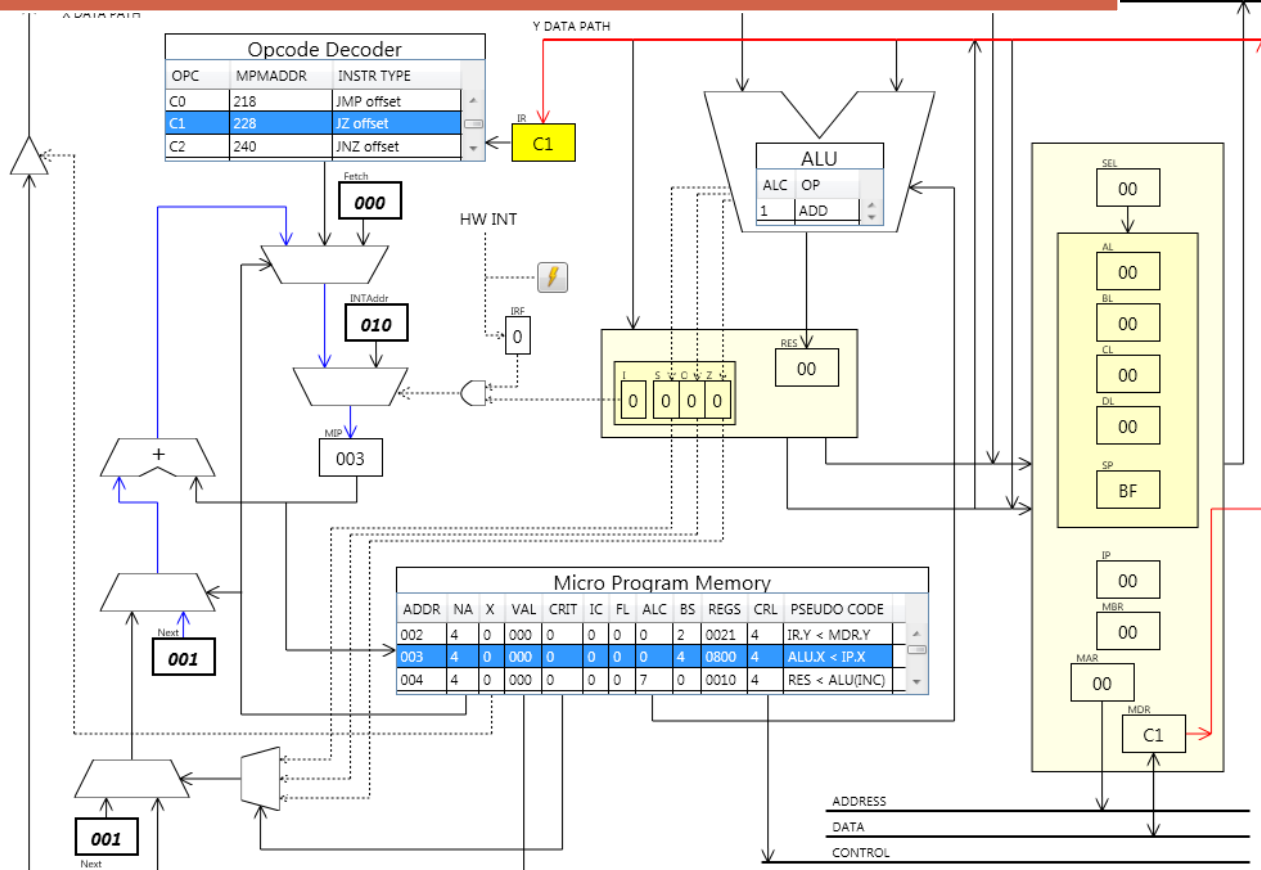


2013

stebs - User Guide



Betreuung:
Ruedi Müller

Auftraggeber:
Simon Felix

Team:
Nicolas Weber, Roman Holzner,
Josiane Manera, Jurij Chamin

Initialisiert:
Ivo Nussbaumer Thomas Schilling
(Bachelor-Thesis Absolventen 2011)

Inhaltsverzeichnis

1	Sinn und Zweck.....	2
2	Installation.....	3
3	Applikation starten	5
4	Fenster Übersicht	6
4.1	Assembler Code Window	6
4.2	Output Window	6
4.3	Registers Window.....	7
4.4	RAM Window	7
4.5	Architecture Window	8
4.5.1	Register.....	9
4.5.2	MIP	9
4.5.3	Micro Program Memory	9
4.5.4	Opcode Decoder	10
4.6	Ribbons.....	10
4.7	Hauptmenu.....	11
4.8	IO-Geräte	12
4.9	Interrupt IO-Gerät	13
5	Layout Manager	14
5.1	Fenster öffnen	14
5.2	Fenster Anordnen	14
5.3	IO-Geräte	14
5.4	Vorgefertigte Layouts	14
5.5	Custom Layouts	15
6	Erste Schritte mit IO-Gerät „Heater“	16
6.1	Assembler-Code vom nachfolgenden Beispiel.....	16
6.2	Assembler-Code Datei öffnen.....	16
6.3	Assembler-Code Assemblieren	17
6.4	Assembler-Code auf verschiedenen Schritt Ebenen erforschen	19
6.5	Auto Run.....	20
7	Spezielles.....	21
7.1	7.1 Parameterübergabe.....	21
7.2	IO-Gerät mehrfach Benutzung.....	21
7.3	IO-Geräte selber programmieren	21
8	Konfiguration.....	22
8.1	Instruktionen und Microcodes	22
9	Tastenkombinationen	24
10	Anhang	25
10.1	Abbildungsverzeichnis	25
10.2	Tabellenverzeichnis	25

1 Sinn und Zweck

An der FHNW erlernen die Informatik-Studierenden im ersten Semester, wie ein Computer im Inneren funktioniert. Da man dies nicht am echten Gerät zeigen kann, weil bekanntlich alles sehr klein ist, braucht man einen Simulator auf Software-Ebene.

Mit „steb“ entstand eine Applikation, mit welcher die grundlegenden Vorgänge, welche in einem Computer ablaufen, erforscht und verstanden werden können. Ebenso können erste Programmiererfahrungen mit einer Assemblersprache gemacht werden.

In diesem „User Guide“ geht es darum den Gebrauch und die Bedienung von „steb“ zu erläutern.

2 Installation

stebis bietet zur einfachen Installation einen Installer. Eine Installation von Hand kann allerdings auch mit Hilfe der Installationsanleitung im separaten Dokument vollzogen werden.

Der Installer benötigt für den Start Administratorrechte. Nach dem Start des Installer, erscheint ein Willkommens-Bildschirm.

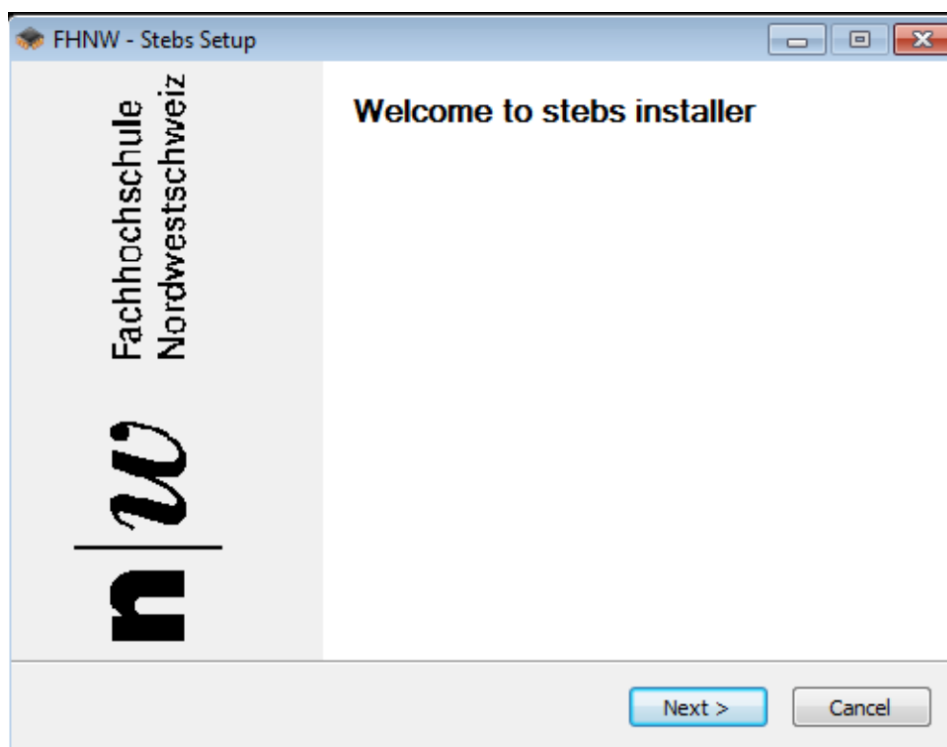


Abbildung 1: Willkommens-Bildschirm

Mit der Taste „Next“ gelangt man zum Auswahlmenu, der zu installierenden Features.

