Was ist ein ToyProcessor?

Der ToyProcessor ist eine vereinfachte Simulation eines normalen Prozessors in einem Computer. Die Bauteile des Processors wurden reduziert auf das Steuerwerk, den RAM (Random-Access-Memory), die ALU (Arithmetic-Logic-Unit), den Accu (Accumulator), das IR (Instruction register) und den PC (Programm counter). Der ToyProcessor setzt die Von-Neumann-Architektur um.

RAM: Im RAM stehen alle Befehle aus dem jeweils geladenen Toy-Programm in binärer oder hexadezimalen Schreibweise. Der Übersichtlichkeit wegen, sind alle restlichen Einträge einfach 0 - natürlich entspricht dies nicht dem realistischen Aussehen eines Speichers. Der RAM kann insgesamt 4096 Einträge beinhalten. Der RAM wird durch das IR und die ALU ausgelesen und kann durch die ALU auch wieder beschrieben werden.

ALU: In der ALU werden alle Rechenoperationen durchgeführt. Dazu wird der jeweilige Befehl aus dem RAM und der aktuelle Wert des Akkus geladen und die Operation ausgeführt. Die ALU kann beim Speicherbefehl den RAM wieder mit dem Ergebnis beschreiben.

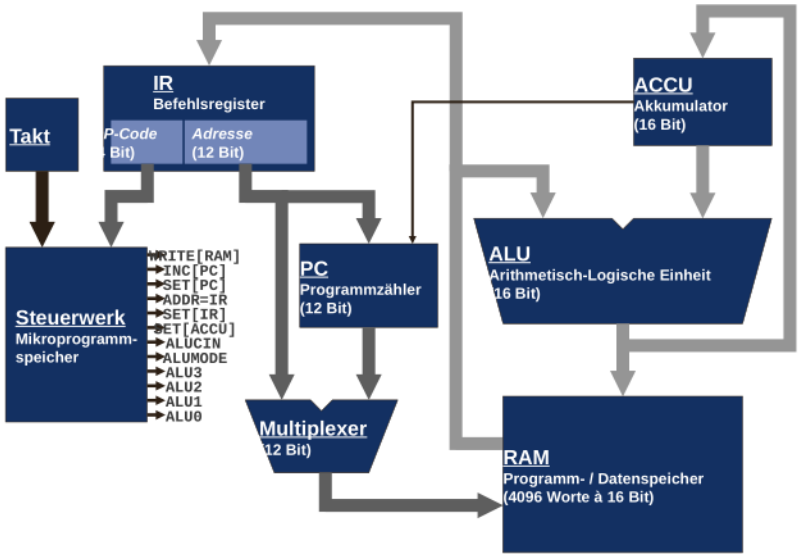


Abbildung 1: Blockschaltbild des ToyProcessors

Accu: Im Accu stehen die aktuellen Werte für eine Rechenoperation zur Verfügung. Diese werden der ALU zur Verfügung gestellt, wenn sie diese für eine Operation benötigt. Auch das Ergebnis einer ausgeführten Rechenoperation wird wieder in den Accu geschrieben. Bei einem Sprungbefehl wird getestet, ob der Accu gleich Null ist.

IR: Das IR ist das Befehlsregister des Steuerwerks. Hier werden die aktuell zu bearbeitenden Befehle geladen. Da die Befehle eines Toy-Programms aus 4 Bit OP-Code und 12 Bit Adresse bestehen, kann das IR ingesamt 16 Bit umfassen.

PC: Der PC ist zu Anfang eines jeden Toy-Programmes auf Null gesetzt. Nach jedem ausgeführten Befehl wird er um Eins erhöht. Somit dient er als Indikator, an welcher Stelle sich der ToyProcessor im Programm im Moment befindet. Er gibt also an, aus welcher Speicherzelle des RAMs der nächste Befehl ausgelesen werden muss. Bei einem Sprungbefehl wird der PC auf den Wert gesetzt, zu dem gesprungen werden soll.

