Schnell-Einstieg in den Mikrocodesimulator MikroSim 2010 und Code-Generator MikroBAT 2010

Die folgenden Seiten sollen einen kurzen Überblick über die Tragweite der Mikrocodesimulation mit Hilfe des vorliegenden Mikrocodesimulators MikroSim 2010 und der Möglichkeit der Assemblerprogrammierung mit dem Code-Generator MikroBAT 2010 geben. Anhand 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 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 Beispiel4 wird ein mikrocodeprogrammiertes Maschinenspracheprogramm zur Berechnung der Fakultät (n!) vorgestellt.

Installation des Mikrocodesimulators MikroSim 2010

Der Mikrocodesimulator MikroSim 2010 wird über ein Programm-Installationspaket installiert. Beim Setup-Start kann als Installationssprache zwischen Deutsch und Englisch gewählt werden. Die Setup-Sprache legt gleichzeitig die Startsprache für den ersten Programmstart der zweisprachig ausgelegten Software fest.

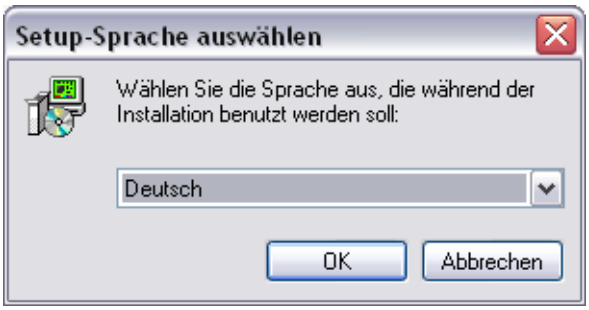

Abb. 1: Setup-Programm mit Sprachauswahl

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 Zielverzeichnises läuft die Installation mit unterbrechenden Anweisungen gesteuert 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).

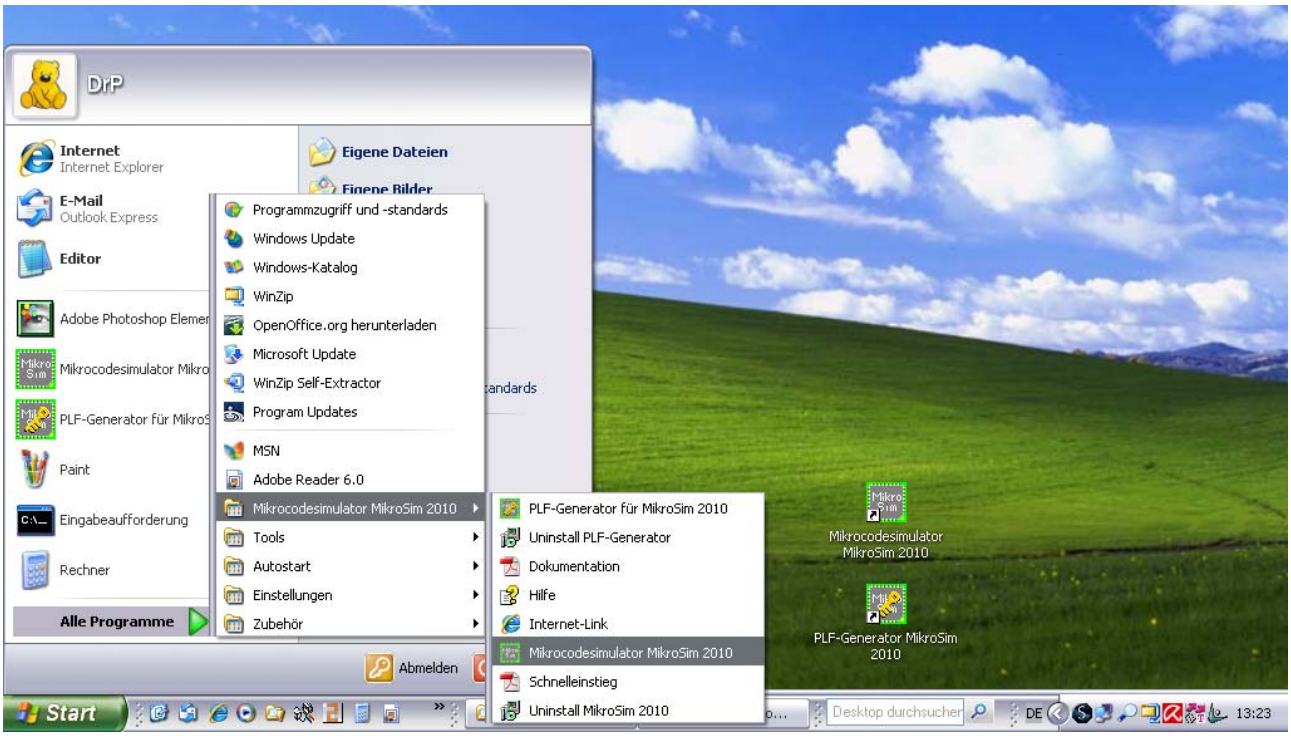

Abb. 2: Bei der Programm-Installation wird der Mikrocodesimulator in einer Programmgruppe und als Applikation in das Startmenü unter MS-Windows aufgenommen.

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 automatisiert wie gewohnt über die Systemsteuerung, Setup-Programm oder das Gruppenmenü erfolgen, keinesfalls jedoch manuell.

Sollte das Programmpaket durch ein aktualisiertes Versionsupdate ersetzt werden, empfiehlt es sich, erst die Deinstallation und anschließend eine komplette Neuinstallation durchzuführen. Auf diese Weise werden nur die veralteten und registrierten Programmkomponenten entfernt und aktualisiert. Eigene angelegte oder bei der Registrierung angelegte oder hinzugefügte Dateien wie „MikroSim.PLF“ bleiben somit erhalten.

Starten des Simulators im Erkundungsmodus

Der Mikrocodesimulator MikroSim 2010 wird durch Klicken auf das graugrünfarbene CPU-Programm-Symbol gestartet. Der Mikrocodesimulator öffnet anfangs ein grünlich, bemustertes Fenster, das eine unbestückte Platine einer aufzubauenden Applikation darstellt. Der Benutzer befindet sich in dem sogenannten *„Erkundungsmodus“*. In diesem Modus kann die CPU-Applikation schrittweise aufgebaut werden, indem unter Zuhilfenahme des Menüpunkts *„Optionen/Simulationsstufe“* nacheinander die leere Platine mit Bauteilen wie dem 32-Bit Register, der Recheneinheit (ALU), dem externen RAM, dem Mikrocode-ROM und dem Mikrocode-Adressrechner bestückt wird.

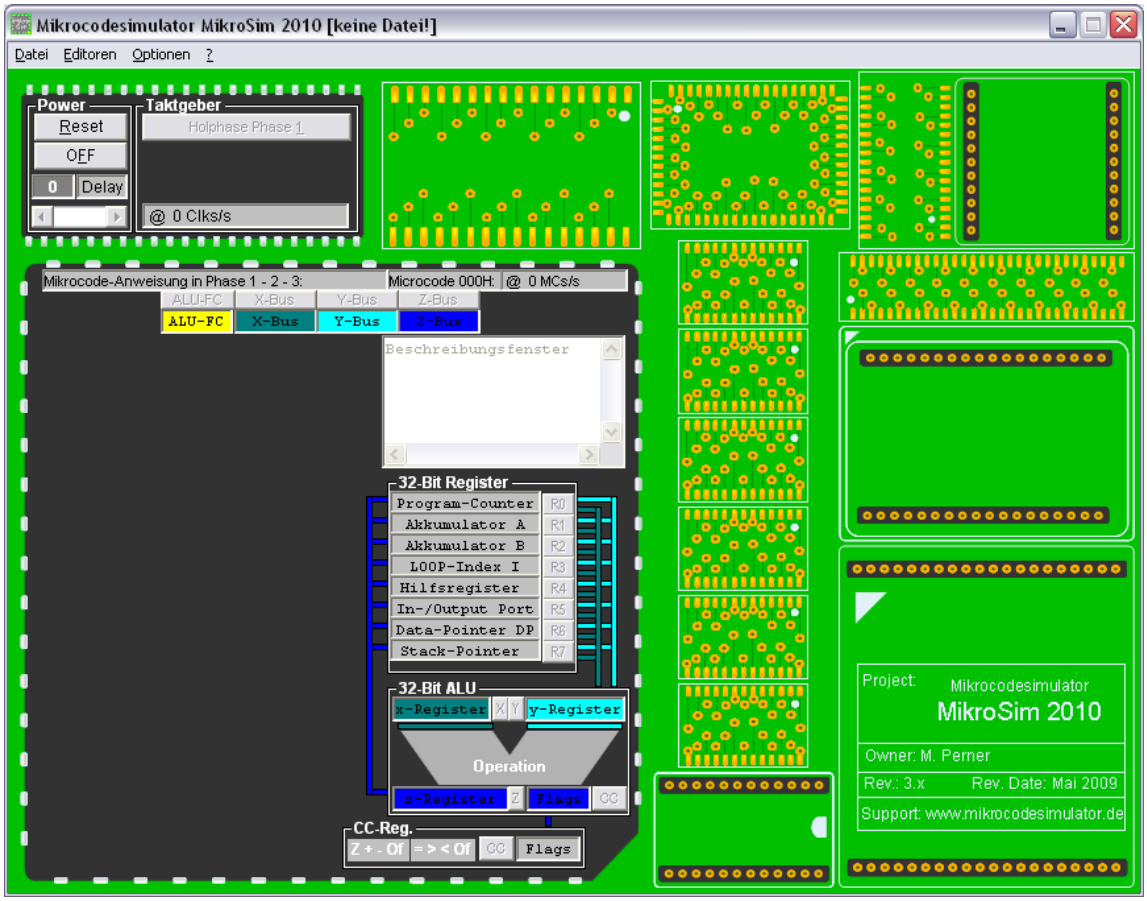

Abb. 3: Schrittweiser Aufbau des Mikrocodesimulators im Erkundungsmodus. Die farblich markierten Register und Datenleitungen werden von den entsprechend farbig hinterlegten Bits des Mikrocodes gesteuert.

