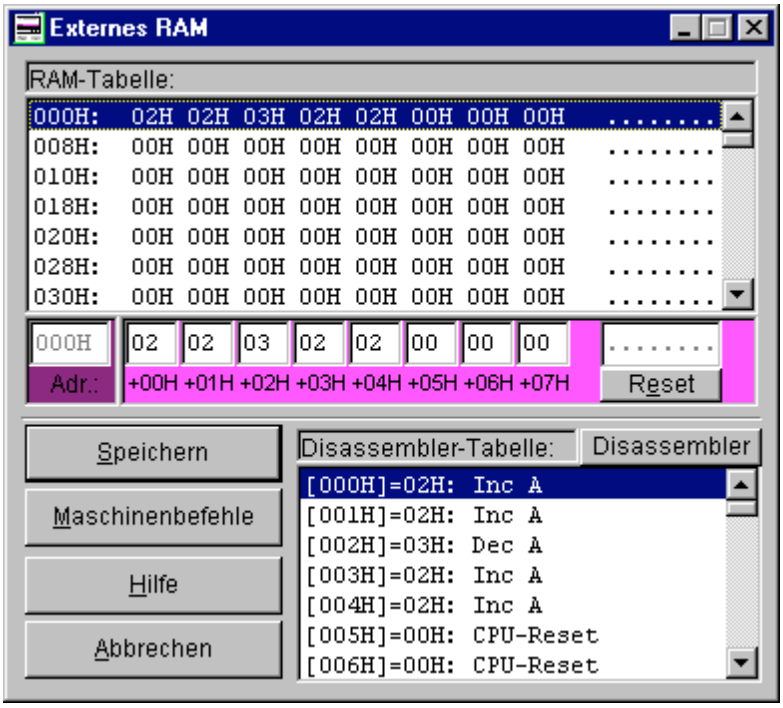

Abb. 12: Inhalt des externen RAMs des Beispielprogramms "Beispiel3". Die Maschinensprachebefehl-Sequenz 02H-02H-03H-02H-02H bewirkt eine Inkrementierung und Dekrementierung des Registers R1 (Akkumulator A).

Die Befehlssequenz 02H-02H-03H-02H-02H auf den ersten Speicherstellen des 2048 Byte großen CPU-RAMs bewirken eine zweimalige Inkrementierung des Registers R1 mittels „02H“, eine Dekrementierung mittels „03H“ und anschließend eine erneute Inkrementierung. Nach dem letzten INCA-Befehl soll das Programm auf den CPU-Reset-Befehl laufen, mit dem die CPU und somit der Programm-Ablauf initialisiert wird. Auch hier wurde nun einen Endlosschleife programmiert, deren Ablauf im Folgenden erläutert wird.

Die Maschinensprachebefehle „Reset Routine“, „INC A“ und „DEC A“ sind, wie in Abb. 13 zu erkennen, auf den Mikrocode-ROM-Segmenten „00H“, „02“ und „03“ abgelegt worden. Zusätzlich wurde ein Maschinenbefehl auf dem Mikrocode-ROM-Segment 01H als Mikrocode-Sequenz implementiert, der eine Befehlslade-Routine für Maschinensprachebefehle des externen RAMs darstellt und als „LOAD-INCREMENT-EXECUTE“-Befehl bezeichnet wird.

Die Mikrocode-Befehlssequenz des „LOAD-INCREMENT-EXECUTE“-Befehl hat die Aufgabe:

1. das Byte im externen RAM, auf welches der Program-Pointer des Registers R0 zeigt, einzulesen und als Adressen im Mikrocode-ROM zu interpretieren (**Load**-Increment-Execute-Zyklus).

2. den Program-Counter, der auf den nächsten Maschinensprachebefehl im RAM zeigt, um eins zu erhöhen (Load-**Increment**-Execute-Zyklus).

3. zur Ausführung an die Mikrocode-ROM-Adresse zu springen. Dort wird im Segment 02H letzten Endes der jeweilige Maschinenbefehl ausgeführt. Im Segment 02H wird daher der INC A-Befehl sowie im Segment 03H der DEC A-Befehl realisiert sein (Load-Increment-**Execute**-Zyklus). Der letzte Mikrocodebefehl im ROM schließt mit einem Rücksprung an den Beginn des „LOAD-INCREMENT-EXECUTE“-Segments ab, womit der Zyklus erneut durchlaufen werden kann.

Ist ein „LOAD-INCREMENT-EXECUTE“-Befehl wie im Beispiel3 gezeigt im Mikrocode-ROM implementiert, kann die Mikrocodebefehl-Abarbeitung phasenweise, dreiphasentaktweise oder als Load-Increment-Execute-Zyklus mit den entsprechenden Taktgeber-Schaltflächen gestartet werden. Welcher Maschinensprachebefehl gerade abgearbeitet wird, kann im Mikrocode-Adressrechner unten rechts unterhalb der Schaltfläche „Maschinenbefehle“ (siehe Abb. 9) abgelesen werden.