Steuerwerk: Das Steuerwerk führt die jeweiligen Befehle aus, die es aus dem IR übergeben bekommt. Da die Befehle eines Toy-Programms aus verschiedenen Teilbefehlen bestehen (siehe "Wie läuft ein Toy-Programm ab?") leuchten in jedem Takt die Flags zu den zugehörigen Teilbefehlen auf, die aktuell durchgeführt werden. Diese sind links des Steuerwerkblocks zu erkennen.

Wie läuft ein Toy-Programm ab?

Der ToyProcessor kann 12 Befehle verarbeiten. Intern hat ein Befehl immer das folgende Format: Die erste 4 Bit entsprechen dem OP-Code, welcher die jeweils auszuführende Operation spezifiziert. Die letzten 12 Bit geben die Adresse an, an der der Wert im RAM liegt, mit dem die jeweilige Operation durchgeführt werden soll. Das Ergebnis einer Operation wird immer in den Accu geschrieben. Die Steuerung der auszuführenden Befehle erfolgt im 2-Phasen-Takt:

OP	Mnemonic	Bedeutung
0	STO <Adresse>	speichere den Inhalt des ACCUs ins RAM
1	LDA <Adresse>	lade den ACCU mit dem Inhalt der Adresse
2	BRZ <Adresse>	springe nach Adresse, wenn der ACCU Null ist
3	ADD <Adresse>	addiere den Inhalt der Adresse zum ACCU
4	SUB <Adresse>	subtrahiere den Inhalt der Adresse vom ACCU
5	OR <Adresse>	logisches ODER des ACCUs mit dem Inhalt der Adresse
6	AND <Adresse>	logisches UND des ACCUs mit dem Inhalt der Adresse
7	XOR <Adresse>	logisches ExODER des ACCUs mit dem Inhalt der Adresse
8	NOT	logisches NICHT der Bits im ACCU
9	INC	inkrementiere den ACCU
10	DEC	dekrementiere den ACCU
11	ZRO	setze den ACCU auf Null
12	NOP	keine Operation
13	NOP	keine Operation
14	NOP	keine Operation
15	NOP	keine Operation

Abbildung 2: Befehlssatz des ToyProcessors

OP-Code	Operation	Phase	Steuerung
0	STO	1	ADDR=IR; ALU=ACCU; WRITE[RAM]
		2	ADDR=PC; SET[IR]; INC[PC]
1	LDA	1	ADDR=IR; ALU=RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
2	BRZ	1	SET[PC]
		2	ADDR=PC; SET[IR]; INC[PC]
3	ADD	1	ADDR=IR; ALU=ACCU+RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
4	SUB	1	ADDR=IR; ALU=ACCU−RAM; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
			...
8	NOT	1	ALU=~ACCU; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
9	INC	1	ALU=ACCU+1; SET[ACCU]
		2	ADDR=PC; SET[IR]; INC[PC]
			...
12 – 15	NOP	1	ADDR=PC; SET[IR]; INC[PC]
		2	–

Abbildung 3: Steuerung der Toy-Befehle

Wie schreibe ich ein Toy-Programm?

Es gibt vier unterschiedliche Möglichkeiten ein Toy-Programm zu schreiben: Die Befehle können sowohl in binärer und in hexadezimaler Schreibweise, als auch in Assembler (= Mnemonic, vgl. Abbildung 2) kodiert werden. Außerdem gibt es noch die Möglichkeit Pseudo-C-Code zu verfassen.

Die Tabelle rechts umfasst die Binär-, Hexadezimal- und Assemblervarianten. Die "xxxxxxxxxxxx" kodieren die Adresse für den jeweiligen OP-Code in binär, "yyy" stehen für die Adresse in hexadezimal und "z" gibt die Adresse als Dezimalzahl an.

Wir werden nun das Programmieren mit den Toy-Befehlen an einigen Beispielen üben. Am Ende dieses Absatzes werden wir dann noch auf die Einschränkungen des Pseudo-C-Codes eingehen.

