2010

RiSa

Anleitung

http://risa.gym-vilshofen.de
r i s a @ g m x . e u

Letzte Änderung: 28.06.2010

Inhalt

1 RiSa: Anleitung

1.1 Installation

RiSa lässt sich mittels eines Installationsprogramms auf dem Zielrechner installieren. Es ist aber auch direkt und ohne Installation lauffähig. Somit ist eine schnelle und einfache Verteilung auch auf mehreren Rechnern möglich.

Wird das Installationsprogramm genutzt, fügt dies auf dem Clientcomputer dem Startmenü des Benutzers eine Verknüpfung hinzu und die Anwendung kann über "Software" deinstalliert werden. Des Weiteren wird automatisch das aktuelle .Net Framework installiert, falls dies noch nicht der Fall sein sollte. Die installierte Version von RiSa kann des Weiteren automatisch nach verfügbaren Updates suchen und diese installieren.

Der zu bevorzugende Weg sollte daher das Installationsprogramm darstellen.

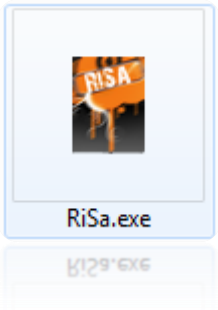
Sie können RiSa direkt über die Homepage http://risa.gym-vilshofen.de installieren. Dort finden Sie auch (als alternativen Download betitelt) die direkt ausführbare EXE-Datei ohne Installationsprogramm.

Systemanforderungen:

- RiSa ist unter Windows XP mit Service Pack 2, Windows Vista, Windows 7 sowie Windows Server 2003 mit Service Pack 1 und Windows Server 2008 lauffähig.
- Es benötigt darüber hinaus ein bereits installiertes, aktuelles .Net Framework (welches aber häufig bereits mit dem zugrundeliegenden Windows eingerichtet ist)[1]. Wird das Installationsprogramm von RiSa genutzt, installiert dies automatisch ein fehlendes .Net Framework nach.
- PC mit einem Pentium III-Klasse Prozessor, mindestens 600 MHz, empfohlen sind aber mindestens 1 GHz. (Ein Intel Atom-Prozessor, wie er in aktuellen Netbooks eingebaut ist, genügt ebenfalls.)
- 256MB RAM (mehr sind auf jeden Fall zu empfehlen)
- Video: Mindestens eine Auflösung von 1024 x 600 Pixel (High-Color 16-Bit).

1.2 RiSa starten

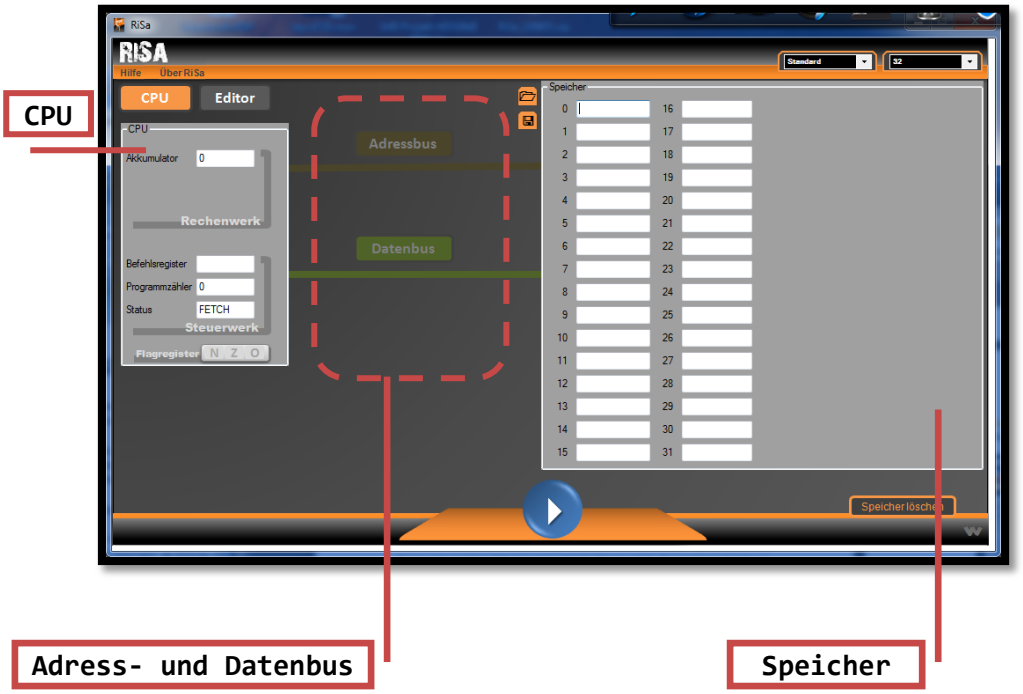
Ist RiSa installiert, findet sich im Startmenü eine Verknüpfung. Ist es nicht installiert, genügt ein Doppelklick auf die ausführbare Datei, um die Anwendung zu starten.