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

ist die CPU komplett und kann um die angeschlossene Peripherie ergänzt werden, wie in Abb. 4 gezeigt.

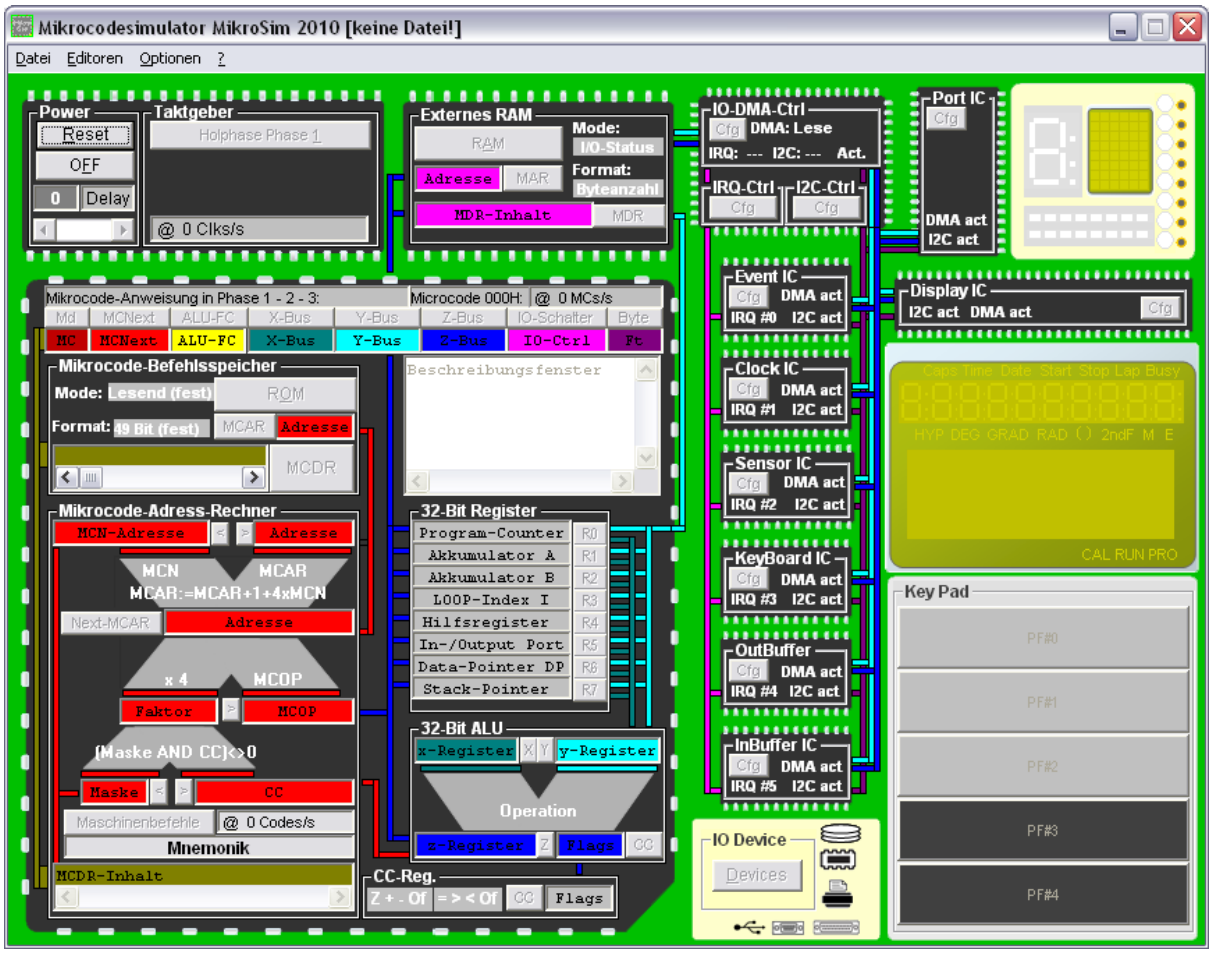

Abb. 4: Vollständig aufgebauter Mikrocodesimulator im Erkundungsmodus.

Zusätzliche Informationen über den Aufbau und die Verwendung der eingesetzten Register und Objekte kann überall dort abgerufen 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).

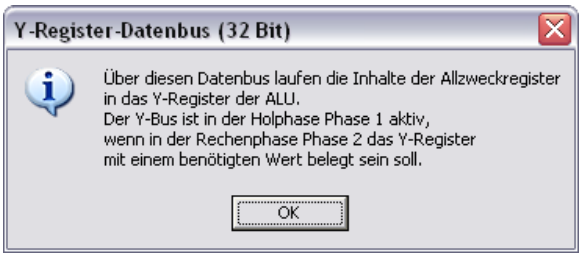

Abb. 5: Informationen zum Y-Register-Datenbus. Informationsfenster wie dieses können geöffnet werden, wenn ausgewählte Bereiche im Mikrocodesimulator angeklickt werden.

Nach dem Erkunden des Simulators sollte der Benutzer mit dem Aufbau der CPU vertraut sein. Er kann sich nun der Wirkungsweise der einzelnen Bits des am unteren Bildschirmrand dargestellten Mikrocodes widmen, welche am oberen Bausteinrand der CPU dargestellt sind. Zum Zwecke der Übersichtlichkeit entferne man alle Bausteine der CPU bis auf den Registerbaustein und der ALU mit dem Menüpunkt *„Optionen/Simulationsstufe/... und RAM“*, wie Abb. 6 gezeigt.

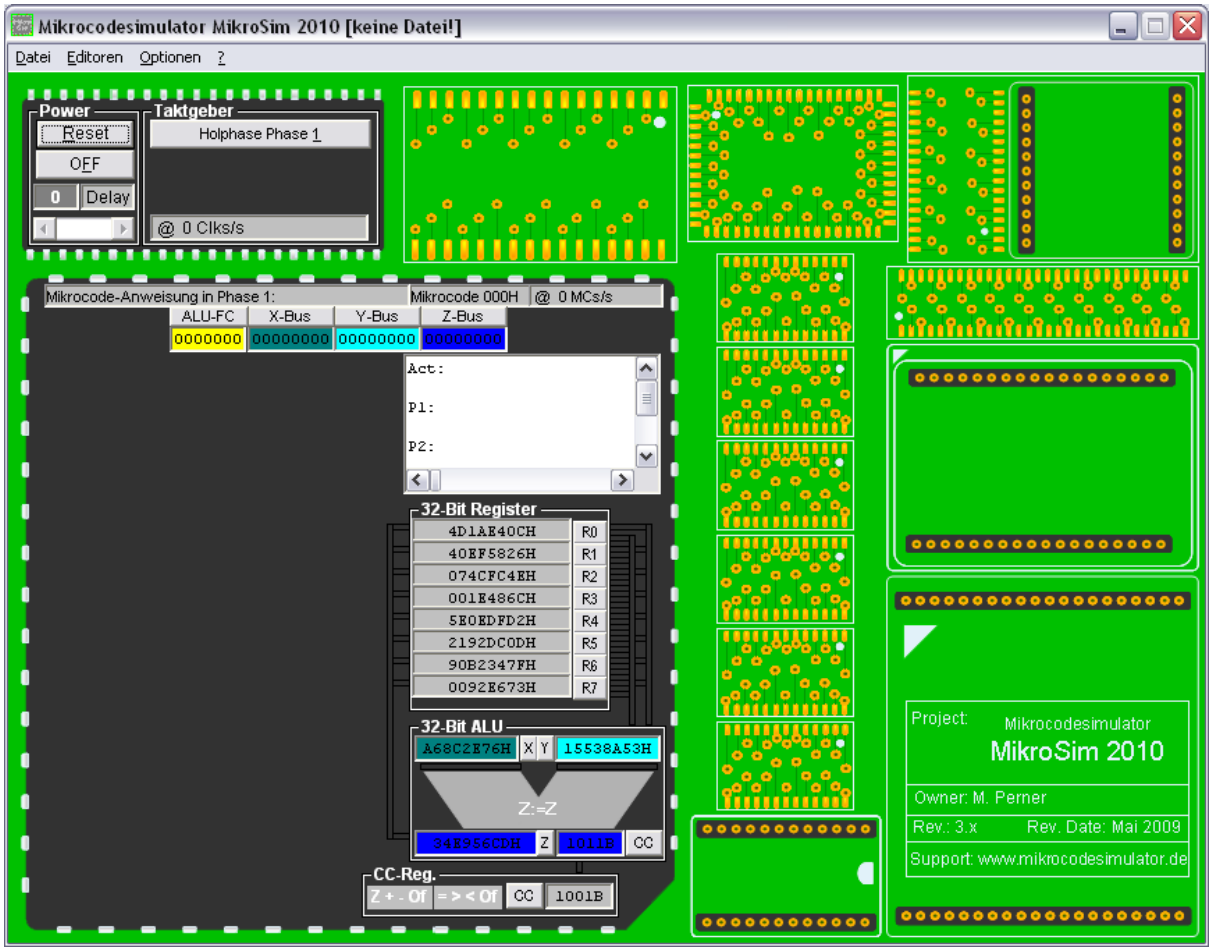

Abb. 6: Initialisierter Mikrocodesimulator im Erkundungsmodus nach Drücken auf den “Reset”-Knopf.