Binär:	**Hexadezimal:**	**Assembler**
0000xxxxxxxxxxxx	$0yyy	STO z
0001xxxxxxxxxxxx	$1yyy	LDA z
0010xxxxxxxxxxxx	$2yyy	BRZ z
0011xxxxxxxxxxxx	$3yyy	ADD z
0100xxxxxxxxxxxx	$4yyy	SUB z
0101xxxxxxxxxxxx	$5yyy	OR z
0110xxxxxxxxxxxx	$6yyy	AND z
0111xxxxxxxxxxxx	$7yyy	XOR z
1000xxxxxxxxxxxx	$8yyy	NOT
1001xxxxxxxxxxxx	$9yyy	INC
1010xxxxxxxxxxxx	$Ayyy	DEC
1011xxxxxxxxxxxx	$Byyy	ZRO
1100xxxxxxxxxxxx	$Cyyy	NOP
1101xxxxxxxxxxxx	$Dyyy	NOP
1110xxxxxxxxxxxx	$Eyyy	NOP
1111xxxxxxxxxxxx	$Fyyy	NOP

Tabelle 1: Binär-, Hexadezimal- und Assemblerdarstellung der Toy-Befehle

Binär-, Hexadezimal- und Assemblerprogrammierung

Format: Das Toy-Programm sollte in einem simplen Texteditor geschrieben werden. Die Textdatei sollte die Endung *.toy bekommen.

Code: Die einzelnen Befehle werden alle untereinander in jeweils neue Zeilen geschrieben. In den RAM werden sie in der gleichen Reihenfolge geschrieben, angefangen bei der nullten Speicherzelle. Wie man der Tabelle 1 entnehmen kann, sind die verschiedenen Varianten (binär, hexadezimal und Assembler) äquivalent, daher können diese im Source-Code gemischt werden. Beispielsweise liefern somit die folgenden drei Toy-Programme zum Subtrahieren der Werte aus RAM[11] und RAM[22] dasselbe Ergebnis:

```
0001000000001011
0100000000010110
0000000000001011
```
(a) Binär

```
$100B
$4016
$000B
```
(b) Hexadezimal

```
LDA 11
SUB 22
STO 11
```
(c) Assembler

```
0001000000001011
$4016
STO 11
```
(d) Gemischt

RAM-Settings: Um mit konkreten Werten zu rechnen, können diese direkt in den RAM geschrieben und die RAM-Adresse für die jeweiligen Rechenoperationen benutzt werden. Angenommen, wir würden bei unserem Beispiel gerne 42 von 84 abziehen, so sähe das ganze wie in der folgenden Tabelle aus: Die ersten zwei Zeilen des jeweiligen Programms setzen RAM[11]=84 und RAM[22]=42, das Ergebnis wird wie oben in RAM[11] gespeichert. Die einzige Einschränkung bezüglich RAM-Setting ist, dass Werte nur in Speicherzellen geschrieben werden dürfen, welche eine höhere Nummer als die Anzahl an OP-Codes im Toy-Programm haben. Es wäre hier also nicht erlaubt gewesen, die Speicherzellen 0, 1 oder 2 zu beschreiben, da das Toy-Programm insgesamt 3 OP-Codes beinhaltet. Achtung: Binäre Ramsettings, die kürzer als 16 Bit sind, werden als Dezimalzahlen interpretiert.

```
:0000000000001011:0000000001010100
:0000000000010110:0000000000101010
0001000000001011
0100000000010110
0000000000001011
```
(a) Binär

```
:$00B:$054
:$016:02A
$100B
$4016
$000B
```
(b) Hexadezimal

```
:11:84
:22:42
LDA 11
SUB 22
STO 11
```
(c) Assembler

```
:$00B:$054
:22:42
0001000000001011
$4016
STO 11
```
(d) Gemischt