[1] Das .Net Framework ist kostenlos unter http://www.microsoft.de/net/ downloadbar.

1.3 Programmansicht

Nach dem Start präsentiert sich RiSa mit folgendem Fenster:



Die Ansicht zeigt dabei standardmäßig den Zentralprozessor (**CPU**) im linken sowie den **Speicher** (mit anfangs 32 Speicherzellen) im rechten Bereich.

Die CPU enthält verschiedene Register:

- Rechenwerk (Datenregister)
  - Akkumulator (AKK): General-Purpose-Register
  - EBX (Basisregister): General-Purpose-Register; Dieses Register steht nur in den Befehlssätzen „Minimal" und „Maximal" zur Verfügung.
  - ECX (Zähler); Dieses Register steht nur in den Befehlssätzen „Minimal" und „Maximal" zur Verfügung.
- Steuerwerk
  - Befehlsregister (IR, Instruction Register): Befehlsregister, das den aktuellen Befehl speichert und dekodiert.
  - Programmzähler (EIP, Instruction Pointer): Befehlszeiger, der auf die Adresse der aktuellen bzw. nächsten Speicherzelle verweist, aus der der aktuelle bzw. nächste Befehl in den Instruction Register (IR) geladen wird.
  - Status (EFLAGS): Statusregister, das hier im vereinfachten Modell von RiSa genutzt wird, um anzugeben, in welchem Zustand (Fetch, Decode oder Execute bzw. Halt oder Error) sich der Prozessor befindet.

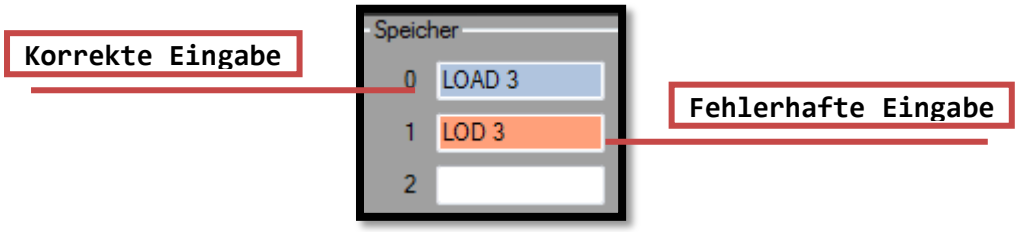
Die Inhalte der Datenregister lassen sich explizit manipulieren, die des Steuerwerkes nur implizit[2].

Außerdem sind folgende Flagregister Bestandteil der CPU:

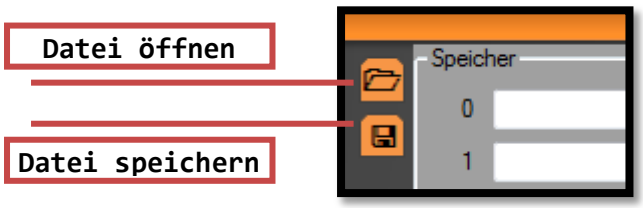
- N: Gibt an, ob der Akkumulator nach der letzten Rechenoperation negativ ist.
- Z: Gibt an, ob der Akkumulator nach der letzten Rechenoperation gleich Null ist.
- O: Gibt an, ob im Akkumulator nach der letzten Rechenoperation ein Overflow aufgetreten ist.

Ein gesetztes Flagregister wird dabei optisch dargestellt und lässt sich über spezielle Jump-Befehle im Programmcode abfragen.

Der Inhalt der einzelnen Zellen des Speichers lässt sich direkt bearbeiten, indem man auf das entsprechende Feld klickt. Die Zellen sind dabei „intelligent". Sie erkennen eine korrekte Eingabe[3] und visualisieren dies entsprechend.



Direkt links neben dem Hauptspeicher befinden sich zwei Buttons, mit deren Hilfe man den Inhalt der Speicherzellen in eine Datei speichern bzw. eine vorhandene Datei öffnen kann. Der Inhalt der Datei wird beim Öffnen zeilenweise in die Speicherzellen geschrieben, beginnend bei Speicherzelle **0**.