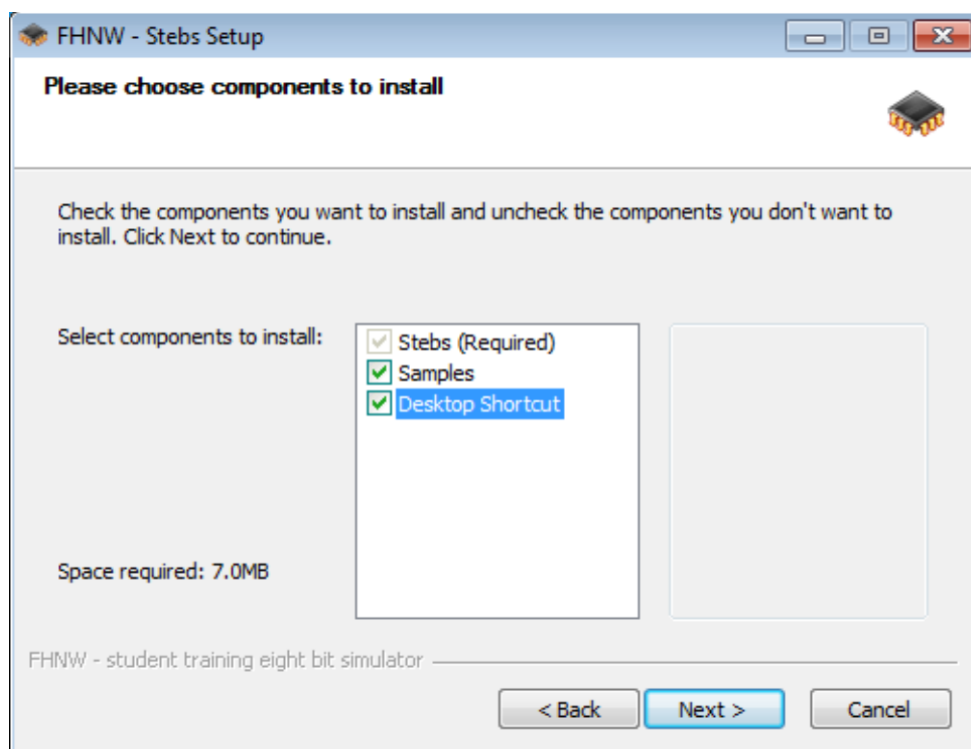


Abbildung 2: Selektion der Features

Bei der Auswahl von Samples, werden Beispielsdateien in den öffentlichen Ordner „C:\Users\Public\Documents\Stebs\Samples“ abgelegt.

Anschliessend kann das Installationsverzeichnis bestimmt werden.

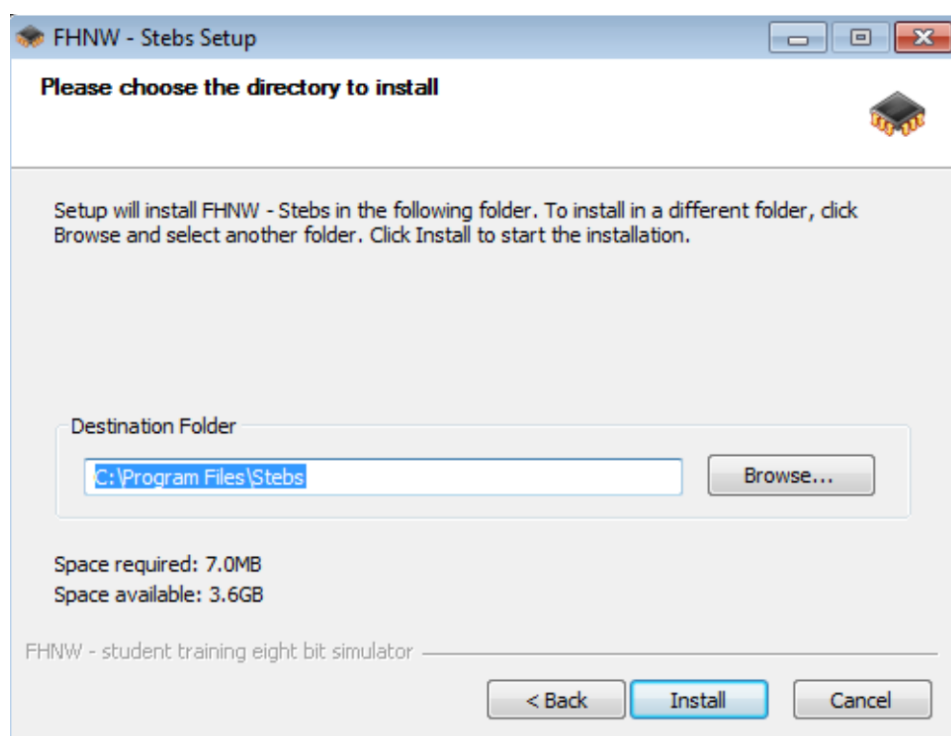


Abbildung 3: Installationsverzeichnis setzen

3 Applikation starten

Die Applikation kann entweder durch den Doppelklick auf die neu erstellte Desktop Verknüpfung (Abbildung 1) oder durch die Auswahl im Startmenü, unter FHNW (Abbildung 2) gestartet werden.



Abbildung 4: Desktopverknüpfung

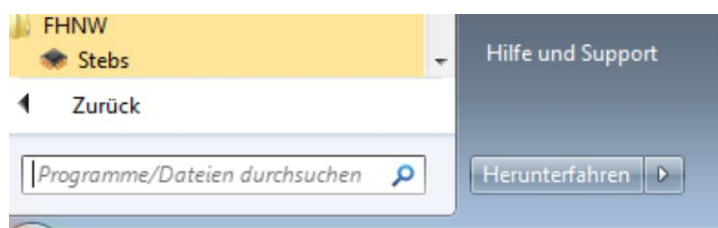


Abbildung 5: Startmenü

4 Fenster Übersicht

4.1 Assembler Code Window

In diesem Fenster wird die aktuell geöffnete Assembler Source Code Datei im Titel des Fensters angezeigt unter „Filename“. Im „Search“ Feld kann man nach einem oder mehreren Suchbegriffen suchen mittels Eingabetaste. Die Suchergebnisse werden gelb markiert (siehe Abbildung 3). Sobald während dem „steppen“ eine neue Instruktion erreicht wird, wird die aktuelle Zeile blau eingefärbt. Ansonsten hilft dieses Fenster beim Programmieren von Assembler mittels Syntax Highlighting und Zeilennummern. Mit dem Slider kann der Zoomfaktor eingestellt werden.