Endlosschleife: Um zu verdeutlichen, dass ein Prozessor niemals aufhört zu arbeiten, muss jedes Toy-Programm in einer Endlosschleife enden. Dies kann durch das Hinzufügen eines "ZRO" (= Setze den Accu auf Null) und eines "BRZ <einen Befehl davor>" (= Springe zum Befehl davor, d.h. zu ZRO, falls der Accu Null ist) am Ende des Toy-Programms erreicht werden.

```
:0000000000001011:0000000001010100
:0000000000010110:0000000000101010
0001000000001011
0100000000010110
0000000000001011
1011000000000000
0010000000000011
```
(a) Binär

```
:$00B:$054
:$016:02A
$100B
$4016
$000B
$B000
$2003
```
(b) Hexadezimal

```
:11:84
:22:42
LDA 11
SUB 22
STO 11
ZRO
BRZ 3
```
(c) Assembler

```
:$00B:$054
:22:42
0001000000001011
$4016
STO 11
1011000000000000
$2003
```
(d) Gemischt

Kommentare: Kommentare können mit "#" gesetzt werden. Wir kommentieren hier nun zur Veranschaulichung die Assembler-Variante unteres Toy-Programms - für die anderen Varianten funktioniert das Kommentieren analog:

```
#RAM-Settings
:11:84  #RAM[11] wird auf den Wert 84 gesetzt
:22:42  #RAM[22] wird auf den Wert 42 gesetzt

#Subtraktion der beiden Werte
LDA 11  #Läd 84, den Wert aus RAM[11]
SUB 22  #Subtrahiert davon 42, den Wert aus RAM[22]
STO 11  #Speichert das Ergebnis in RAM[11]

#Endlosschleife
ZRO     #Setzt den Accu auf Null
BRZ 3   #Springt zurück zum ZRO Befehl in RAM[3]
```

Variablen: Im Assemblercode können anstatt RAM-Settings auch Variablen deklariert werden. Diese werden dann vom ToyProcessor automatisch in Speicherzellen nach den OP-Codes geschrieben. Variablen werden wie folgt deklariert: "<variablenname> = <wert>". Variablennamen dürfen aus Groß- und Kleinbuchstaben, sowie Zahlen bestehen. Die zugewiesenen Werte dürfen eine Dezimal- oder Hexadezimalzahl sein. Es dürfen keine RAM-Settings und Variablendeklarationen gleichzeitig in einem Toy-Programm vorkommen. Des weiteren kann der Assembler mit Variablen und später auch Labeln nicht mehr mit binärem oder hexadezimalem Code gemischt werden. Hier das obige Beispiel mit Variablendeklarationen anstatt den RAM-Settings:

```
#Variablendeklarationen
a = 84  #Deklariere eine Variable mit dem Namen a und dem Wert 84
b = $2A  #Deklariere eine Variable mit dem Namen b und dem Wert 0x2A = 42

#Subtraktion der beiden Variablenwerte
LDA a  #Läd die Variable a mit dem Wert 84
SUB b  #Subtrahiert davon die Variable b mit dem Wert 42
STO a  #Speichert das Ergebnis in Variable a

#Endlosschleife
ZRO     #Setzt den Accu auf Null
BRZ 3   #Springt zurück zum ZRO Befehl in RAM[3]
```

Labels: Labels dienen als Anhaltspunkte, wohin ein Sprungbefehl im Code zurückkehren muss. Labels können also anstatt der RAM-Adresse des Befehls, zu dem zurückgesprungen werden soll, angegeben werden. Labels werden im Stil von: "<labelname>:" deklariert. Labelnamen dürfen Groß- und Kleinbuchstaben, aber keine Zahlen enthalten. Unsere Endlosschleife könnten wir mit Hilfe von Labels also wie folgt gestalten:

```
#Endlosschleife
loop:
ZRO         #Setzt den Accu auf Null
BRZ loop    #Springt zurück zum ZRO Befehl, da dieser nach dem Label "loop" kommt
```

Pseudo-C-Code

Datentypen: Da dem ToyProcessor nur 16 Bit lange Befehle zur Verfügung stehen, dürfen nur die Datentypen "short int", "char" und "bool" benutzt werden. Selbstverständlich dürfen diese ihren Wertebereich nicht überschreiten. Chars können außerdem nur Zahlen und keine Buchstaben zugewiesen werden.