Um die einzelnen Bits der vier ausgewählten Bitgruppen des 49-Bit langen Mikrocodes in Hinblick auf ihre Wirkungsweise zu untersuchen, wird der Simulator zunächst mit dem „Reset“-Knopf initialisiert. Wie in Abb. 6 dargestellt tragen nun alle Register anstelle der erklärenden 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). Für 32-Bit lange Register ist eine Eingabe als Gleitkommazahl mit einfacher Genauigkeit ebenso möglich. 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.

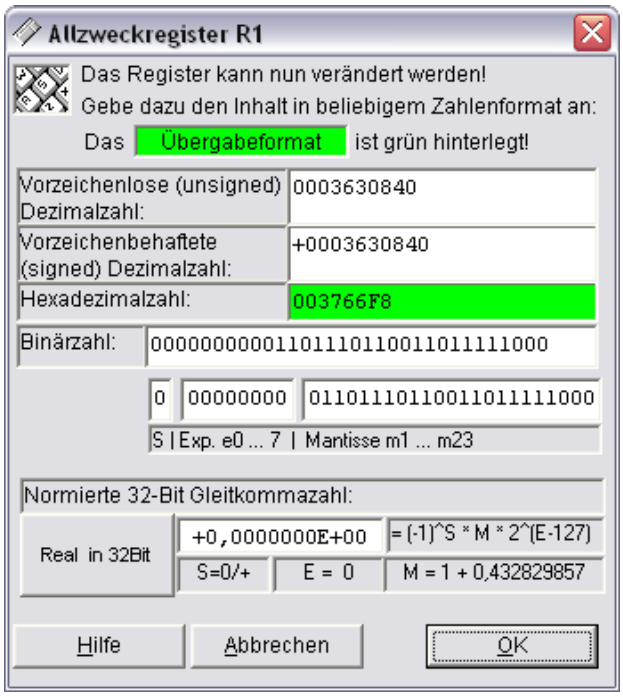

Abb. 7: Im Systemzahlen-Editor können die Registerwerte des Mikrcodesimulators bis zu fünf Darstellungsarten verändert werden.

Der Mikrocodesimulator kann im Erkundungsmodus auf diese Weise durch Modifikation von Schalterstellungen und Registerinhalten in seinem Simulationsablauf erkundet werden. Schreibt man z. B. in das Y-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 R1 zum Y-Bus öffnen, um einen Wert in das Y-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 R1 zurückgeschrieben. Setzt man zusätzlich das 7-Bit-Register des ALU-Funktionscodes im Mikrocodesimulator auf den Wert 9, so kann durch Drücken des Taktgebers das Register R1 schrittweise nach jedem Hol-Rechen-Bring-Zyklus um Eins erhöht werden, da nun in der Rechenphase Phase 2 das Y-Register der ALU stets um Eins inkrementiert wird.

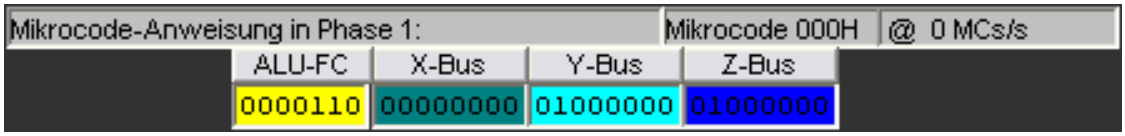

Abb. 8: Diese Mikrocode-Befehlszeile alleine bewirkt im Erkundungsmodus bereits das kontinuierliche Inkrementieren des Registers R1.

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, die über den Menüpunkt „*Datei/Öffnen ...*“ geladen werden können.

© Dr. Martin Perner, Josef-Sterr-Str. 13, D-81377 München, Deutschland
URL: http://www.mikrocodesimulator.de
E-Mail: info@mikrocodesimulator.de

Beispielprogramm Beispiel #1:

Im Erkundungsmodus kann nur ein Mikrocodebefehl in einem Drei-Phasentakt abgearbeitet werden, nämlich der am oberen CPU-Bausteinrand farblich nach Bitgruppen aufgegliederte, der in diesem Modus 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 "*Demo001.ROM*" werden auch alle anderen zu dem Projekt dazugehörigen Dateien des Typs „*Demo001.**“ 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.

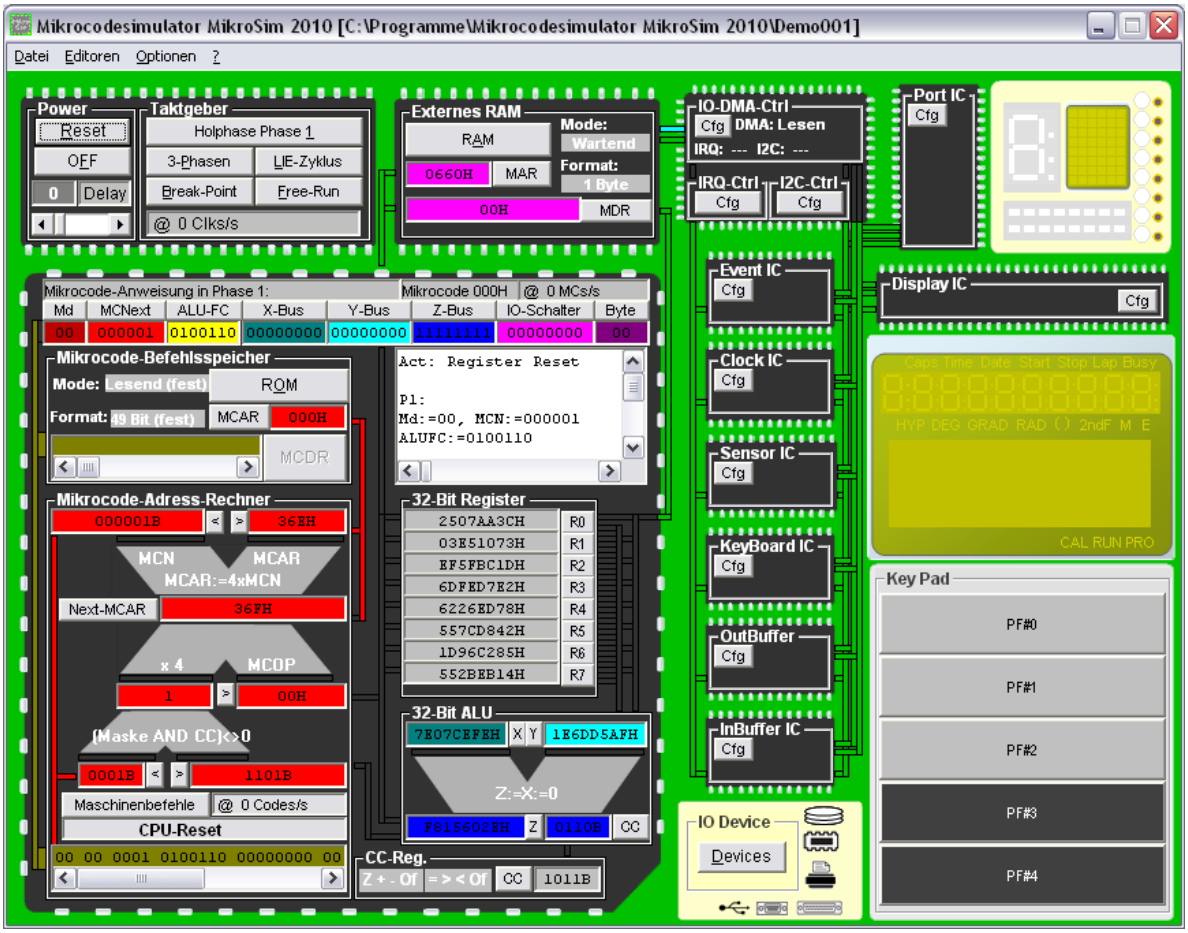

Abb. 9: Initialisierter Mikrocodesimulator mit geladener Beispieldatei "Demo001".

Hat man die Datei „*Demo001.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 "*Demo001.ROM*" anschauen. Nur in einigen der ersten acht Zeilen dieses Beispiels sind 49-Bit-Mikrocodes abgelegt (Abb. 10).

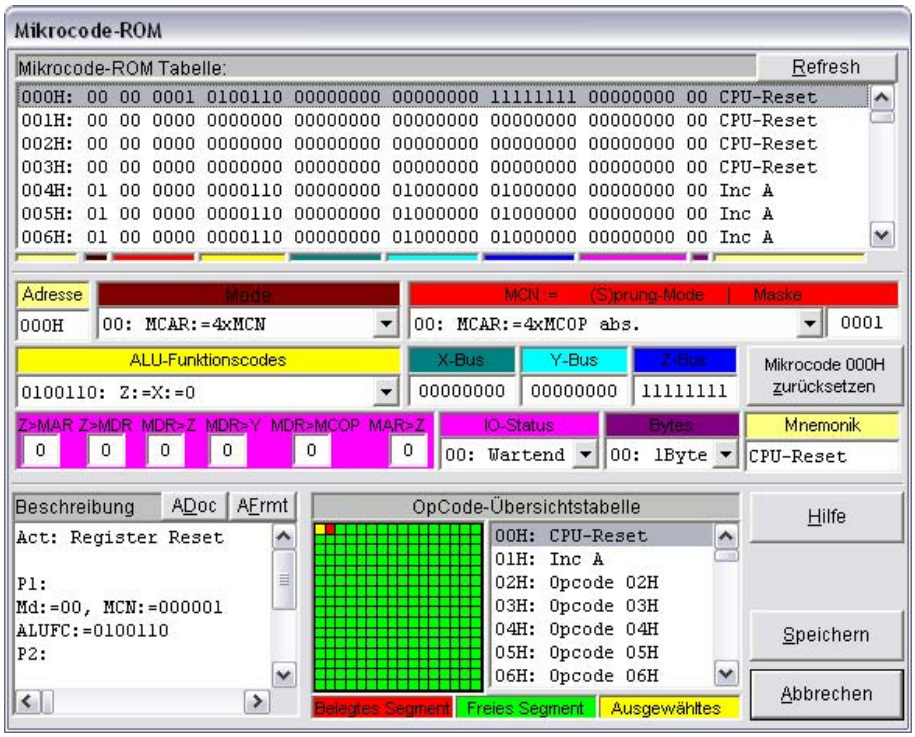

Abb. 10: Mikrocodebefehlsspeicher-Inhalt. Die Mikrocodes können mit dem Editor bitgruppenweise bearbeitet werden.