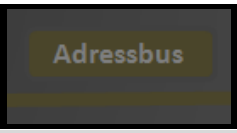
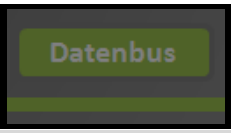
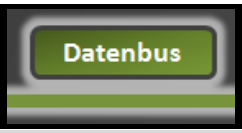


[2] Siehe dazu die Beschreibung der einzelnen Befehle.
[3] Dies kann ein korrekt eingegebener Befehl oder ein Zahlenwert sein.

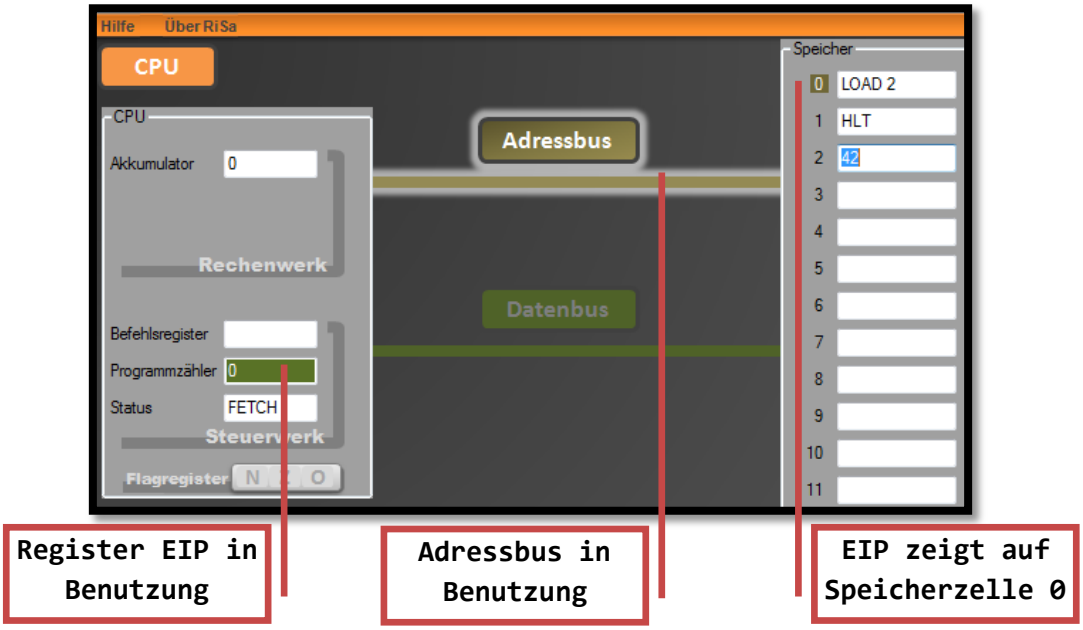
RiSa besitzt eine integrierte Hilfe, die sich über den entsprechenden Button starten lässt. Die Hilfe zeigt jeweils nur genau die Befehle an, die der aktuell gewählte Befehlssatz unterstützt.

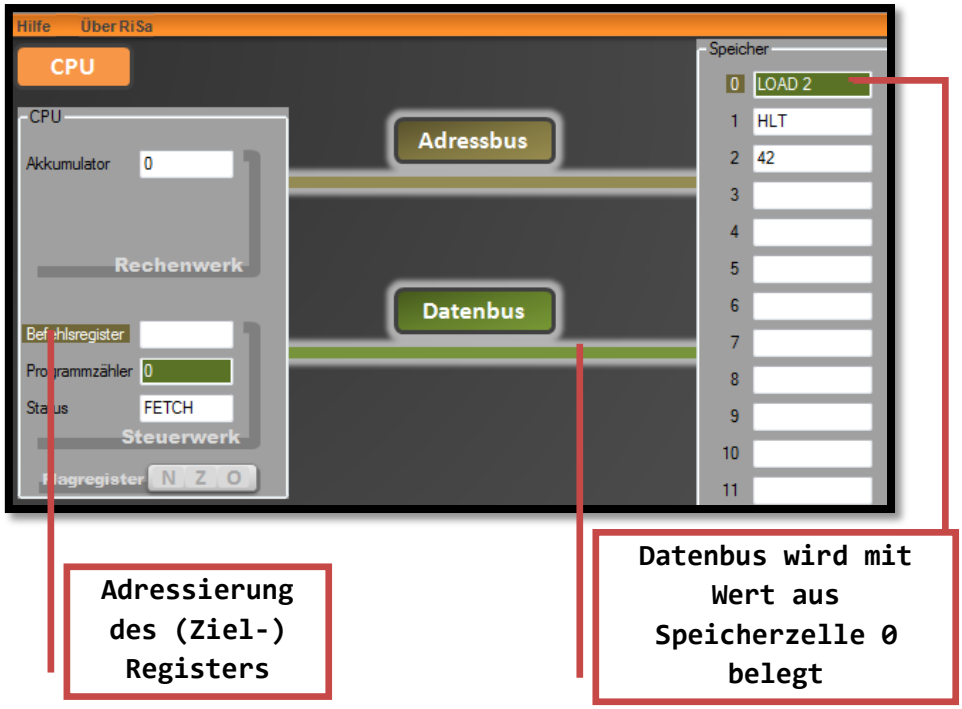
Informationen über RiSa lassen sich über den entsprechend benannten Button anzeigen.