Rechenoperationen: Es sind alle Rechenoperationen, die auch der ToyProcessor durchführen kann, erlaubt. Das heißt: +, -, |, &, ^, !, ++, - - . Zu beachten hierbei ist, dass die Operanden einer Rechenoperation immer an Variablen gebunden sein müssen und nicht beliebige Zahlenwerte sein können. Somit wären die Statements links erlaubt und die Statements rechts nicht: (Angenommen, dass die Variablen a und c richtig deklariert wurden.)

```
a = a + b;                    a = a + 4;  # nicht erlaubt
c = !c;                       c = 1 | 0;  # nicht erlaubt
```

Außerdem können keine Klammern verwendet werden und nicht mehrere Operationen in einer Reihe durchgeführt werden. Sollen also mehrere Rechenoperationen auf einer Variablen gemacht werden, müssen diese in ihre Teilschritte heruntergebrochen werden. Beispielsweise müsste der Ausdruck links, wie folgt umgewandelt werden (siehe rechts, nach jedem Semicolon soll dort jeweils eine neue Zeile angefangen werden.):

```
a = b | (c - d + e);    ---->    a = c - d; a = a + e; a = b | a
```

Konstrukte:

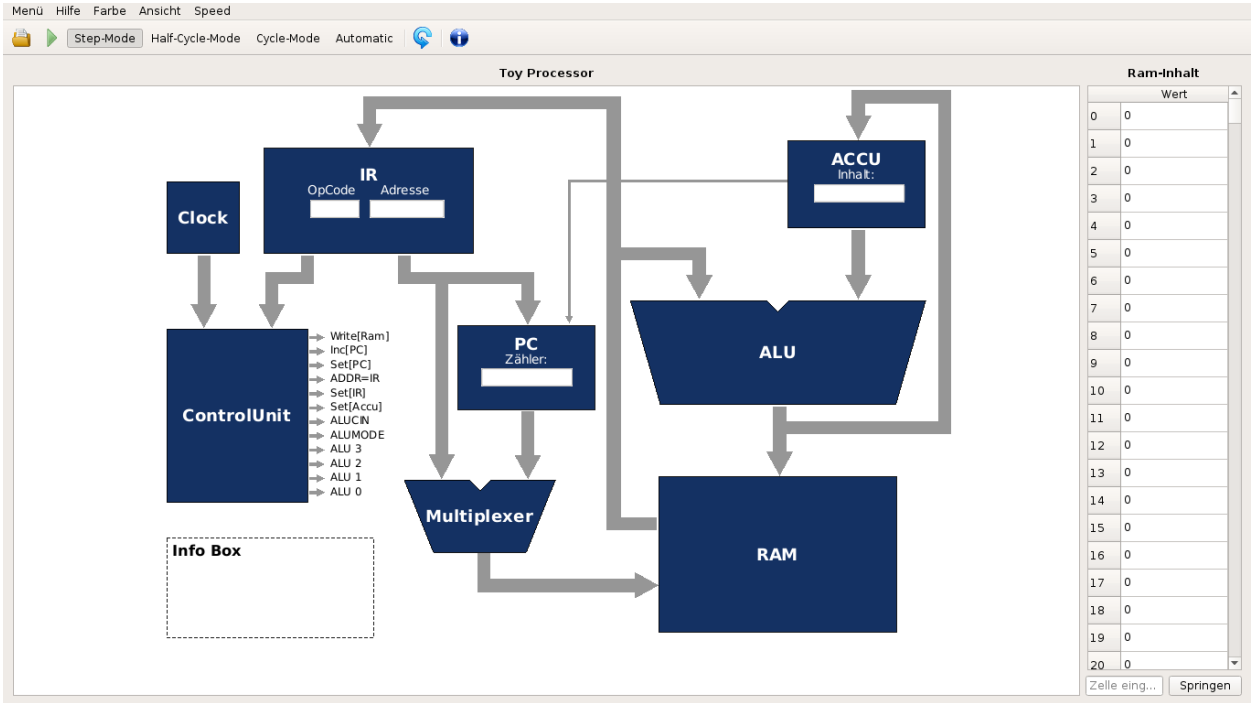
- If-Statements werden analog zum normalen C-Code deklariert. Jedoch können If-Statements in Toy-Programmen nur darauf testen, ob Variablen ungleich Null sind, d.h. if(var != 0) oder if(var).
- While-Schleifen werden genau so, wie in normalem C deklariert. Die Abbruchkonditionen im Pseudo-C-Code darf jedoch nur darauf testen, ob eine Variable ungleich null ist.
- For-Schleifen werden auch analog zum bekannten C-Code geschrieben. Die Laufindexvariablen werden grundsätzlich in der for-Schleife selbst deklariert. Es sind jedoch nur folgende Konstruktionen erlaubt (sei "i" eine undeklarierte Variable, "x" und "y" beliebige Dezimalwerte):
  for(short int i = x; i < y; i++) bzw. for(short int i = x; i > y; i++) bzw. for(short int i = x; i < y; i- -) und for(short int i = x; i > y; i- -)