Der erste abzuarbeitende Mikrocode ist der mit der Adresse 000H. Nach einem Reset startet der Simulator stets bei der Anfangsadresse „000H“. Daher wird hier sogleich 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 Mikrocodeprogrammzeiger 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 Eins erhöht wird. Das geladene Mikrocodeprogramm kann entweder schrittweise in den drei Einzeltakten untersucht werden oder dreiphasentaktweise abgearbeitet werden.

Beispielprogramm Beispiel #2:

Im Beispielprogramm "Demo002" 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. 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, sind nur das Segment 00H und 02H belegt. In der Übersichtstafel sind diese als rote Felder anstelle grüner Felder dargestellt. Die Segmente tragen die Bezeichnung "CPU-Reset" und "INC A".

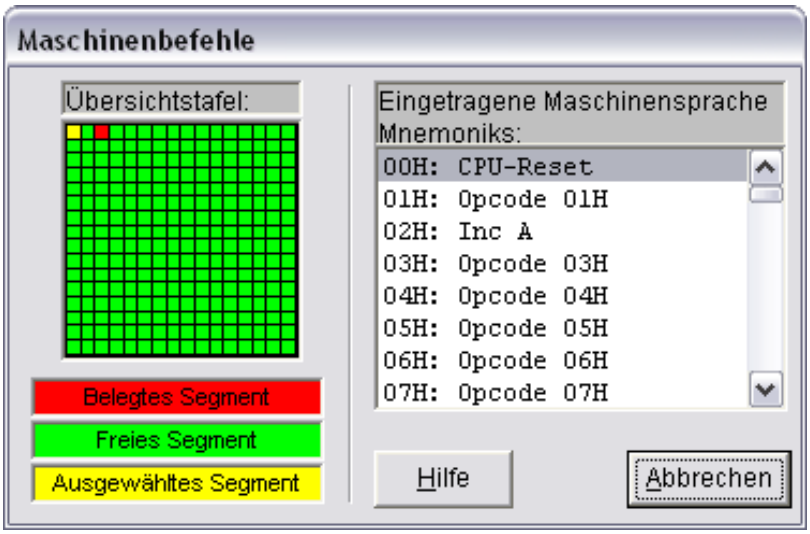

Abb. 11: Maschinensprachebefehlstabelle. Mit diesem Monitor hat man den Überblick über alle bereits verwendeten Mikrocode-ROM-Segmente.

Beispielprogramm Beispiel #3:

Im Beispielprogramm "Demo003" 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).

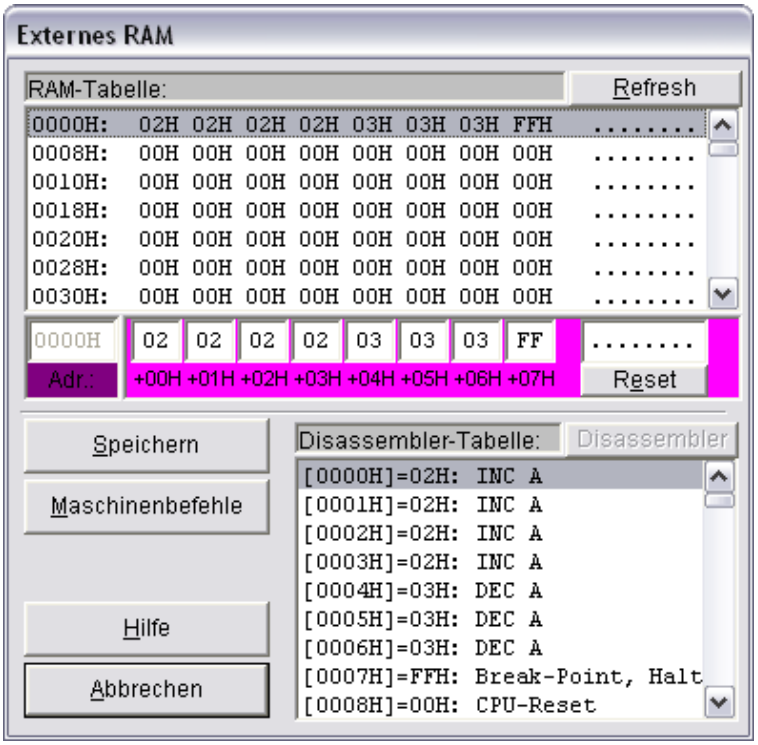

Abb. 12: Inhalt des externen RAMs des Beispielprogramms "Demo003". Die Maschinensprachebefehl-Sequenz 02H-02H-02H-02H-03H-03H-03H bewirkt eine Inkrementierung und Dekrementierung des Registers R1 (Akkumulator A).

Die Befehlssequenz 02H-02H-02H-02H-03H-03H-03H auf den ersten Speicherstellen des 16 kByte großen CPU-RAMs bewirken eine viermalige Inkrementierung des Registers R1 mittels „02H“, eine dreimalige Dekrementierung mittels „03H“. Nach dem letzten DECA-Befehl wird der Maschinenbefehl „FFH“ erreicht, der eine Programmstopp im Break-Point-Modus ermöglicht. Wird allerdings über diesen Befehl mittels Einzelschritten die Abarbeitung weiter fortgesetzt, so läuft der PC-Counter den CPU-Reset-Befehl „00H“ an, mit dem die CPU und somit der Programm-Ablauf initialisiert wird. Auch hier wurde nun eine Endlosschleife programmiert, deren Ablauf im Folgenden erläutert wird.

Die Maschinensprachebefehle „CPU-Reset“, „INC A“ und „DEC A“ sind, wie in Abb. 13 zu erkennen, auf den Mikrocode-ROM-Segmenten „00H“, „02H“ und „03H“ 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 Beispiel #3 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.

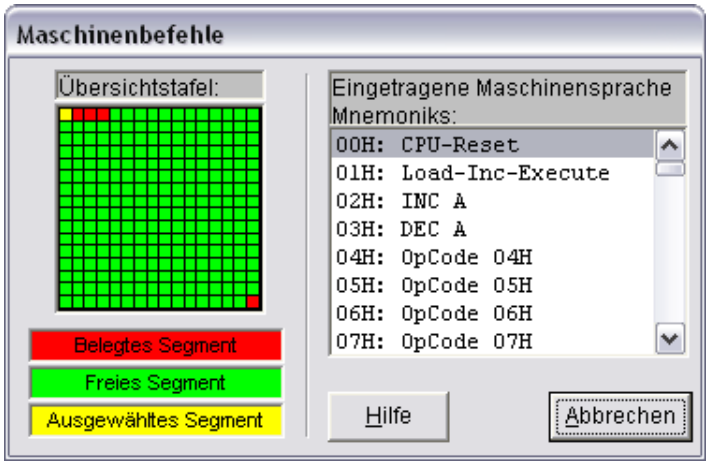

Abb. 13: Der Maschinensprachebefehls-Editor zeigt die Mikrocode-Segmente des Mikrocode-ROMs an. Die Maschienensprachebefehle entsprechen der mnemonischen Bezeichnung der ROM-Segmente.

Beispielprogramm Beispiel #4:

Im Letzten zum Einstieg besprochen Beispielprogramm "Demo006" 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 berechnet, die im Register R1 erzeugt wird, indem die Konstante 03H auf den Stack gelegt wird und nach dem Zurückholen um Eins inkrementiert wird. Anschließend 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 Beispiels 4 dargestellt.

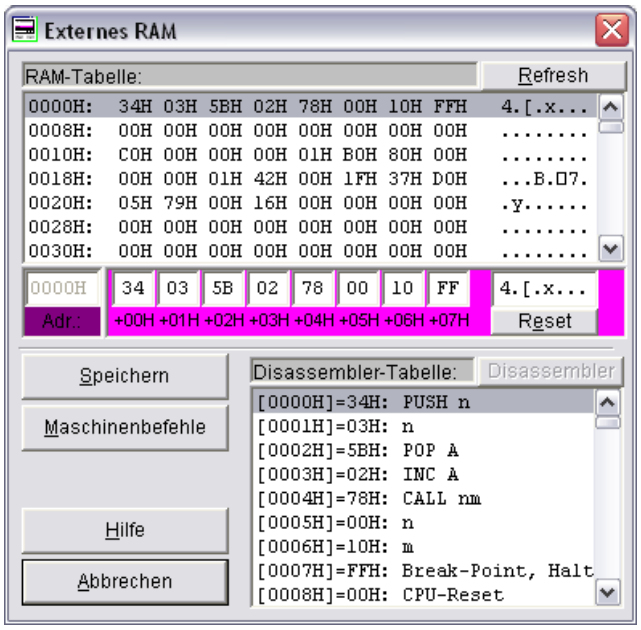

Abb. 14: Externes RAM mit dem Maschinenspracheprogramm, das die Fakultät der ins Register R1 geladenen Zahl 4 berechnet und im Register R2 ablegt.