Der „**Adress- und Datenbus**"-Bereich visualisiert, welcher der Busse gerade im jeweiligen Ausführungsschritt benutzt wird. Man kann vier Möglichkeiten unterscheiden:

			
Adressbus nicht aktiv	Adressbus aktiv	Datenbus nicht aktiv	Datenbus aktiv

Die in Benutzung befindlichen Register, Speicherzellen beziehungsweise Register- und Speicheradressen werden dabei jeweils in gleicher Farbe markiert. Beispielhaft wird dies in den nächsten Abbildungen illustriert.

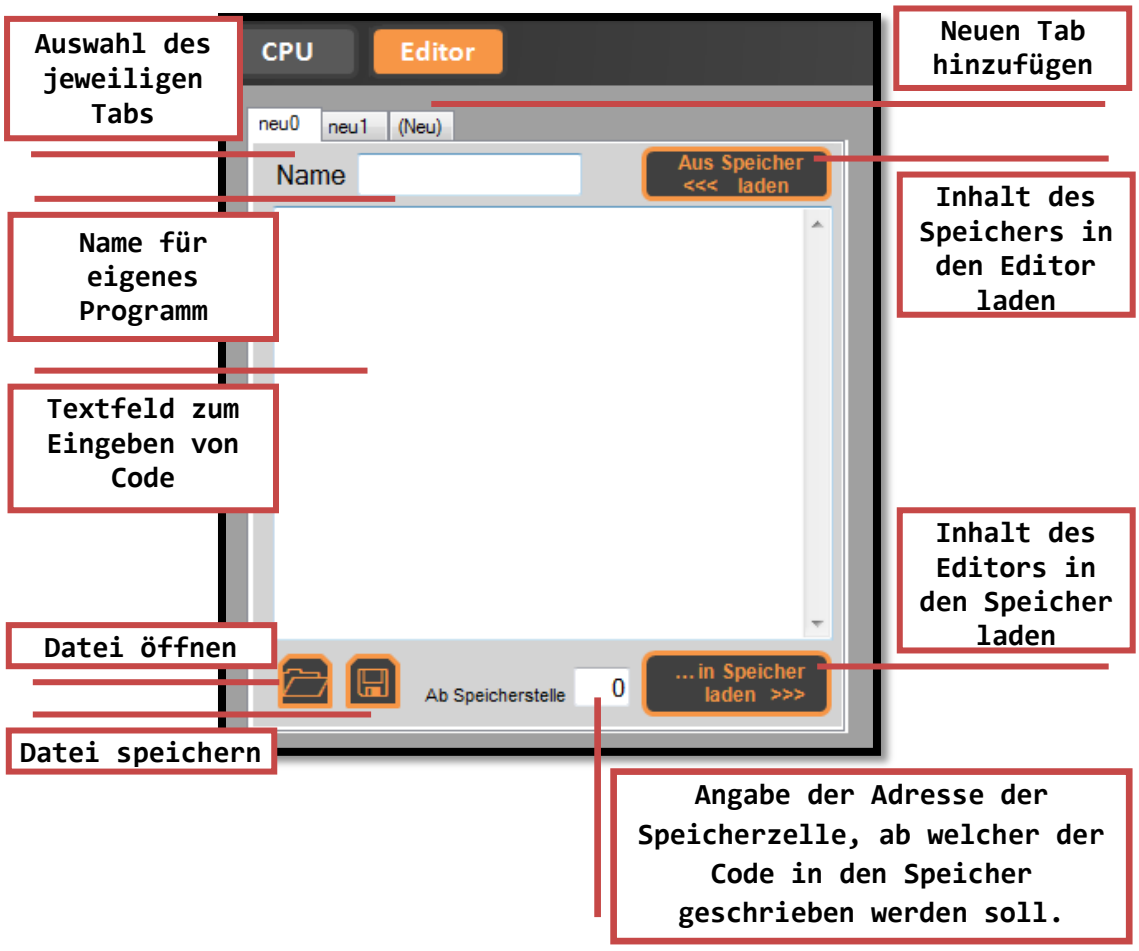




Editor

Mit Hilfe der Buttons „CPU“ und „Editor“ lassen sich die CPU aus- und der Editor einblenden.