Weiteres: Im Pseudo-C-Code wird keine Endlosschleife am Ende des Toy-Programmes benötigt, da diese automatisch vom ToyProcessor hinzugefügt wird. Außer den oben beschriebenen C-Konstrukten sind alle anderen Datentypen, Operationen, Funktionen, Library-Elemente etc. nicht erlaubt. Kommentare funktionieren genauso, wie bei der Binär-, Hexadezimal- und Assemblerprogrammierung. Beispiel eines Pseudo-C-Code-Toy-Programm:

```
short int a = 0;
char b = 10;
bool c = 1;

if (c) {
    #Schleife, die a = 10 + 9 + 8 + ... + 1 = 55 rechnet.
    while (b != 0) {
        a = a + b;
        #Schleife, die nur als Demonstration von for-Loops dient.
        for(short int i = 4; i < 6; i++) {
            a++;
            a--;
        }
        b--;
    }
}
```

Bedienung



Das Bild oben zeigt die GUI des ToyProcessors an. In der Mitte ist das Blockschaltbild des ToyProcessors erkennbar. Die Pfeile zwischen den einzelnen Prozessoreinheiten sind noch grau, sie werden erst beim Ausführen eines Programms (genau gesagt beim Ausführen eines Befehls) gefärbt. Die Einzelteile werden wie folgt bedient:

Steuerungsleiste: In der Steuerungsleiste ganz oben befinden sich 8 verschiedene Buttons/Optionen:

- Mit dem Button ganz rechts kann ein Programm in den ToyProcessor geladen werden.
- Der grüne Pfeil daneben dient zum Starten des AutomaticModus bzw. zum Durchklicken in den anderen Modi.
- Mit der Auswahl eines der vier Buttons ”Step-Mode”, ”Half-Cycle-Mode” und ”Cycle-Mode” kann festgelegt werden, wieviele der Einzelschritte eines Befehles (siehe Abbildung 3) gleichzeitig ausgeführt werden.
- Ist ”Automatic” gewählt, dann werden die Befehle nacheinander abgearbeitet, bis das Programm in die Endlosschleife gerät.
- Der Button mit dem gekringelten blauen Pfeil setzt das Programm zurück auf den Anfang des geladenen Programms.
- Durch das Klicken des Buttons ganz rechts kann man die Dokumentation ansehen.

Direktes Manipulieren von Werten: In das Instruction Register (IR), den Programmcounter (PC) und den Accu können direkt Werte geschrieben werden. Dazu dienen die Textfelder in den jeweiligen Bausteinen. Dies ermöglicht es, dass Programm während der Laufzeit zu verändern, indem z.B. durch das Umschreiben des PC-Wertes an eine andere Stelle gesprungen oder im Instruction Register ein neuer Befehl ausgeführt wird. Am Anfang stehen die Werte alle auf Null. Fehlerhafte Eingaben werden abgefangen.