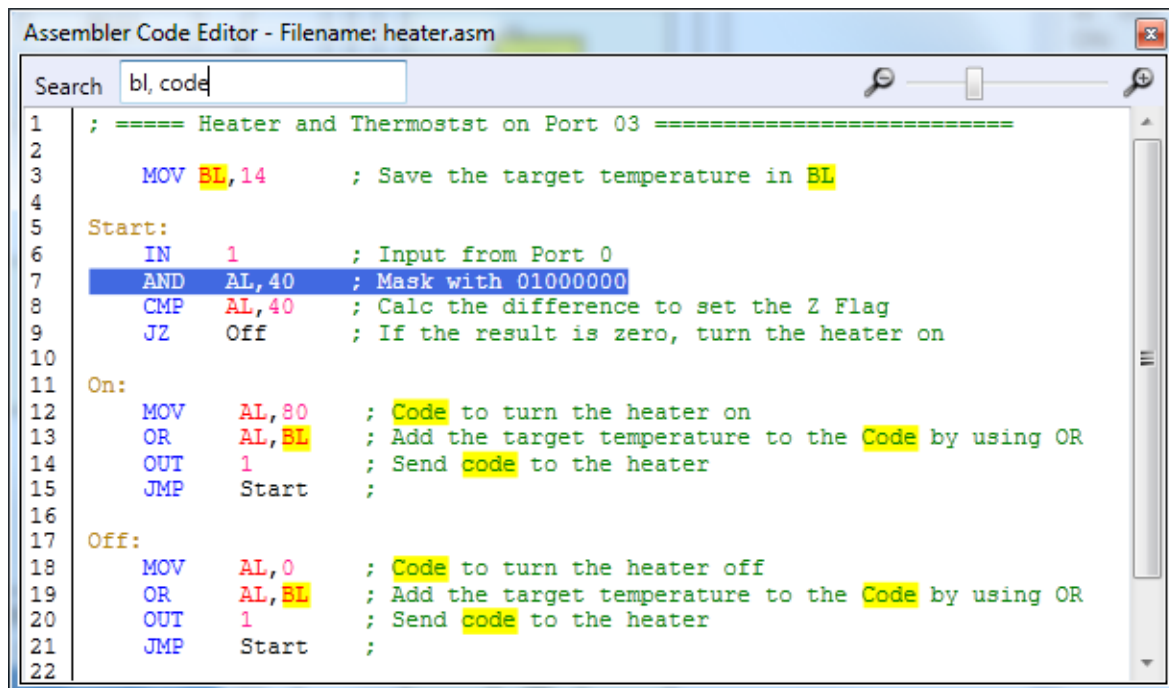


Abbildung 6: Assembler Code Window

4.2 Output Window

Im Output Window werden beim Assemblieren des Assembler-Codes Fehlermeldungen in Rot oder Erfolgsmeldungen in Grün ausgegeben. Meist steht noch auf welcher Zeile sich der Fehler befindet. Das Fenster wird erst beim Öffnen oder dem Erstellen einer neuen Assembler Datei geleert. Ansonsten kann man dies mittels Button „Clear All“ selbständig tun. Mittels Slider kann der Zoomfaktor eingestellt werden, ansonsten bietet dieses Fenster keine anderen Funktionalitäten an.

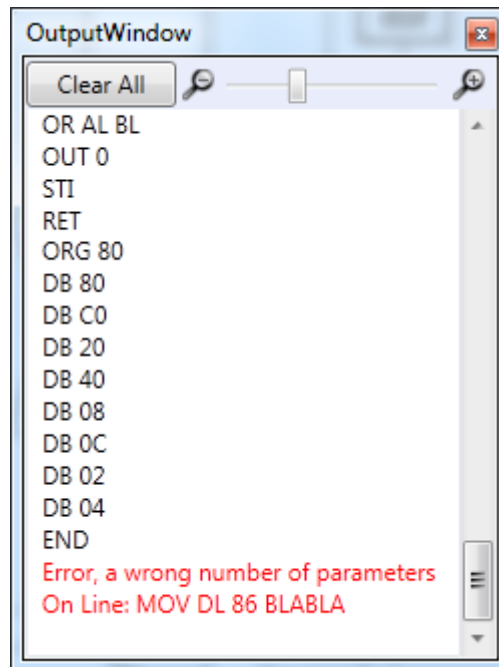


Abbildung 7: Output Window

4.3 Registers Window

Im Registers-Window sind alle programmierrelevanten Register-Werte (AL, BL, CL, DL, SP, IP und SR) in den Zahlensystemen Binär, Zweierkomplement und Hexadezimal ersichtlich. Die beim letzten Schritt geänderten Register Werte werden gelb markiert. Mittels Slider kann der Zoomfaktor eingestellt werden. Die aktuellen Werte der Register können mit Ctrl+C herauskopiert werden.

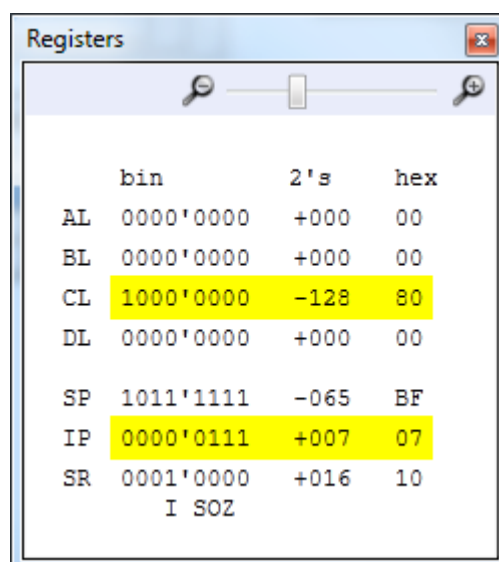


Abbildung 8: Registers Window

4.4 RAM Window

Auf dem RAM Window ist der Inhalt des 256 Bytes grossen Simulator RAMs ersichtlich. Mittels Radiobutton kann man zwischen Hexadezimaler oder ASCII Darstellung wechseln. Der IP (Instruction Pointer) wird rot markiert und der SP (Stack Pointer) wird blau markiert. Mittels Slider kann der Zoomfaktor eingestellt werden. Der Inhalt des RAMs kann mit Ctrl+C oder über das Kontext Menu (Rechtsklick auf RAM-Register) mit „Kopieren“ herauskopiert werden.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	C0	03	40	FC	D0	02	80	D0	03	86	CA	50	DB	02	83	C2
10	07	D0	02	80	C0	04	A4	02	DB	03	87	C2	07	D0	03	84
20	C0	04	A4	03	CA	50	C0	E6	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	D0	00	11	F1	00	CD	00	00	00	00	00	00	00	00	00	00
50	FD	D3	00	02	D3	01	03	AB	00	01	F1	00	FC	CB	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	80	C0	20	40	08	0C	02	04	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	26
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Abbildung 9: RAM Window

4.5 Architecture Window

Im Architecture Window ist die Rechnerarchitektur ersichtlich. In der ALU sind die aktuellen Operation ersichtlich. Die „Micro Program Memory“-Tabelle enthält alle Microcodes. Sie werden je nach Ausführungsschritt markiert. Mithilfe des Sliders kann der Zoomfaktor eingestellt werden.