Ein Klick auf den „Editor“-Button blendet den Editor ein.



Der Editor dient dem einfachen Erstellen und Bearbeiten von eigenen Programmen. Diesen lässt sich je ein Name zuordnen, der automatisch auch als Bezeichnung des jeweiligen Tabs verwendet wird. Es lassen sich beliebig viele Tabs (und somit Programme) anlegen. Neue Tabs werden einfach durch einen Klick auf den Tab mit der Beschriftung „(Neu)“ hinzugefügt. Somit können auf einfache Weise verschiedene Aufgaben oder Lösungsmöglichkeiten bearbeitet und getestet werden.

Eingegebene Programme lassen sich als Datei speichern. Der Name des Tabs wird dabei automatisch als neuer Dateiname vorgeschlagen. Werden zuvor bereits gespeicherte Programme wieder geöffnet, wird der Dateiname als Name des Tabs gesetzt.

Es lassen sich Kommentare in den Quellcode einfügen, dazu muss die betreffende Zeile mit zwei Schrägstrichen // begonnen werden.

Um den Code eigener Programme auszuführen, muss dieser zuvor in den Speicher geladen werden. Ein späteres Kopieren zurück in den Editor ist ebenfalls möglich.

Will man Code in den Speicher laden, lässt sich zusätzlich die Adresse der Speicherzelle angeben, ab welcher der Code geschrieben wird. Die folgenden zwei Abbildungen sollen diese Möglichkeit verdeutlichen. In der ersten Abbildung wird ab Speicherzelle `0` geschrieben, in der zweiten Abbildung ab Speicherstelle 3.