Das Programm wird gestartet, indem der Simulator mit dem „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 „Disassembler“ aktiviert werden kann.

Inhalt des externen RAMs:

```
0000H: 34H 03H 5BH 02H 78H 00H 10H FFH 00H 00H 00H 00H 00H 00H 00H 00H
0010H: C0H 00H 00H 00H 01H B0H 80H 00H 00H 00H 01H 42H 00H 1FH 37H D0H
0020H: 05H 79H 00H 16H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
0030H: 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
0040H: 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
```

Disassembler-Tabelle:

```
[0000H]=34H: PUSH n
[0001H]=03H: n
[0002H]=5BH: POP A
[0003H]=02H: INC A
[0004H]=78H: CALL nm
[0005H]=00H: n
[0006H]=10H: m
[0007H]=FFH: Break-Point, Halt
[0008H]=00H: CPU-Reset
[0009H]=00H: CPU-Reset
[000AH]=00H: CPU-Reset
[000BH]=00H: CPU-Reset
[000CH]=00H: CPU-Reset
[000DH]=00H: CPU-Reset
[000EH]=00H: CPU-Reset
[000FH]=00H: CPU-Reset
[0010H]=C0H: MOVB, nmlk
[0011H]=00H: n
[0012H]=00H: m
[0013H]=00H: l
[0014H]=01H: k
[0015H]=B0H: MOVAux, A
[0016H]=80H: CMPAux, nmlk
[0017H]=00H: n
[0018H]=00H: m
[0019H]=00H: l
[001AH]=01H: k
[001BH]=42H: JL nm
[001CH]=00H: n
[001DH]=1FH: m
[001EH]=37H: RETURN
[001FH]=D0H: MULB, Aux
[0020H]=05H: DEC Aux
[0021H]=79H: JP nm
[0022H]=00H: n
[0023H]=16H: m
[0024H]=00H: CPU-Reset
```

© Dr. Martin Perner, Josef-Sterr-Str. 13, D-81377 München, Deutschland
URL: http://www.mikrocodesimulator.de
E-Mail: info@mikrocodesimulator.de

Ein Software-Produkt von

Installation des Code-Generators MikroBAT 2010

Der Code-Generator MikroBAT ist ein Zusatzprogramm für den Mikrocodesimulator. MikroBAT ist ein „Basic Assembler Tool“ für MikroSim, mit dem es möglich ist, Binär-Code aus Assembler-Sprach-Skripts zu erzeugen. Dieses Zusatzprogramm benötigt grundsätzlich die gleichen Treiber wie die der Simulator und setzt auf der Installation von MikroSim auf. Das Programm kann aber auch ohne den Simulator installiert und zur Binärcodeerzeugung benutzt werden. Da das Zusatzprogramm in der Dokumentation und dem Hilfe-System zu MikroSim integriert ist, empfiehlt es sich das Tool in der Programmgruppe von MikroSim zu installieren. Die bei der Installation von MikroSim bereits angelegten Dateien werden von MikroBAT ggf. aktualisiert und gemeinsam verwendet. Die bei der Installation getroffene Installations- und Ausführungssprache wird gleichzeitig für MikroSim und MikroBAT festgelegt.

Mit MikroBAT erzeugte Code-Dateien können nur im Simulator zur Mikrocodesimulation herangezogen werden. Die zur Code-Erzeugung verwendeten Dateien von MikroSim können in MikroBAT verwendet werden, wenn die von MikroSim signierten RAM- und ROM-Dateien zu dem selben Gruppenlizenznehmer wie dem von MikroBAT gehören. In diesem Fall kann auch der mit MikroBAT erzeugte Binärcode als RAM-Datei signiert abgespeichert und von MikroSim verwendet werden.

© Dr. Martin Perner, Josef-Sterr-Str. 13, D-81377 München, Deutschland
URL: http://www.mikrocodesimulator.de
E-Mail: info@mikrocodesimulator.de

Start und Betrieb des Code-Generators

Das Programm MikroBAT ist, wie der Name bereits andeutet, ein grundsätzliches Assembler-Sprach-Tool, mit dem es ermöglicht wird, Assemblerprogramme für MikroSim zu schreiben, diese in Binärcode umzuwandeln und anschließend in die RAM-Datei des externen RAMs zu übertragen.

Damit MikroBAT die Maschinensprachebefehle erkennt, die im Assemblerprogramm-Quelltext verwendet werden sollen, muss zunächst eine Mikrocode-ROM-Datei über den Menüpunkt „Datei“ geladen werden, in der die Segmentbeschreibung der verwendeten OpCodes enthalten sind. Die mit der ROM-Datei verbundene RAM-Datei wird als Ziel-Datei für den erzeugten Binärcode bereitgestellt. Beim Öffnen einer Mikrocode-ROM-Datei werden die erkannten OpCodes in der OpCode-Liste aufgeführt. Damit OpCodes als einzelne Token erkannt werden können, werden Leerzeichen, Bindestrich und Komma in der Segmentbeschreibung durch das Unterstrich-Zeichen „_“ ersetzt.

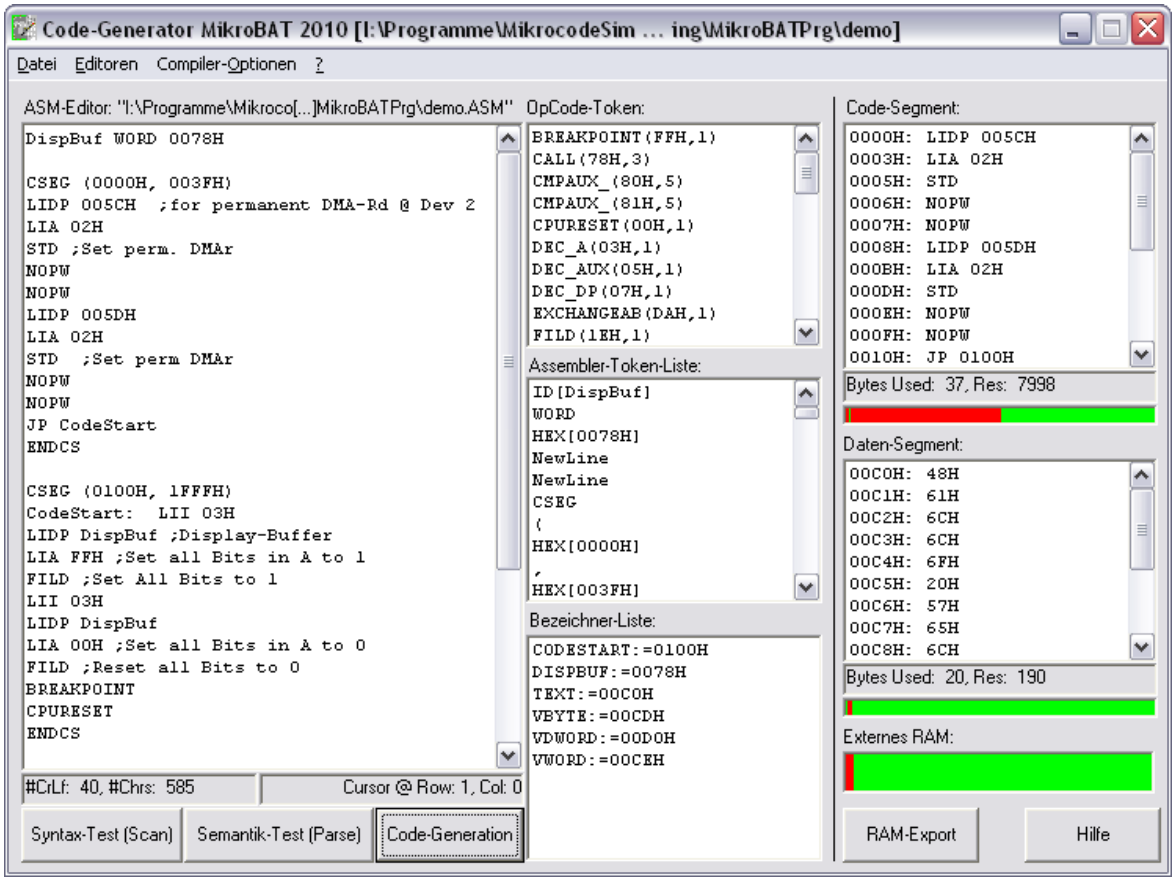

Abb. 15: Der Code-Generator MikroBAT nach erfolgreicher Compilation.

Das eigentliche Assembler-Programm befindet sich in einer Text-Datei mit der Endung ASM. Diese kann ebenso über den Menüpunkt „Datei“ in den ASM-Editor geladen werden. In dem ASM-Editor-Textfeld erfolgen die üblichen Programmierungen, die über MikroBAT abgespeichert werden können, oder die mittels Ausschneiden-und-Einfügen direkt in das Textfeld eingefügt,

ausgeschnitten oder kopiert werden können. Das fertig gestellte ASM-Programm wird für anschließende Binärcode-Erzeugung verwendet. Die Erzeugung des Binärcodes aus der Assembler-Text erfolgt in den drei Prozessierungsschritten in folgender Reihenfolge:

1. **Token-Erzeugung und Syntaxüberprüfung** im sogenannten Scan-Vorgang
2. **Semantik-Test mit Bezeichner- und Grammatiküberprüfung**
   im sogenannten Parse-Vorgang.
3. **Codegeneration und Binärcodeerzeugung** für Code-und Daten-Segment
   im Compilier-Vorgang.