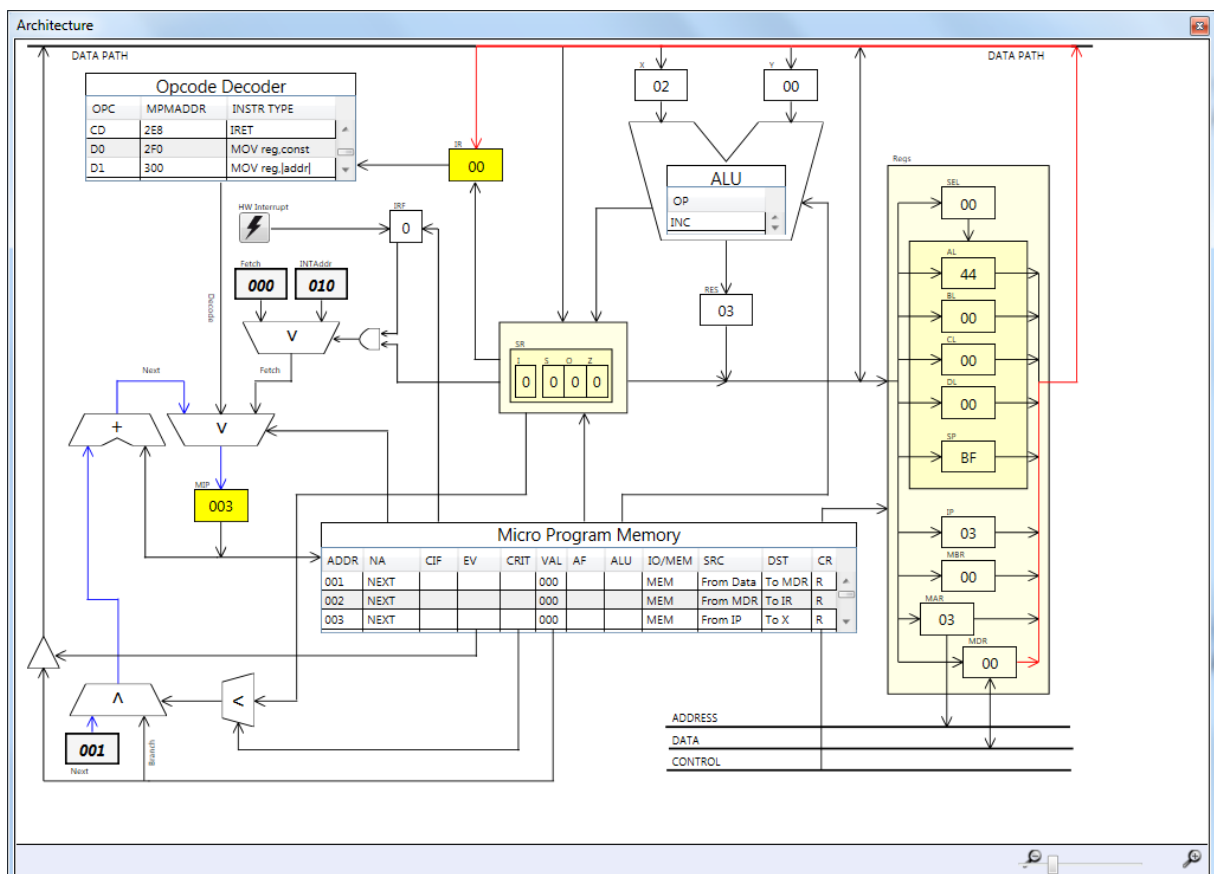


Abbildung 10: Architektur Window

4.5.1 Register

Die Register sind als Rechtecke mit einem Hexadezimalen Wert dargestellt und enthalten immer den aktuellen Wert. Diese enthalten alle einen Tooltip, welcher weitere Informationen zum Register enthält und erscheint sobald man mit dem Cursor auf das Register zeigt. Die Register werden gelb markiert, falls sie im letzten Schritt geändert wurden. Zudem ist durch die rot markierten Pfeile und Linien ersichtlich von welchem Register der Wert kopiert wurde.

Die Rechtecke, bei welchen der Wert fett und kursiv geschrieben ist, repräsentieren konstante Werte, wie z.B. die Microcode Adressen von der Fetch- oder Interrupt-Routine.

4.5.2 MIP

Die blauen Pfeile und Linien visualisieren die Art der Erhöhung des MIP (Micro Instruction Pointer). Der MIP ist nach jedem Schritt gelb markiert, da er auch nach jedem Schritt einen neuen Wert enthält.

4.5.3 Micro Program Memory

Die Tabelle mit dem Micro Program Memory enthält alle Mikroinstruktionen der Assembler Befehle. Diese Daten können über ein externes Excel-Dokument manipuliert werden (siehe Kapitel 8). Es wird immer die aktuelle Instruktion markiert.

Spalte	Beschreibung
ADDR	Microcode Adresse
NA	Definiert wie die nächste MIP Adresse berechnet wird. 1: Fetch Adresse 2: Adresse des nächsten Assembler Befehls aus dem Decoder 4: Nächste Adresse
X	Falls dieses Bit gesetzt ist, wird der Wert der VAL-Spalte auf den X-Bus geschrieben.
VAL	Enthält einen Offset, eine Adresse oder einen anderen Wert.
CRIT	Definiert welches Kriterium bei einem Jump geprüft wird. 1: NZ (Not Zero) 2: NO (Not Overflow) 3: NS (Not Signed) 4: Z (Zero) 5: O (Overflow) 6: S (Signed)
IC	Falls gesetzt wird das IRF-Register (Interrupt Flag) zurückgesetzt.
FL	Falls gesetzt, werden die Werte im Status Register bei einer ALU-Operation gesetzt.
ALC	Definiert welche ALU-Operation durchgeführt wird. 1: ADD (Addition) 9: XOR (Logisches exklusives ODER) 2: SUB (Subtraktion) A: NOT (Logisches NICHT) 3: MUL (Multiplikation) B: AND (Logisches UND) 4: DIV (Division) C: SHR (Rechts Verschiebung) 5: MOD (Modulare Division) D: SHL (Links Verschiebung) 6: DEC (Dekrementation) E: ROR (Rechts Rotation) 7: INC (Inkrementation) F: ROL (Links Rotation) 8: OR (Logisches ODER)
BS	Definiert von welchem Bus geschrieben oder gelesen wird. 1: DATA-Bus 2: Y-Bus 4: X-Bus
REGS	Definiert das Register aus welchem ein Wert kopiert wird, und auch in welches Register der Wert kopiert wird. 0001: to IR 0100: to MAR 0002: from SR 0200: from MBR 0004: to SR 0400: to MBR 0008: from RES 0800: from IP 0010: to RES 1000: to IP

	0020: from MDR	2000: from [SEL]
	0040: to MDR	4000: to [SEL]
	0080: from MAR	8000: to SEL
CRL	Kontrolliert den CONTROL-Bus, bzw. die Art des Zugriffs auf den DATA-Bus. 1: I/O 2: Write 4: Read	
PSEUDO CODE	Beschreibung der Mikro Instruktion als Pseudo Code.	

Tabelle 1: Spalten des Micro Program Memory

4.5.4 Opcode Decoder

Der Opcode Decoder greift auf das IR (Instruction Register) zu und ermittelt mithilfe des Wertes den Assembler-Befehl. Die Assembler-Befehle mit den dazugehörigen Opcodes sind in der Decoder-Tabelle im externen Excel-Dokument definiert. Ebenfalls in dieser Tabelle definiert ist die MPM-Adresse eines Befehls, welche definiert an welcher Adresse im MPM (Micro Program Memory) der Befehl startet.

Spalte	Beschreibung
OPC	Opcode
MPMADDR	Micro Program Memory Adresse
INSTR TYPE	Typ der Instruktion

Tabelle 2: Spalten des Opcode Decoders

4.6 Ribbons

Es gibt drei verschiedene Ribbons: Home, View und Help. Links vom Home Ribbon befindet sich das Hauptmenu.

Auf dem Home-Ribbon werden mittels „Reset“ das RAM, alle Register und der Rechner zurückgesetzt in die Ausgangssituation. Mit „Assemble“ wird der Assemblercode assembliert und in das RAM abgefüllt. Im Bereich „Steps“ macht man Einzel-Schritte in der gewünschten Schritt-Genauigkeit (im Hintergrund werden immer Micro-Steps ausgeführt). Im „Auto Run“ Modus wird mittels „Run“ das Assemblerprogramm abgearbeitet bis „Pause“ gedrückt wird. Danach kann der Auto Run mittels „Continue“ fortgesetzt werden oder mit „Restart“ von vorne abgearbeitet werden. Mit dem „Speed“-Slider wird die Geschwindigkeit des Auto Run Modus festgelegt.