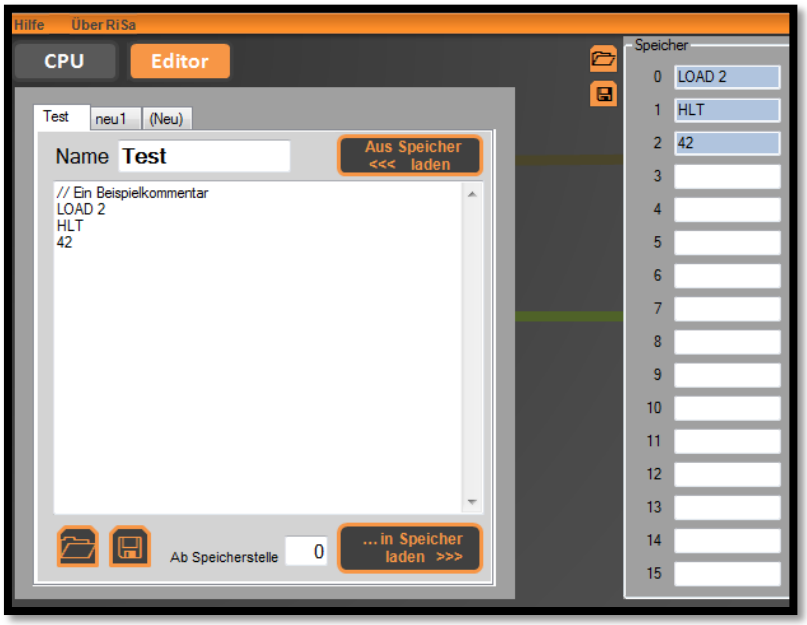


Abbildung 1 - Laden ab Speicherstelle `0`

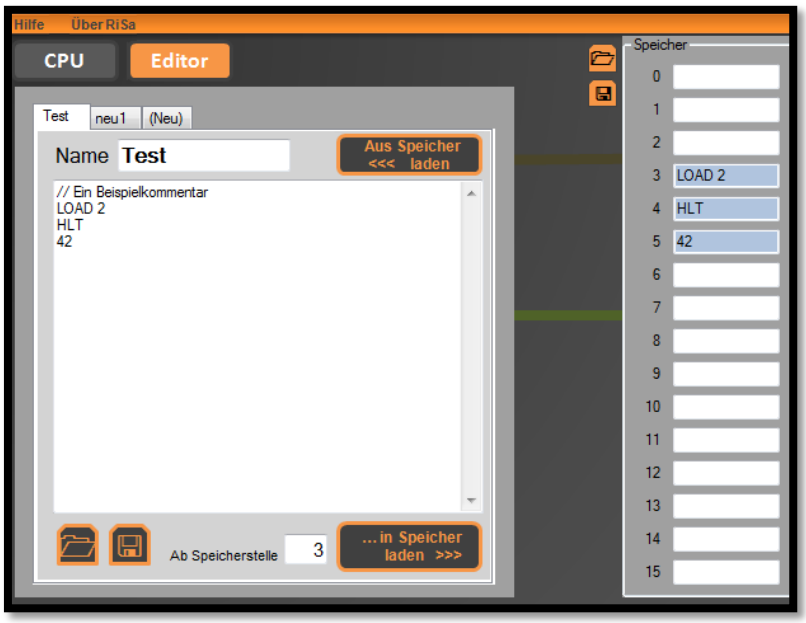
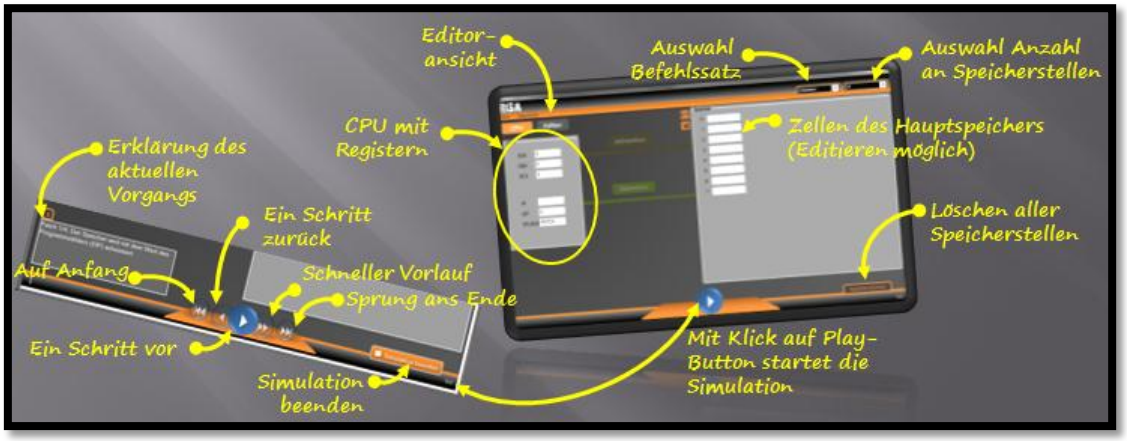


Abbildung 2 - Laden ab Speicherstelle 3

Die weitere Bedienung des Programmes lässt sich mit Hilfe des folgenden Schaubildes verdeutlichen:



Auswahl Befehlssatz

Durch Klick auf das Dropdown-Menü lassen sich die verschiedenen Befehlssätze auswählen. Eine Änderung löscht alle eventuell eingegebenen Werte des aktuellen Speichers.

Auswahl Anzahl an Speicherstellen

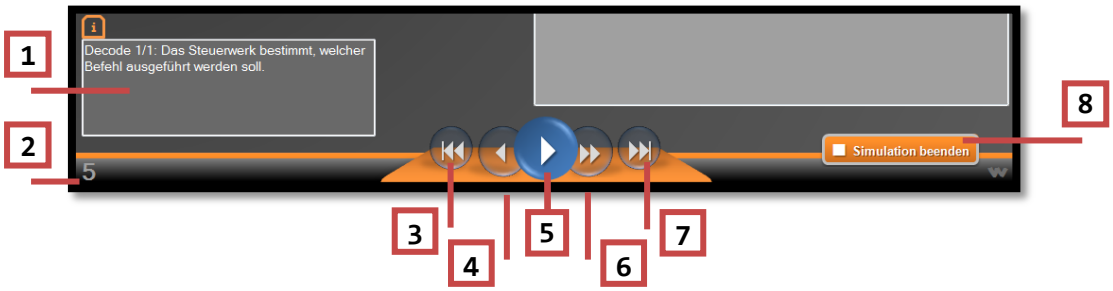
Hier lässt sich die Anzahl der Zellen des Speichers einstellen. Mögliche Werte sind dabei: 8, 16, 32 und 64. Eine Änderung löscht alle eventuell eingegebenen Werte des aktuellen Speichers.

Löschen aller Speicherstellen

Ein Klick auf diesen Button löscht alle eingegebenen Werte des aktuellen Speichers.

Play-Button

Mit einem Mausklick auf diesen Button startet die Ausführung bzw. die Simulation des in die Speicherzellen eingegebenen Programmcodes. Gleichzeitig wird die Ansicht um weitere Steuerelemente und ein Informationsfeld erweitert.



Das Informationsfeld (1) zeigt an, in welchem Zustand sich der Prozessor gerade befindet, und erläutert dies durch eine kurze Beschreibung.

(2) stellt einen „Schrittzähler" dar, der den jeweiligen Schritt der Simulation zählt und somit auch das Vor- bzw. Zurück-Navigieren erleichtert.