Jeder dieser drei Prozesse ist eine Schaltfläche im unteren Teil des Assembler-Editors zugeordnet, die ggf. vorangehende Prozessschritte und den Vorgang an sich und ausführt. Eine fehlerfreie Code-Erzeugung umfasst die Schritte 1, 2 und 3 und wird mit einer Meldung bestätigt.

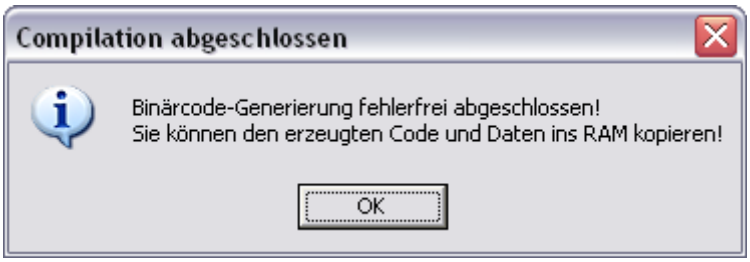

Abb. 16: Meldung einer erfolgreichen Code-Erzeugung nach vollständiger Compilation.

Im Falle von Fehlern bei der Codeerzeugung erscheinen Hinweisfenster, die auf Probleme bei der Semantik-Analyse oder der Code-Erzeugung hindeuten. Mit den Hinweisen kann der Nutzer mit der OpCode-Token-Liste und der aufgestellten Bezeichnerliste feststellen, welche Assembler-Token nicht korrekt erkannt worden sind.

Nach erfolgreicher Code-Erzeugung ist die Code- und Daten-Segment-Liste auf der rechten Hälfte von MikroBAT mit Einträgen gefüllt. Jeder Eintrag im Code-Segment besteht aus einer Adresse, dem OpCode und ggf. einem 1,2,3 oder 4-Byte Hex-Wert als OpCode-Argument. Die Einträge der Datensegment-Liste bestehen aus der Adresse und einem Byte-, Byte-Wort- oder Doppel-Wort-Eintrag. Code und Daten können durch Betätigen der RAM-Export-Schaltfläche an die ermittelten Stellen im externen RAM exportiert werden. Das abgeänderte RAM muss abschließend gespeichert werden, wenn die neue RAM-Datei im Mikrocodesimulator verwendet werden soll.

Anleitung und Beispiel zur Code-Erzeugung

Hinweise zur Mnemonik-Bezeichnung

- Leerzeichen, Komma oder Bindestriche in der Segmentbeschreibung, die als Mnemonik verwendet werden, werden in Unterstriche umgewandelt. Die Schreibweise der verwendeten Mnemonics in der OpCode-Token-Tabelle ist maßgeblich für die Erkennung gültiger OpCodes im Assembler-Editor.
- Die Schreibweise ist schreibungsunabhängig (Case-Insensitiv).
- Endungen in der Segmentbezeichnung [_|#|@][N|NM|NML|NMLK] werden als Platzhalter für das dem Opcode folgende Byte-Argument interpretiert. Der OpCode wird entsprechend als 1-, 2-, 3-, 4- oder 5-Byte-Befehl erkannt und beim Semantik-Test und Code-Erzeugung berücksichtigt.

Regeln zur Erstellung von Assembler-Text-Dateien

In Stichpunkten sind alle wesentlichen Sprachmerkmale der von MikroBAT unterstützten Assembler-Programmierung zusammengestellt. Auf eine exakte Angabe der Sprachunterstützung in Form regulärer Ausdrücke und Produktionen in Backus-Naur-Form (BNF) wird hier verzichtet. Ein Code-Beispiel mit der exemplarischen Ausführung der grundsätzlichen Sprachmerkmalen ist im folgende Assembler-Programmierbeispiel angefügt.

- Kommentare sind erlaubt, werden mit „;“(Semikolon) oder „’“(Apostroph) eingeleitet und gelten bis Zeilenende.
- Konstanten-Definition über „Define [Byte|Word|DWord]“ wird unterstützt.
- Erlaubte Mnemonics und die Zuordnung des Byte-Codes und deren Byte-Argumentgröße werden aus der Segmentbeschreibung ermittelt.
- Byte-Argument der OpCodes können Konstantenzuweisungen sein
- Direktive CSEG bzw. DSEG ermöglicht Aufteilung in Code- und Daten-Segmente in feste Adressbereiche.
- Segment bekommt direkte Start-Adresse, End-Adresse. Diese dürfen sich nicht gegenseitig überschneiden und müssen ausreichend groß für den Assembler-Code und die Daten sein. Die Segmente enden mit der Direktiven ENDCS bzw. ENDDS.
- Label als Adressmerker sind erlaubt
- Argumente nur als Hexadezimal-Konstante erlaubt
- Datendefinition im Datensegment mit „DB, DW, DD“ möglich. Bezeichner wird als Adress-Pointer verstanden. DB erlaubt auch String-Definition „Text“.

© Dr. Martin Perner, Josef-Sterr-Str. 13, D-81377 München, Deutschland
URL: http://www.mikrocodesimulator.de
E-Mail: info@mikrocodesimulator.de

Ein Software-Produkt von

Assembler-Programmierbeispiel

```
DispBuf WORD 0078H

CSEG (0000H, 003FH)
  LIDP 005CH ;for permanent DMA-Rd @ Dev 2
  LIA 02H
  STD        ;Set perm. DMAr
  NOPW
  NOPW
  LIDP 005DH
  LIA 02H
  STD        ;Set perm. DMAr
  NOPW
  NOPW
  JP CodeStart
ENDCS

CSEG (0100H, 1FFFH)
CodeStart:  LII 03H
            LIDP DispBuf ;Display-Buffer
            LIA FFH ;Set all Bits in A to 1
            FILD ;Set All Bits to 1
            LII 03H
            LIDP DispBuf
            LIA 00H ;Set all Bits in A to 0
            FILD ;Reset all Bits to 0
            BREAKPOINT
            CPURESET
ENDCS

DSEG (0040H, 00BFH)
ENDDS

DSEG (00C0H, 00FFH)
 Text DB „Hallo Welt!“
 vByte DB FCH
 vWord DW CAFEH
 vDWord DD CAFEF00DH
ENDDS
```

Tipps und Hinweise bei Installationsproblemen

Obwohl der Simulator im getesteten Installations-Setup ausgeliefert wird, kann es auf dem einen oder anderen System zu Installationsproblemen kommen. Der Simulator ist auf Windows 2000, XP, Vista und 7 installierbar und betreibbar. Dennoch kann es sein, dass Ihnen bei der Installation und dem Betrieb Probleme und Fragen aufkommen. Im Folgenden werden auf bekannte Probleme und Fragen Tipps und Hinweise zum erfolgreichen Simulatoreinsatz gegeben. Sollten Ihnen die folgenden Hinweise keine Lösung zum erfolgreichen Einsatz von MikroSim bereithalten, so können Sie Ihre Fragen zu Installation und Betrieb gerne an info@mikrocodesimulator.de senden.

Installations-Setup-Paket ist korrupt

Beim Herunterladen des ca. 20 MByte großen Installationspakets aus dem Internet kann es vorkommen, dass dieses aufgrund der Datenübertragung beschädigt wird bzw. Daten verloren gehen. In diesem Fall stimmt die Prüfsumme des Installationspakets nicht mit dem des Inhalts überein und die Installation wird mit einer Fehlermeldung gestoppt. Bitte versuchen Sie, das Installations-Setup-Paket nochmals herunterzuladen. Für die fehlerfreie Verteilung des Setup-Programms empfiehlt es sich, eine nicht beschädigte Kopie des Setups über CD-ROM oder USB-Stick zu verteilen.

Laufzeitfehler mit MikroSim.OCX

Diese Fehlermeldung erscheint kurz vor dem öffnen des Hauptfensters, wenn MikroSim.exe die Datei MikroSim.ocx nicht findet, diese nicht korrekt registriert ist oder die erwartete Revision nicht vorgefunden wird. Im Setup-Paket von MikroSim befindet sich zwar die aktuelle und zu MikroSim.exe passenden OCX-Datei, doch es kann vorkommen, dass bei Installation und Deinstallation die Datei MikroSim.ocx nicht korrekt entfernt wird, nicht korrekt ausgetauscht oder im System unvollständig registriert wird. In diesem Fall empfiehlt es sich, nach der OCX-Datei zu suchen, diese zu löschen und MikroSim erneut zu installieren. Üblicherweise wird diese Datei unter Windows im Verzeichnis „C:/windows/system32“ abgelegt. Wenn allerdings das automatisierte Setup wider Erwarten fehl schlägt, können Sie die Datei versuchen mit „regsrv32.exe“ manuell über die Kommandozeile zu registrieren, oder Sie kopieren die OCX-Datei in das Applikationsverzeichnis direkt.

Abb. 17: Laufzeitfehler bei nicht erfolgreich registrierter Datei MikroSim.OCX.

PLF-Datei wird nicht gefunden

MikroSim überprüft grundsätzlich nur das Applikationsverzeichnis, (d.h. das Verzeichnis von MikroSim.exe) nach einer PLF-Datei. Wird diese dort nicht gefunden und auch kein Wechseldatenträger mit einer PLF-Datei im Wurzelverzeichnis, erscheint die Meldung „PLF nicht vorhanden“ zu Programmstart. Das Programm kann weiterhin als Demoversion verwendet werden.