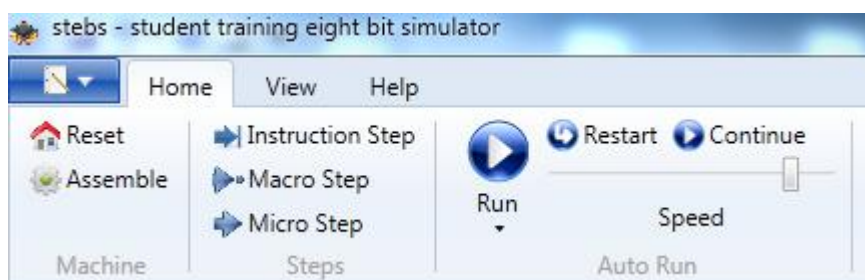


Abbildung 11: Home Ribbon

Auf dem View-Ribbon können im Bereich „Views“ die verschiedenen Windows geöffnet werden. Im Bereich „IO Devices“ kann mit dem „Add New Device“ eine IO-Geräte-DLL von einem beliebigen Ort her geöffnet werden. Alle der Applikation hinzugefügten „IO Devices“ können hier zudem einzeln

sichtbar gemacht werden. Im Bereich „Layout“ kann zwischen sechs vordefinierten Fenster-Darstellungen ausgewählt werden oder mittels „Custom 1-3“ jeweils die aktuelle Fenster-Darstellung gespeichert werden oder geöffnet werden.

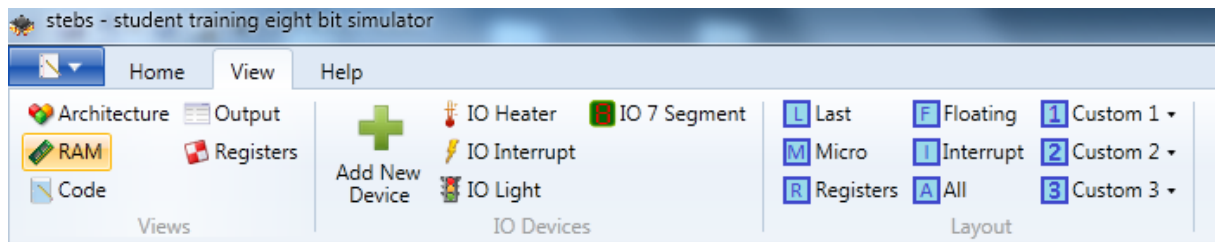


Abbildung 12: View Ribbon

Auf dem Help-Ribbon kann dieser „User Guide“ jederzeit als PDF geöffnet werden mit dem „Help“ Button. Mittels „Hint“ wird ein Dialog mit Tipps geöffnet, auf dem jeweils ein nächster Tipp angezeigt werden kann.



Abbildung 13: Help Ribbon

4.7 Hauptmenu

Im Hauptmenu kann eine Assembler Datei: Erstellt werden, geöffnet werden, gespeichert oder an einem andern Ort gespeichert werden. Zudem kann das Programm mittels Exit beendet werden. In der „Recent Documents“ Liste finden sich die letzten acht Dateien, um diese schnell und bequem per Mausklick zu öffnen.

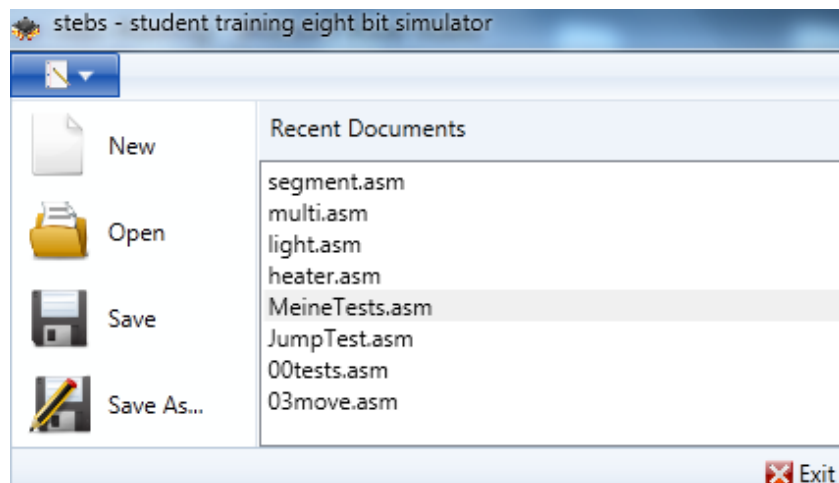


Abbildung 14: Hauptmenu

4.8 IO-Geräte

Auf den mitgelieferten IO-Geräten befindet sich eine Darstellung in welcher je nach gesetztem Wert zur Laufzeit etwas verändert wird. Im Titel sieht man anhand der Nummer in den eckigen Klammern auf welcher Port Nummer sich dieses Gerät befindet.

Mittels Assembler-Befehl „IN 02“ oder „OUT 02“ kann zum Beispiel die 7-Segmentanzeige gesetzt oder abgefragt werden. Das vorderste Bit entscheidet welche Ziffer angesprochen wird, in den hinteren Bits der Zifferwert.

Beim Lichtsignal können einfach alles 8 Lampen ein oder ausgeschaltet werden, wobei die Fussgänger Ampeln selbständig blinken.

Das „Heater“ Window kann zusätzlich mit dem „Reset“ Button auf die Ausgangstemperatur zurückgesetzt werden. Dieses Gerät hat Inputs und Output: Man kann die Heizung an lassen oder abschalten und zusätzlich kann die „Target Temperature“ gesetzt werden. Der „Thermostat“ kann abgefragt werden, ob die Temperatur darüber oder darunter ist. Die „Current Temperature“ sinkt oder steigt kontinuierlich, je nach Heizungszustand.

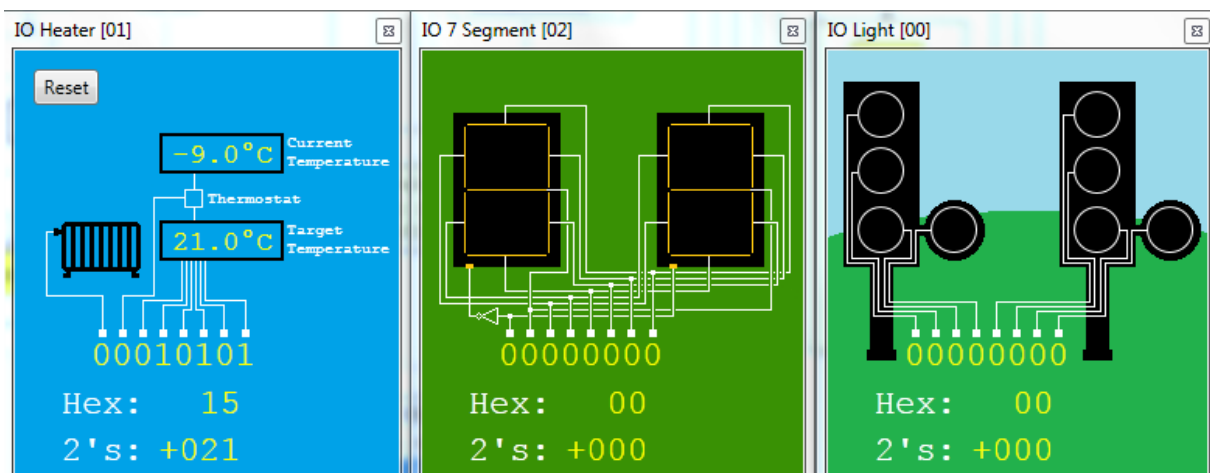


Abbildung 15: IO Geräte

Solche IO-Geräte können auch selber entwickelt werden. Dazu gibt es das separate Dokument „Plugin Entwicklung“.

4.9 Interrupt IO-Gerät

Das IO-Gerät „Interrupt“ welches mitgeliefert wird ist ein Sonderfall und erhält die Port Nummer FF. Mit ihm ist es möglich einzelne Interrupts per Button oder periodische Interrupts an den Simulator zu senden. Dazu wird die Checkbox „Periodic Interrupt“ markiert und mittels Slider die Sekunden ausgewählt. Somit wird immer das IRF (Interrupt Flag) auf der Architektur gesetzt und der Assemblerprogrammierer kann auf dieses Flag reagieren. Wichtig dabei ist, dass der Assembler Programmierer Hardware Interrupts im SR (Status Register) zulässt. Dies wird mit dem Assembler Befehl „STI“ und „CLI“ verwaltet.