RAM-Anzeige: Ganz rechts wird der Inhalt des RAMs in einer Tabelle angezeigt. Ist noch kein Programm geladen, so sind alle Einträge Null - dies entspricht natürlich nicht der Realität, in der die unterschiedlichsten Werte im Speicher stehen können, bis sie überschrieben werden. Sobald ein Toy-Programm geöffnet wurde, erscheinen die Werte in der Tabelle und das Programm kann ausgeführt werden. Der aktuell ausgeführte Befehl wird grau unterlegt. Alle Werte im RAM können sowohl als Dezimalzahlen, als auch Hexadezimalzahlen dargestellt werden. Umgestellt wird dies im Menü "Ansicht –> Darstellung".

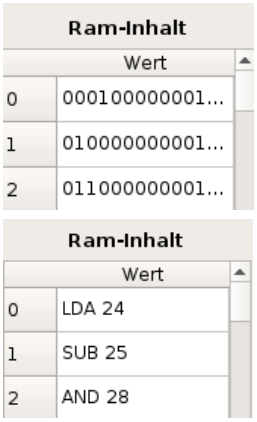


Abbildung 4: Binär- und Mnemonicdarstellung des RAMs

Weitere Einstellungen:

- Unter "Farbe" können die Farben für aktive und inaktive Pfeile beliebig gewählt werden. Als Beispiel siehe die Abbildungen unten.
- Unter "Ansicht" kann entschieden werden, ob die exakten Pfade, die genau zeigen, welche Bausteine in einem Schritt angesprochen werden, oder die ganzen Pfade, die der Realität entsprechen, angezeigt werden. Außerdem kann im Unterpunkt die Werteanzeige in den Bausteinen zwischen den Formaten dezimal, binär, hexadezimal und mnemonic verändert werden.
- Unter "Speed" kann die Schnelligkeit des AutomatikModus gesetzt werden.

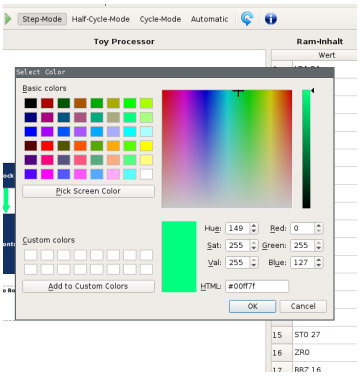


Abbildung 5: Auswahl der Farbe für die Pfeile

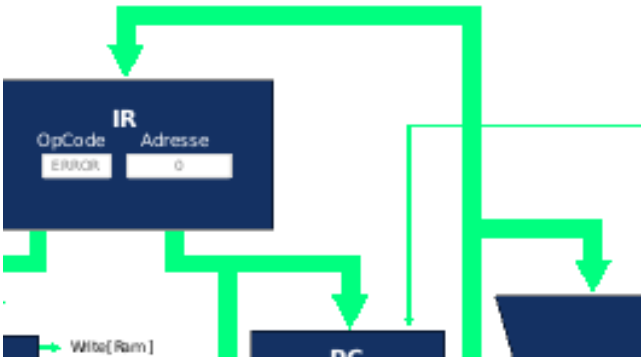


Abbildung 6: Veränderte Pfeile

Menükürzel:

Datei laden	Strg+Umschalt+E	Half-Cycle-Mode	Strg+2
Beenden	Strg+Umschalt+Q	Cycle-Mode	Strg+3
Dokumentation	Strg+I	Automatic	Strg+4
Start/Next Step	Leertaste	Reset	Strg+R
Step-Mode	Strg+1		

Literaturverzeichnis

1. Prof. Dr. Bringmann, Foliensatz 7 ”Register Transfer Ebene” der Vorlesung ”Einführung in die technische Informatik”, 13.11.2013
2. Phil Koopman, ”Microcoded versus hard-wired control”, BYTE, Januar 1987, S. 235, (vgl. www.ece.cmu.edu/ koopman/ - Zugriff am 15.06.2015)
3. Fiete Botschen, ”Toy Processor - Beta 0.4 Documentation”, 17.10.2014
4. https://en.wikipedia.org/wiki/Instruction_register - Zugriff am 14.06.2015
5. www.whatis.techtarget.com/definition/program-counter - Zugriff am 14.06.2015