Mittels der Taste (3) lässt sich an den Anfang, mit der Taste (7) an das Ende des Simulationslaufes springen. Mit dem Play-Button (5) wird der jeweils nächste Schritt der Simulation durchgeführt, der Zurück-Button (4) geht einen Schritt zurück. Ein Klick auf (8) beendet die Simulation und führt zurück zur Standardansicht, in welcher das eigene Programm wieder editiert werden kann.

Der schnelle Vorlauf startet mit einem Mausklick auf die Taste (6). Die Ansicht der Steuerelemente ändert sich wie folgt:

Ein Klick auf den Pause-Button (10) beendet den schnellen Vorlauf und führt wieder zum normalen Einzelschrittmodus zurück. Mit Hilfe von (11) lässt sich die Geschwindigkeit des schnellen Vorlaufs einstellen. Die jeweils zuletzt gewählte Geschwindigkeit wird bei weiteren schnellen Vorläufen automatisch wieder verwendet.

Niedrige Geschwindigkeit	Mittlere Geschwindigkeit (Standard)	Hohe Geschwindigkeit

2 RiSa: Befehlssätze

2.1 Befehlssatz „Standard“

Der Befehlssatz „Standard“ stellt den „normalen“ Befehlssatz dar. Dies ist der Befehlssatz, den Sie für den Unterricht benötigen. Die CPU hat hier als Register den Akkumulator, aber es fehlen die weiteren Register EBX und ECX. Die Befehle benötigen daher meist nur einen Operanden, da sie sich meist implizit auf den Akkumulator beziehen. Der Befehlssatz umfasst folgende Befehle.

Registerbefehle

Befehl	Erklärung	Beispiel
ADD	Addiert den Wert des ersten Operanden (Quelle) zum Wert des Akkumulators und speichert das Ergebnis im Akkumulator.	`ADD 3`
COMP	Vergleicht den Wert des Akkumulators mit dem Wert des angegebenen Operanden. Sind beide Werte gleich, wird das Flag Z der CPU gesetzt. Ist der Akkumulator kleiner, wird das Flag N gesetzt.	`COMP 3`
DIV	Dividiert den Akkumulator (AKK) durch den Wert des angegebenen Operanden und speichert das Ergebnis im Akkumulator.	`DIV 3`
LOAD	Kopiert den Wert des angegebenen Operanden in den Akkumulator (AKK).	`LOAD 3`
LOADI	Die ganze Zahl des angegebenen Operanden wird in den Akkumulator (AKK) geladen. Negative Werte sind möglich.	`LOADI -42`
MUL	Multipliziert den Wert des angegebenen Operanden mit dem Akkumulator (AKK) und speichert das Ergebnis im Akkumulator.	`MUL 3`
STORE	Speichert den Wert des Akkumulators in die angegebene Speicherzelle.	`STORE 3`
SUB	Subtrahiert den Wert des ersten Operanden (Quelle) vom Wert des Akkumulators und speichert das Ergebnis im Akkumulator.	`SUB 3`

Programmbefehle

Befehl	Erklärung	Beispiel
HLT	Hält den Prozessor an.	`HLT`
JMP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher.	`JMP 3`
JMPN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag N gesetzt.	`JMPN 3`
JMPNN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag N nicht gesetzt.	`JMPNN 3`
JMPO	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag O gesetzt.	`JMPO 3`
JMPNO	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag O nicht gesetzt.	`JMPNO 3`
JMPP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls weder Flag N noch Flag Z gesetzt sind.	`JMPP 3`
JMPNP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls entweder Flag N oder Flag Z gesetzt ist.	`JMPNP 3`
JMPZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag Z gesetzt.	`JMPZ 3`
JMPNZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag Z nicht gesetzt.	`JMPNZ 3`

2.2 Befehlssatz „Erweitert“

Der Befehlssatz „Erweitert“ ergänzt den „Standard“-Befehlssatz um folgende Befehle:

Registerbefehle (zusätzlich zum „Standard“-Befehlssatz)

Befehl	Erklärung	Beispiel
DEC	Der Inhalt des Akkumulators (AKK) wird um die Zahl 1 vermindert.	`DEC`
INC	Der Inhalt des Akkumulators (AKK) wird um die Zahl 1 erhöht.	`INC`
NOP	Wartet einen Takt und führt in dieser Zeit nichts aus.	`NOP`

2.3 Befehlssatz „Maximal“

Mit dem Befehlssatz „Maximal“ können alle Befehle von RiSa genutzt werden. Es stehen zusätzlich die Register EBX und ECX zur Verfügung. Die Befehle erwarten meist zwei Operanden, insofern ist die Syntax zum Teil anders als die der entsprechenden Befehle in den Befehlssätzen „Standard“ und „Erweitert“. Es werden folgende Kommandos unterstützt:

Registerbefehle

Befehl	Erklärung	Beispiel
ADD	Addiert den Wert des zweiten Operanden (Quelle) zum Wert des ersten Operanden (Ziel) und speichert das Ergebnis im ersten Operanden (Ziel). Es können nicht beide Operanden Adressen des Speichers sein.	`ADD AKK 3`
COMP	Vergleicht den Wert des Akkumulators mit dem Wert des angegebenen Operanden. Sind beide Werte gleich, wird das Flag Z der CPU gesetzt. Ist der Akkumulator kleiner, wird das Flag N gesetzt.	`COMP 3`
DEC	Der Inhalt des genannten Registers oder Speicherplatzes wird um die Zahl 1 vermindert.	`DEC EBX`
DIV	Dividiert den Akkumulator (AKK) durch den Wert des angegeben Operanden und speichert das Ergebnis in AKK.	`DIV 3`
INC	Der Inhalt des genannten Registers oder Speicherplatzes wird um die Zahl 1 erhöht.	`INC ECX`
LOAD	Kopiert den Wert des angegebenen Operanden in den Akkumulator (AKK).	`LOAD 3`
LOADI	Die ganze Zahl des angegebenen Operanden wird in den Akkumulator (AKK) geladen. Die Zahl kann dabei auch negativ sein.	`LOADI -42`
MUL	Multipliziert den Wert des angegebenen Operanden mit dem Akkumulator (AKK) und speichert das Ergebnis in AKK.	`MUL 3`
MOV	Kopiert den Wert des zweiten Operanden (Quelle) in den ersten Operanden (Ziel). Es können nicht beide Operanden Adressen des Speichers sein.	`MOV AKK 3`
NEG	Negiert den angegebenen Operanden	`NEG 3`
STORE	Speichert den Wert von AKK in die angegebene Speicherzelle.	`STORE 3`

SUB	Subtrahiert den Wert des zweiten Operanden (Quelle) vom Wert des ersten Operanden (Ziel) und speichert das Ergebnis im ersten Operanden (Ziel). Es können nicht beide Operanden Adressen des Speichers sein.	`SUB AKK 3`

Programmbefehle

Befehl	Erklärung	Beispiel
HLT	Hält den Prozessor an.	`HLT`
JEBXNZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls EBX ungleich **0**.	`JEBXNZ 3`
JEBXZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls EBX= **0**.	`JEBXZ 3`
JECXNZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls ECX ungleich **0**.	`JECXNZ 3`
JECXZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls ECX=**0**.	`JECXZ 3`
JEBXN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls EBX < **0**.	`JEBXN 3`
JEBXP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls EBX > **0**.	`JEBXP 3`
JECXN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls ECX < **0**.	`JECXN 3`
JECXP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls ECX > **0**.	`JECXP 3`
JMP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher.	`JMP 3`
JNZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls ECX ungleich **0**.	`JNZ 3`
JMPN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag N gesetzt.	`JMPN 3`

JMPNN	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag N nicht gesetzt.	`JMPNN 3`
JMPO	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag O gesetzt.	`JMPO 3`
JMPNO	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag O nicht gesetzt.	`JMPNO 3`
JMPP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls weder Flag N noch Flag Z gesetzt sind.	`JMPP 3`
JMPNP	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls entweder Flag N oder Flag Z gesetzt ist.	`JMPNP 3`
JMPZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag Z gesetzt.	`JMPZ 3`
JMPNZ	Springt für die Befehlsausführung an die angegebene Stelle im Speicher, falls Flag Z nicht gesetzt.	`JMPNZ 3`
NOP	Wartet einen Takt und führt in dieser Zeit nichts aus.	`NOP`
OUT	Gibt den Wert der angegebenen Speicherzelle in einem extra Fenster aus.	`OUT 3`

2.4 Befehlssatz „Minimal“

Der Befehlssatz „Minimal“ umfasst folgende Befehle, die analog zu den Befehlen des Befehlssatzes „Maximal“ zu verwenden sind. Wie beim Befehlssatz „Maximal“ stehen die zusätzlichen Register EBX und ECX zur Verfügung.

Registerbefehle

DEC
INC
MOV

Programmbefehle

HLT
JMP
JNZ
JMPZ
JMPN
JMPO

Eine Beschreibung der Befehle findet sich im Kapitel „2.3 Befehlssatz ‚Maximal‘“.

3 Links

RiSa Homepage

http://risa.gym-vilshofen.de