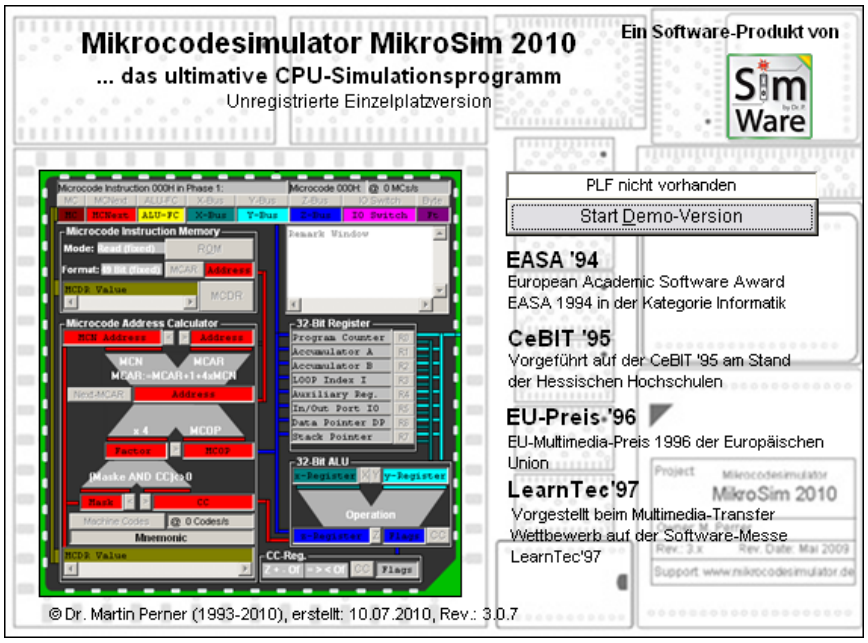

Abb. 18: Wenn keine PLF-Datei gefunden wird, kann MikroSim nur als Demoversion gestartet werden.

USB-Stick-Lizenz (Dongle) wird nicht erkannt

Wenn Sie MikroSim mit einer USB-Stick-Lizenz als Vollversion betreiben möchten, wird diese Lizenz nur dann verwendet, wenn der USB-Stick bei Programmstart als Wechseldatenträger erkannt wurde. Dabei muss sichergestellt sein, dass keine andere (evtl. ungültige) PLF-Lizenzdatei in dem Applikationsverzeichnis vorgefunden wird.

Haben Sie beispielsweise MikroSim getestet und nach Ablauf der Testversion die Dongle-Version erworben, müssen Sie nach der angelegten Datei „**MikroSim.plf**“ aufsuchen und löschen. Unter dem Betriebssystem Windows Vista(64Bit) kann diese Datei unter dem Verzeichnis „**C:\Users*Username*\AppData\Local\VirtualStore\Program Files (x86)\Mikrocodesimulator MikroSim 2010/**“ abgelegt sein. U.U. ist dieses Verzeichnis ist üblicherweise schreibgeschützt und versteckt. Mit den entsprechenden Einstellungen des Explorers können diese Verzeichnisse sichtbar gemacht werden.

Testversion ist abgelaufen

MikroSim kann zum ausgiebigen Testen als 10-tägige Testversion betrieben werden. Dazu kann das Programm im Selbstregistrierungsprozess eine zeitlich begrenzt gültige PLF-Datei erzeugen. Ist der Testzeitraum abgelaufen, erscheint nach der Überprüfung der PLF-Datei im Einstiegsfenster der Hinweis, dass die 10-tägige Evaluierungsdauer abgelaufen ist. Das Programm kann dennoch über den Testzeitraum hinaus als Demoversion betrieben werden.

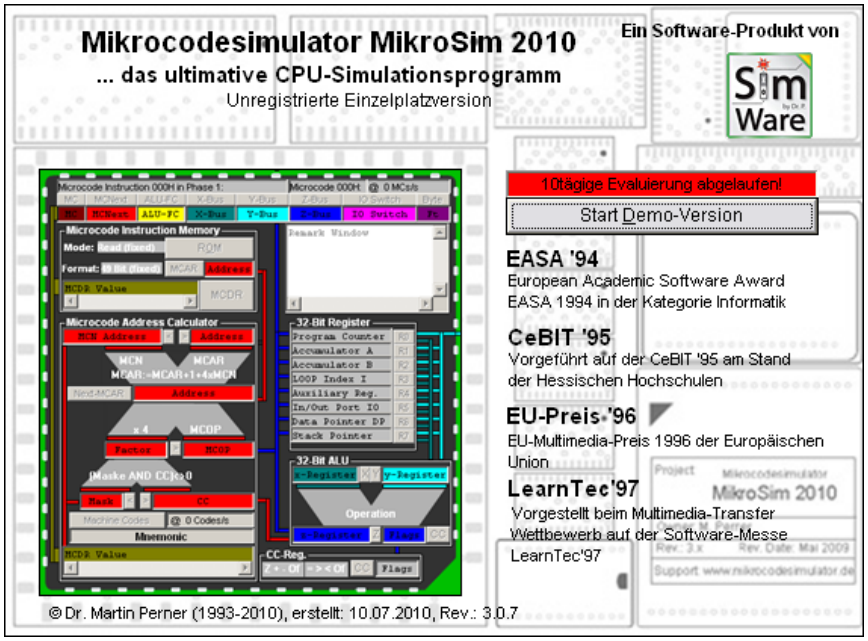

Abb. 19: Einstiegsfenster mit Hinweis auf eine abgelaufene Evaluationsperiode von MikroSim. Das Programm kann weiterhin als Demo-Version und den damit verbundenen Einschränkungen gestartet werden.

Internet-Zeit nicht verfügbar

Wird das Programm als Mehrplatzversion oder als 10tägige Testversion betrieben, so muss zu Programmstart eine Verbindung mit dem Internet aufgebaut sein, über die MikroSim die Gültigkeit der Lizenz ermitteln kann. Ist keine System-Zeit vom Internet-Zeitserver verfügbar, kann MikroSim nur als Demo-Version betrieben werden.

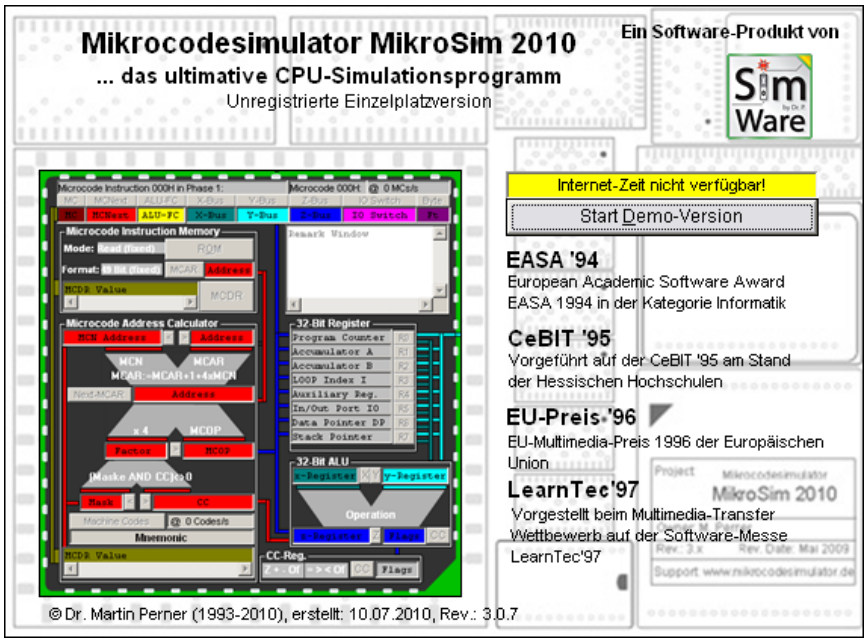

Abb. 20: Wenn die Internet-Zeit zu Programmstart zur Überprüfung der Lizenzgültigkeit nicht verfügbar ist, kann MikroSim nur im Demoversionsmodus gestartet werden.

System-Zeit nicht synchronisiert

Dieser Hinweis erscheint, wenn Sie MikroSim aktivieren möchten und keine Internet-Verbindung zum Internet-Zeitserver hergestellt werden konnte, um die Systemzeit auf dem Rechner zu kontrollieren, oder eine Abweichung der Systemzeit von der Internet-Zeit festgestellt wurde.

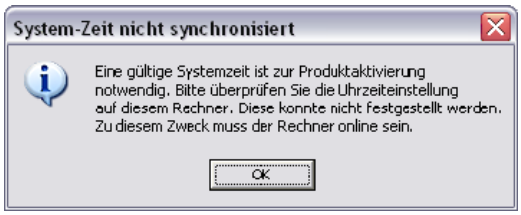

Abb. 21: Systemzeit nicht synchronisiert: Dieser Hinweis erscheint, wenn die Überprüfung auf eine gültige Systemzeit fehl schlägt.

In diesem Fall überprüfen Sie bitte, ob die am Rechner eingestellte Uhrzeit und Datum nicht mehr als 15 Minuten mit der Internet-Zeit übereinstimmt. Ist diese korrekt eingestellt und der Hinweis erfolgt immer noch, überprüfen Sie, ob Sie mit dem Rechner online sind und auf das Internet zugreifen können. Wenn Sie eine Fire-Wall oder ein Internet-Schutzprogramm benutzen, überprüfen Sie, ob die Applikation MikroSim auf das Internet zugreifen darf und nicht von diesem Programm vom Internet-Zugang sicherheitshalber abgewiesen wird.