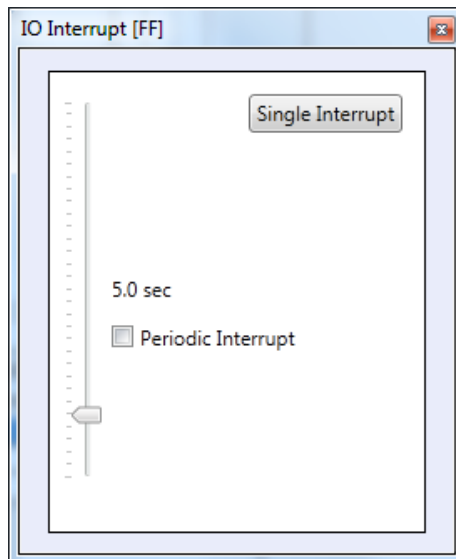


Abbildung 16: Interrupt Window

Der einzelne Interrupt kann beliebig auch direkt auf dem Architecture Window mittels des Buttons mit dem „Blitz“-Icon erzeugt werden.

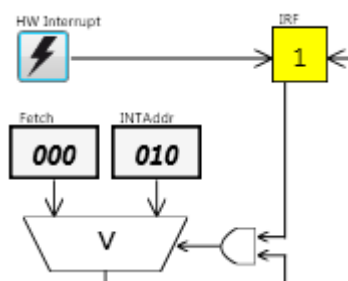


Abbildung 17: Interrupt auf Architecture Window

Der Hardware Interrupt IRF ist nur aktiv wenn im SR (Status Register) das I Flag gesetzt wurde mittels Assembler-Befehl STI (Set Interrupt). Das Gegenstück ist CLI (Clear Interrupt).

5 Layout Manager

5.1 Fenster öffnen

Alle Fenster können über das Ribbon „View“ geöffnet werden, mittels der Kreuz-Schaltfläche oben rechts wieder geschlossen.

5.2 Fenster Anordnen

Ein Fenster kann in drei verschiedenen Arten angezeigt werden.

Art	Beschreibung
Floating	Man kann ein Fenster frei irgendwo auf dem Bildschirm platzieren, indem man es durch klicken und ziehen auf der Titelleiste an einem beliebigen Ort platziert, oder durch den Kontext-Menü Eintrag „Float“.
Auto Hide	Durch klicken des mittleren Buttons oder durch den Kontext-Menü Eintrag „Auto Hide“ kann ein Fenster versteckt werden, so dass nur der Titel am Rand sichtbar bleibt. Durch bewegen des Cursors über den Titel wird das Fenster kurz wieder eingeblendet, aber nach einiger Zeit wieder versteckt.
Docked/ Tabbed	Zieht man ein Fenster auf dem Bildschirm herum, erscheinen an verschiedenen Orten Buttons, wie im nebenstehendem Bild. Zeigt man nun mit dem Cursor auf einen Button, wird dieses blau eingefärbt und im Hintergrund erkennt man an der dunklen Fläche wo das Fenster angedockt werden würde. Wählt man den mittleren Button, wird das Fenster als Tab in diesem Unterfenster angezeigt, so dass man anschliessend zwischen den Tabs wechseln kann.



Abbildung 18: Layout Manager Buttons

Tabelle 3: Arten von Darstellungen der Fenster

5.3 IO-Geräte

Die IO-Geräte werden automatisch geöffnet, falls sie nicht schon offen sind, sobald der Programm-Schritt bei einem Assembler-Befehl „IN“ oder „OUT“ mit der entsprechenden Port-Nummer des IO-Gerätes angelangt ist.

5.4 Vorgefertigte Layouts

Zusätzlich sind noch fünf Layouts vordefiniert, welche durch klicken auf den jeweiligen Button im Ribbon „View“ aktiviert werden können.

Layout	Beschreibung
Last	Beim Beenden des Programms wird das aktuelle Layout hier gespeichert und beim Programmstart wieder geladen.
Registers	In dieser Ansicht ist das Architecture Window ausgeblendet.
Interrupt	In dieser Ansicht sind alle IO Geräte ausgeblendet. Nur das Interrupt Gerät ist eingeblendet.

Floating	In dieser Ansicht sind die IO Geräte und das RAM als Floating Window definiert. So dass sie z.B. auf einem erweiterten Desktop verschoben werden können.
Micro	In dieser Ansicht ist das Registers Window ausgeblendet, da diese Register ebenfalls im Architecture Window ersichtlich sind.
All	In dieser Ansicht sind alle Fenster eingeblendet.

Tabelle 4: Vorgefertigte Layouts

5.5 Custom Layouts

Es ist möglich drei eigene Layouts zu speichern und zu laden. Diese befinden sich auf dem Ribbon „View“ in der Gruppe „Layouts“ und heissen „Custom 1-3“. Um ein neues Layout zu speichern, können die Fenster nach eigenen Wünschen arrangiert werden, und dann durch klicken des Save-Buttons gespeichert werden. Dabei wird das bereits gespeicherte Layout sofort überschrieben.

6 Erste Schritte mit IO-Gerät „Heater“

6.1 Assembler-Code vom nachfolgenden Beispiel

In diesem Kapitel wird immer von folgendem Assembler-Programm ausgegangen:

```
; ===== Heater and Thermostat on Port 01 =====
MOV BL,14      ; Save the target temperature in BL
Start:
IN 01          ; Input from Port 01
AND AL,40      ; Mask with 01000000
CMP AL,40      ; Calc the difference to set the Z Flag
JZ Off         ; If the result is zero, turn the heater on
On:
MOV AL,80      ; Code to turn the heater on
OR AL,BL       ; Add the target temperature to the Code by using OR
OUT 01         ; Send code to the heater
JMP Start      ;
Off:
MOV AL,0       ; Code to turn the heater off
OR AL,BL       ; Add the target temperature to the Code by using OR
OUT 01         ; Send code to the heater
JMP Start      ;
END
; =====
```

Auf dem IO-Gerät „Heater“ kann im vordersten Bit die Heizung ein oder ausgeschaltet werden. Je nach Zustand steigt oder sinkt die Temperatur kontinuierlich. Auf dem zweiten Bit kann der Temperatur-Sensor abgefragt werden, ob die aktuelle Temperatur darüber oder darunter ist. Mit den 6 Bits von rechts wird die gewünschte Temperatur eingestellt im Bereich 0-63 Grad Dezimal oder 0-3F Hexadezimal.

Dieses Programm versucht nun die aktuelle Temperatur, der gewünschten Temperatur, mittels Heizung ein und ausschalten, anzugleichen.

Pro Assemblercode-Zeile darf nur ein Assembler-Befehl stehen. Ob man die Parameter Komma- oder nur mit einem Leerzeichen trennt ist dem Benutzer überlassen. Mittels Semikolon können ganze Kommentarzeilen geschrieben werden oder am Ende einer Programm-Zeile ein Kommentar eingefügt werden.

Dieser Code kann nun im Assembler Code Editor Window mittels „Copy&Paste“ eingefügt werden oder im nächsten Kapitel wird erklärt wie dieser Assembler-Code von einer Datei geöffnet werden kann, welche sich im Unterordner „assembler“ im Applikationsordner befinden sollte.

6.2 Assembler-Code Datei öffnen

Im Hauptmenu wird mittels „Open“ die Assembler-Datei „Heater.asm“ gesucht und geöffnet. Das Assembler Code Editor Window kann geöffnet werden und man sieht den Inhalt der Assembler-Datei.