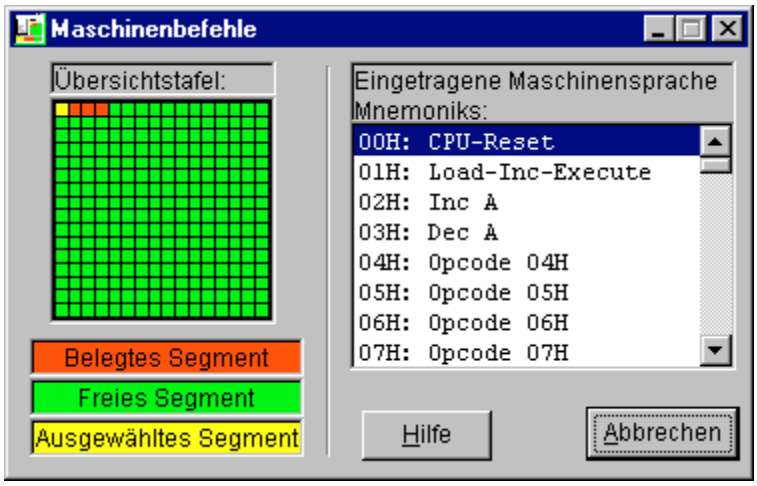

Abb. 13: Der Maschinensprachebefehls-Editor zeigt die Mikrocode-Segmente des Mikrocode-ROMs an. Die Maschienensprachebefehle entsprechen der mnemonischen Bezeichnung der ROM-Segmente.

Beispielprogramm Beispiel #4:

Im letzten Beispielprogramm "Beispiel4" wird ein Maschinenspracheprogramm zur Berechnung der Fakultät einer Zahl im Register R1 in Mikrocode-Befehlsschritten abgearbeitet. Als Beispiel wird hier die Fakultät der Zahl 4 durchgeführt, die im Register R1 durch vierfache Inkrementierung erzeugt wird, wie im Beispiel 3 bereits erläutert. Dann wird mit dem Befehl „CALL 00 10“ an die Adresse „00 10H“ im RAM gesprungen und die Rücksprungadresse im Stack abgelegt. Ab Adresse 10H beginnt das Fakultätsprogramm, welches in das Register R2 das Ergebnis der Fakultätsberechnung ablegen soll. Dieses Register wird zunächst mit einer 1 initialisiert (Fakultät der Zahl 0 ist 1). Anschließend wird der Wert von R2 mit dem Wert von R1 multipliziert und R1 um Eins dekrementiert, solange R1>0 ist. Falls R1=0 ist, wird mit einem RETURN die Stack-Adresse ausgelesen und zur weiteren Programmabarbeitung hinter den CALL-Befehl gesprungen. Dort wird der Programmablauf durch den Break-Point-Befehl FFH gestoppt. In Abb. 14 ist das im RAM abgelegte Maschinenspracheprogramm des Beispiel 4 dargestellt.

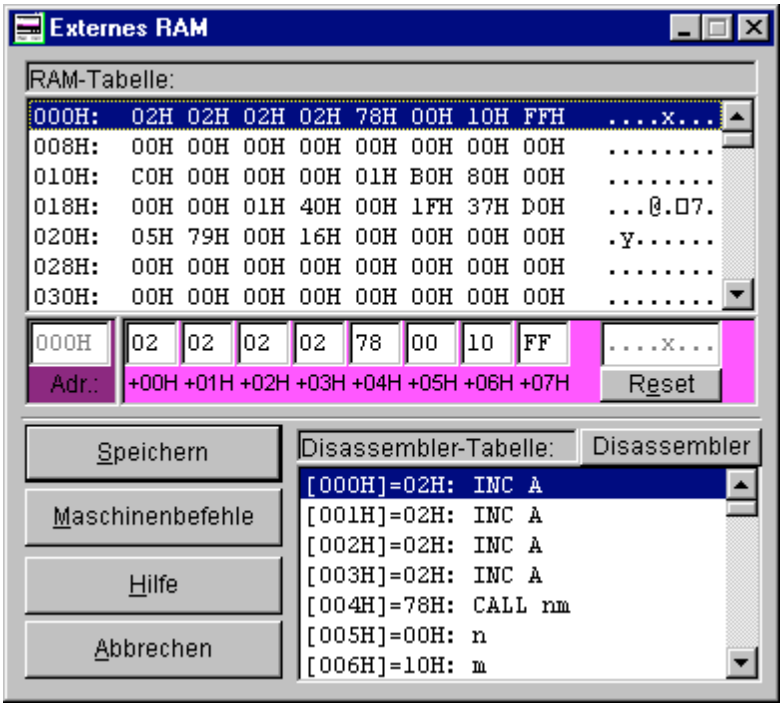

Abb. 14: Externes RAM mit der Maschinenspracheprogramm, das die Fakultät der ins Register R1 geladenen Zahl 4 berechnet und im Register R2 ablegt.

Das Programm wird gestartet, in dem der Simulator mit der „Reset“-Knopf zurückgesetzt wird und anschließend der Taktgeber „Break-Point“ gedrückt wird. Am Ende der Simulation steht im Register R2 der Hexadezimalwert 18H für die Zahl 4!=1*2*3*4=24. Das Programm hält automatisch am Break-Point-Befehl FFH an. Um den Break-Point-Modus als Taktgeberoption verwenden zu können, muß dieser Befehl wie im Beispiel 4 als Maschinenbefehl implementiert worden sein.

Das Programm kann aber auch mit einer beliebig gewählten Ausführungsgeschwindigkeit und einer beliebigen Schrittweite in der Phasenabarbeitung betrieben werden. Im Folgenden ist das kommentierte Assembler-Programm des RAM-Inhalts abgedruckt. Es wurde mit Hilfe des RAM-eigenen Disassemblers erzeugt, der mit der Schaltfläche „Dissassembler“ aktiviert werden kann.

Content of 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
```