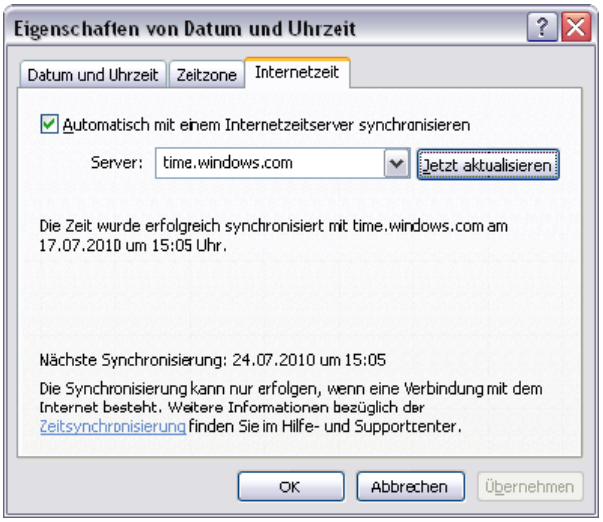

Abb. 22: Das Formular „Eigenschaft von Datum und Uhrzeit“ wird aufgerufen über die Systemsteuerung „Datum und Uhrzeit“ oder über „Doppelklick“ auf die Uhrzeit in der Statusleiste. Hier kann überprüft werden, ob der Internetzeitserver erreichbar ist oder der Dienst deaktiviert wurde.

Im letzten Schritt überprüfen Sie bitte, ob der Internetzeitserver erreichbar ist. Dies erkennen sie in dem Fenster, welches zur Einstellung der Uhrzeit benutzt wird. Hier können Sie manuell die Uhrzeit mit dem Internetzeitserver synchronisieren. Diese Funktion wird auch bei der Zeitüberprüfung verwendet. Funktioniert diese für Windows, so kann auch MikroSim diesen Dienst erfolgreich ausführen, um die Internet-Zeit für die notwendige Überprüfung zu ermitteln.

Testversion nicht aktivierbar

Über den Menüpunkt Registrierung kann MikroSim als 10-tägige Testversion aktiviert werden. Dazu muss der Rechner mit dem Internet verbunden und eine Überprüfung der Systemzeit möglich sein. Die automatisierte Registrierung ist möglich, wenn noch keine PLF-Datei im Applikationsverzeichnis angelegt oder dort abgelegt wurde. In diesem Fall ist die Schaltfläche „Starte 10tägige Produktaktivierung“ aktiv. Nach erfolgtem Versuch einer Produktaktivierung wird eine gültige PLF-Datei im Applikationsverzeichnis abgelegt und die Schaltfläche ist nicht mehr aktivierbar.

Abb. 23: Formular zur Produkt-Registrierung und 10-tägiger Produktaktivierung. Über die Schaltfläche „Erzeuge PLF“ kann eine inaktive Produkt-Lizenzdatei (PLF) generiert werden, welche eine gültige und personalisierte Produkt-ID erzeugt. Eine 10-tägige Produktaktivierung ist möglich, wenn die Schaltfläche aktiv ist und keine vorherige Produktaktivierung erfolgt ist.

Sollte zuvor bereits auf dem Rechner eine Programmversion aktiviert und somit auf dem Lizenzserver unter www.mikrocodesimulator.de registriert worden sein, wird der Registrierprozess mit folgendem Informationsfenster abgeschlossen:

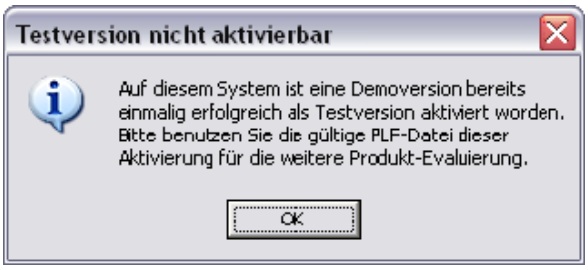

Abb. 24: Testversion nicht aktivierbar.

Lizenzschlüssel nicht gültig

Dieser Hinweis erscheint, wenn Sie eine Programmdatei benutzen möchten, die von einem Programm eines anderen, nicht der Gruppenlizenz angehörigen Lizenz-Nehmers erzeugt worden ist. Grundsätzlich können Lizenznehmer nur die mitgelieferten Demoversions-Beispiel laden und Ihre eigenen erzeugten Simulatordateien. Deswegen ist es wichtig, dass Lizenznehmer, die Simulatordateien zu Übungszwecken laden, speichern und gegenseitig austauschen möchten, Mitglieder der gleichen Lizenznehmergruppe sind, d.h. sind gemäß gültiger Lizenzdatei Lizenznehmer mit dem gleichen Benutzernamen von derselben Institution.

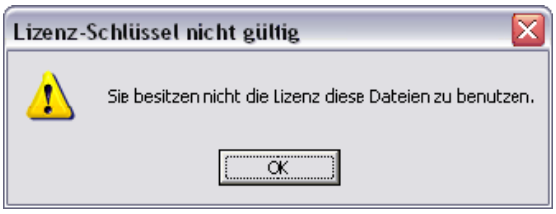

Abb. 25: Lizenzschlüssel nicht gültig.

Einschränkung der Demoversion

In der Demoversion sind einige Funktionen nicht benutzbar. Diese Einschränkungen werden durch dieses Informationsfenster angezeigt. Bei Benutzung einer Vollversion oder 10-tägigen Testversion sind alle Funktionen nutzbar. In diesem Fall erscheint das Informationsfenster somit nicht.

Abb. 26: Hinweisfenster einer Funktionseinschränkung aufgrund des Programmbetriebs als Demoversion.

Runtime-Error 13; Laufzeitfehler 13 – Type Mismatch

MikroSim ist unter der deutschsprachigen Entwicklungsumgebung und Betriebssystemeinstellungen entwickelt und getestet worden. Bei abweichenden Sprach und Ländereinstellungen kann es zu Typkonvertierungsproblem kommen, insbesondere bei der Verarbeitung von Datumsangaben. Dieser Fehler tritt bei MikroSim gleich zu Programmstart auf.

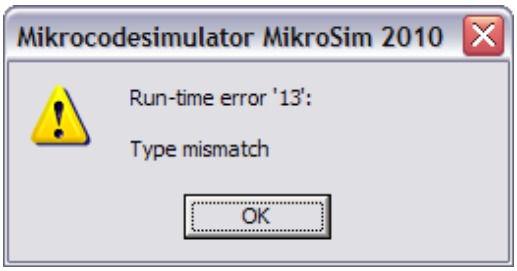

Abb. 27: Laufzeitfehler durch Typkonvertierungsprobleme bei länderabhängigen Einstellungen wie dem Datumsformat.

Dieser Fehler kann jedoch behoben werden, indem in der Systemsteuerung unter Regions- und Sprachoptionen die Deutsche Länderkennung gewählt wird und insbesondere bei den Trennzeichen für die Datumsangabe auf das Format „TT.MM.JJJJ“ geachtet wird. Dieses Problem ist mit einem Programm-Update ab Revision 3.0.11/1.0.4 von MikroSim/MikroBAT behoben.

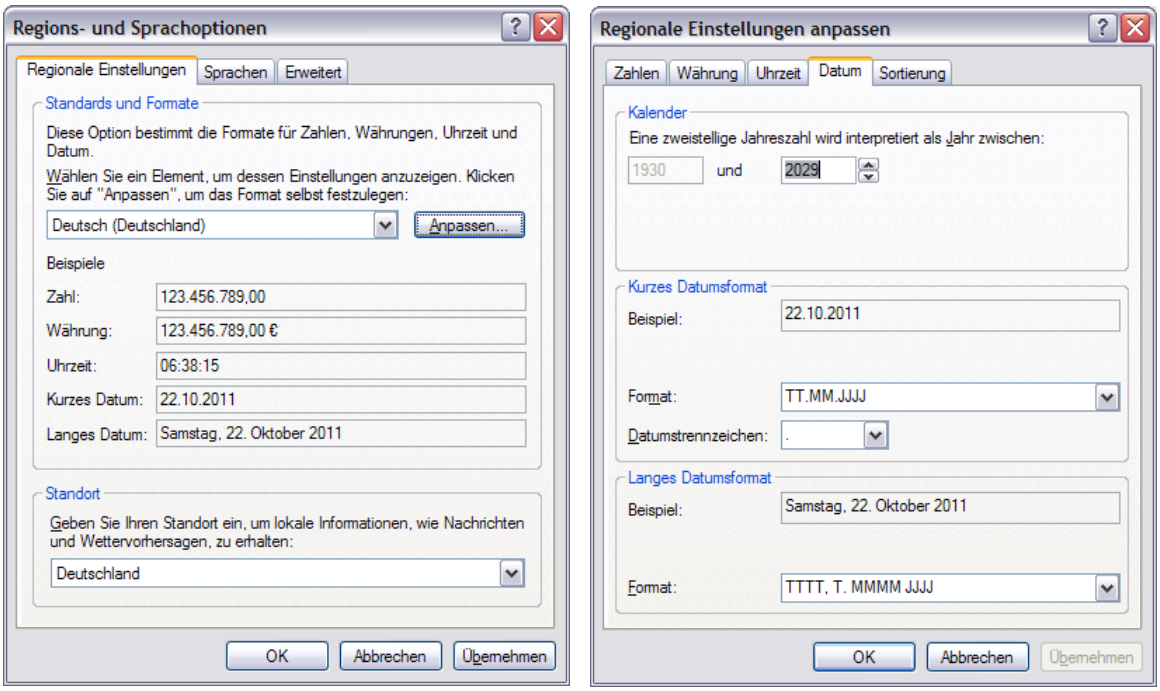

Abb. 28: Laufzeitfehler durch Typkonvertierungsprobleme bei länderabhängigen Einstellungen, insbesondere bezüglich des Datumformats.