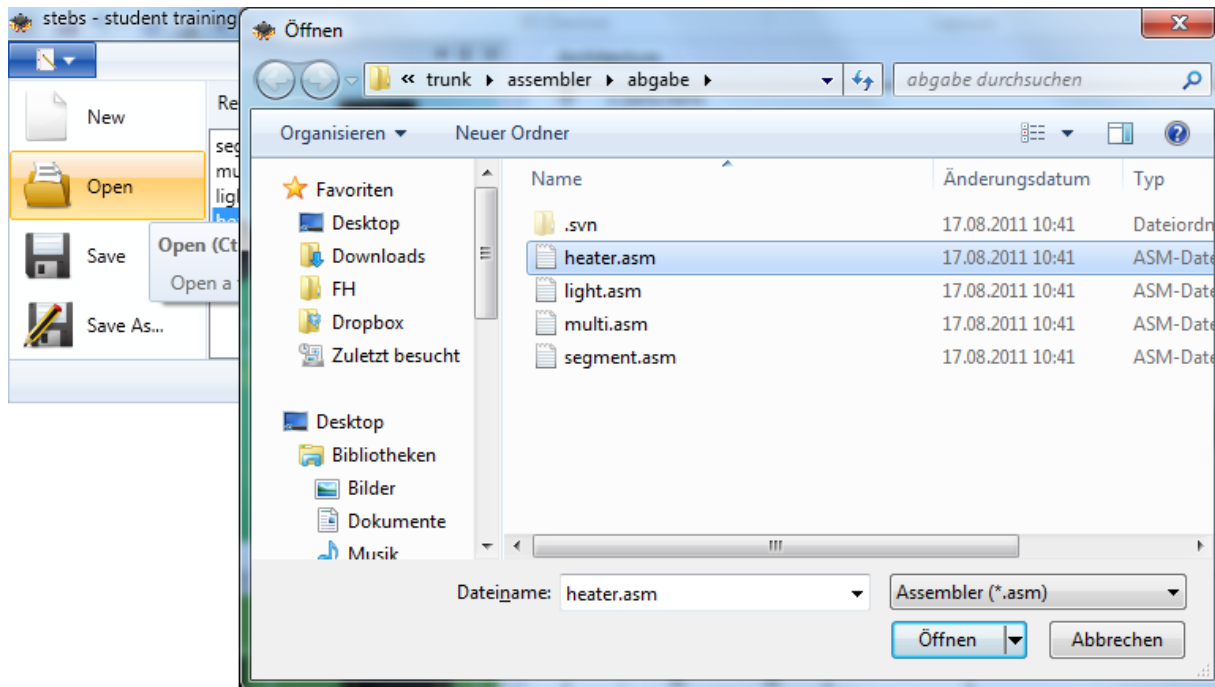


Abbildung 19: Datei öffnen

Für die nächsten Schritte ist es am einfachsten, wenn alle Fenster geöffnet sind. Dazu geht man auf das Ribbon „View“ und klickt auf die Layout-Schaltfläche „All“.

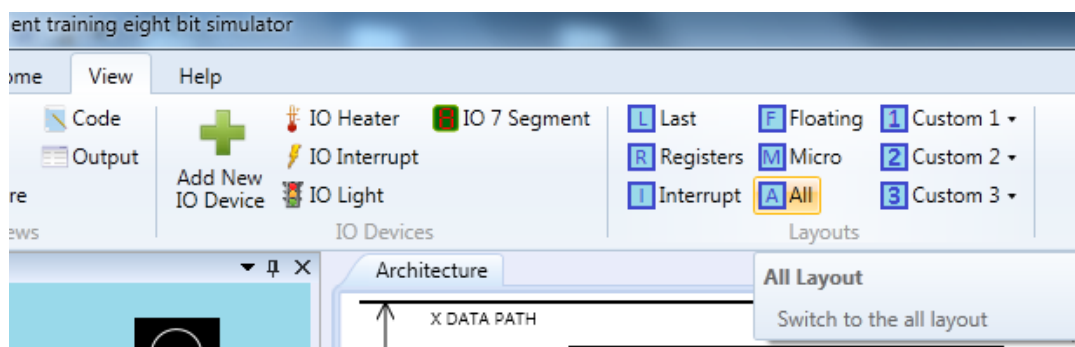
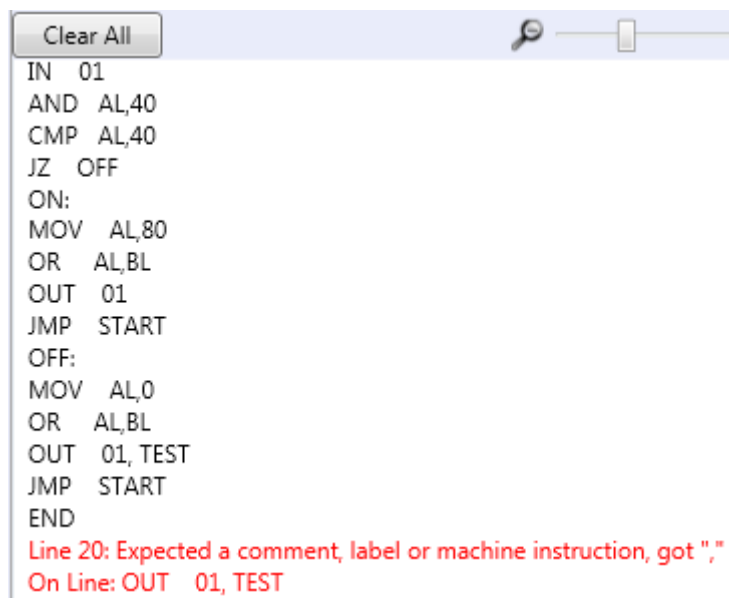


Abbildung 20: All Layout

Als nächstes könnte diese Assembler-Datei bearbeitet werden. Dies ist hier jedoch nicht nötig und wir sehen im nächsten Kapitel wie diese assembliert werden.

6.3 Assembler-Code Assemblieren

Der Assemblercode muss nun assembliert und in das RAM geschrieben werden. Wenn das RAM nach drücken des „Assemble“ Buttons leer bleibt, hat es wohl einen Fehler im Assemblercode und man öffnet am besten das Output Window um die Fehlermeldung zu lesen und den Fehler zu verbessern. (Dieser Fehler ist nicht im Beispiel Assembler-Code von Heater.asm enthalten und somit kommt diese Meldung nicht.)

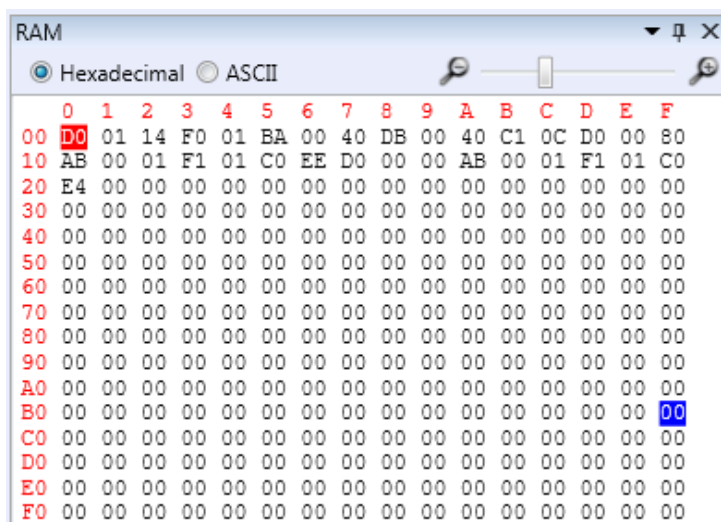


```

Clear All
IN 01
AND AL,40
CMP AL,40
JZ OFF
ON:
MOV AL,80
OR AL,BL
OUT 01
JMP START
OFF:
MOV AL,0
OR AL,BL
OUT 01, TEST
JMP START
END
Line 20: Expected a comment, label or machine instruction, got ","
On Line: OUT 01, TEST
  
```

Abbildung 21: Beispiel einer Fehlermeldung

Wenn der Assemblercode fehlerfrei ist und auf „Assemble“ geklickt wird, erscheint eine Erfolgsmeldung im Output Window in grüner Schrift. Der Code wird nun in das RAM Window geschrieben. Der Simulator ist nun bereit mit verschiedenen Schritten erforscht zu werden, dazu mehr im nächsten Kapitel. Rot ist der aktuelle Fortschritt des Instructionpointers markiert und blau die Position des Stackpointers, welche sich nach dem Reset auf der Position BF befindet.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	D0	01	14	F0	01	BA	00	40	DB	00	40	C1	0C	D0	00	80
10	AB	00	01	F1	01	C0	EE	D0	00	00	AB	00	01	F1	01	C0
20	E4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Abbildung 22: RAM Inhalt von Heater.asm

6.4 Assembler-Code auf verschiedenen Schritt Ebenen erforschen

Wenn man das Architektur Window offen hat, sieht man mit Micro-Steps am detailreichsten was abläuft. Die die aktiven Leitungen werden auf dem Architecture Window markiert, ebenso die verschiedenen betroffenen Register. Mit Macro-Steps werden acht Micro-Steps auf einmal ausgeführt und mit Instruction-Steps wird ein ganzer Assembler-Befehl auf einmal abgearbeitet. (Im Hintergrund läuft alles in Micro-Steps ab).

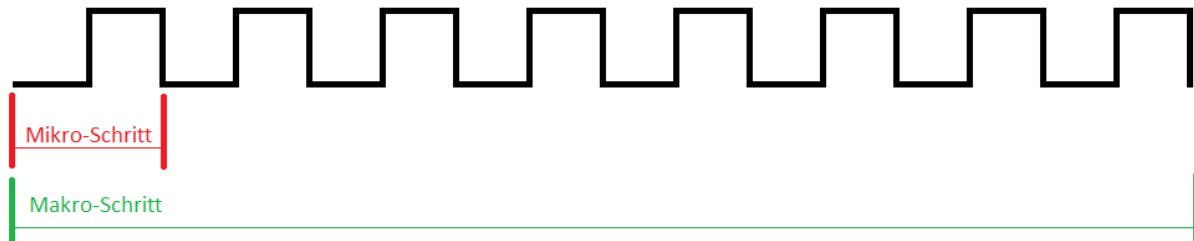


Abbildung 23: Schritte

Im Assembler Code Editor Window sieht man mit blauer Markierung auf welchem Assembler-Befehl man sich gerade befindet.

```

1  ; ===== Heater and Thermostst on Port 01 =====
2
3      MOV BL,14      ; Save the target temperature in BL
4
5  Start:
6      IN    01      ; Input from Port 01
7      AND   AL,40    ; Mask with 01000000
8      CMP   AL,40    ; Calc the difference to set the Z Flag
9      JZ    Off      ; If the result is zero, turn the heater on
10
11  On:
12      MOV   AL,80    ; Code to turn the heater on
13      OR    AL,BL    ; Add the target temperature to the Code by using C
14      OUT   01      ; Send code to the heater
15      JMP   Start   ;
16
17  Off:
18      MOV   AL,0     ; Code to turn the heater off
19      OR    AL,BL    ; Add the target temperature to the Code by using C
20      OUT   01      ; Send code to the heater
21      JMP   Start   ;
22
23
24      END
25  ; =====
  
```

Abbildung 24: Assembler Code Editor auf aktueller Zeile

Zu Beginn des Programms sinkt die Temperatur kontinuierlich, bis man auf die Zeile 14 gelangt, wo die Heizung eingeschaltet wird. Jetzt steigt die Temperatur kontinuierlich an. Um nun die gewünschte Temperatur von 21 Grad zu halten, muss das ganze etwas schneller ablaufen. Dazu eignet sich der „Auto Run“-Modus welcher im nächsten Kapitel beschrieben wird.

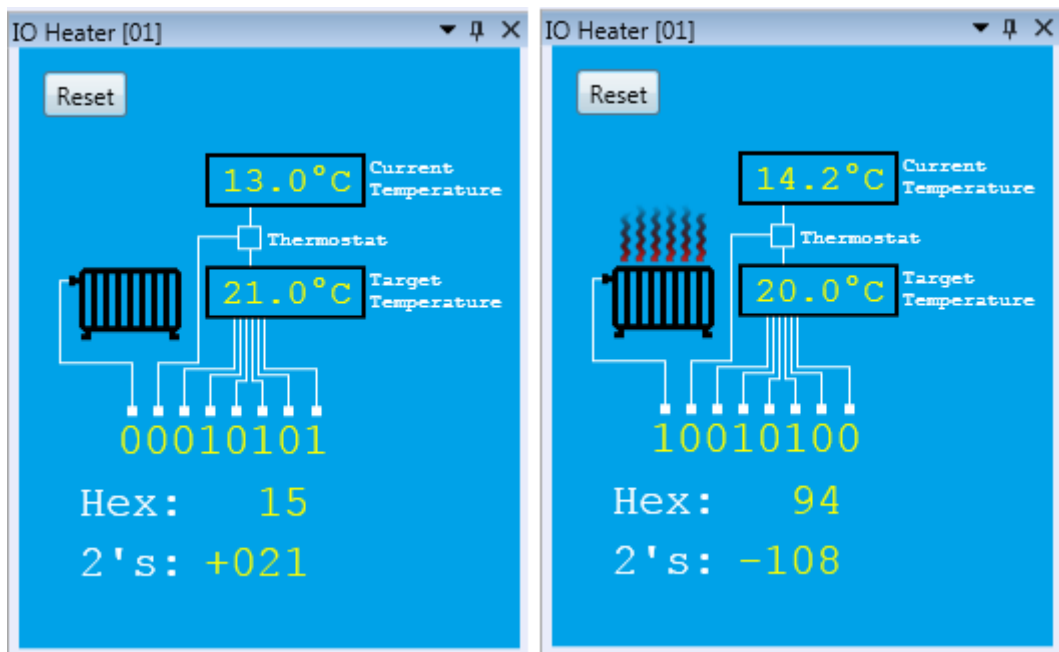


Abbildung 25: Heizung aus und eingeschaltet

6.5 Auto Run

Mittels Klick auf den Button „Run“, kann man das Ganze auch automatisch ablaufen lassen. Man wählt dazu die gewünschte Schritt-Genauigkeit (Instruction, Macro oder Micro) aus und stellt die Geschwindigkeit mittels „Speed“-Slider ein (auch zur Laufzeit möglich).

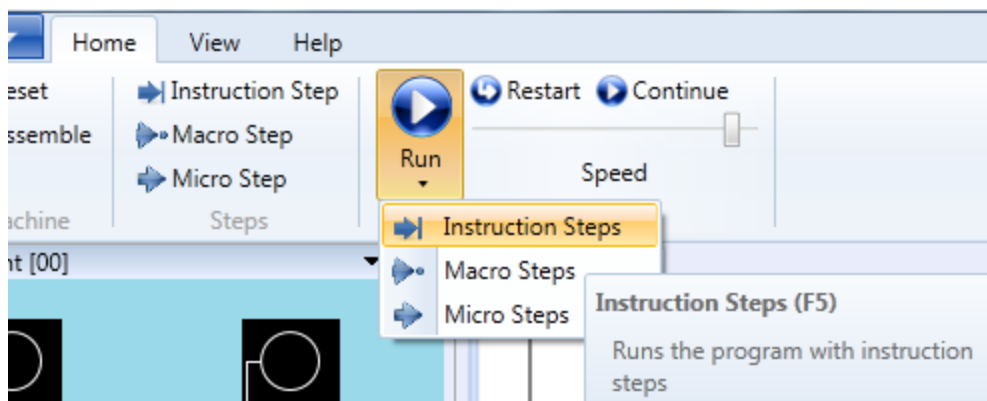


Abbildung 26: Auto Run

Desto schneller man den Auto Run laufen lässt (mit dem Speed-Slider), desto kleiner wird die Temperatur Schwankung. Wenn man den Auto Run in der „Instruction“-Ebene laufen lässt, läuft das Ganze noch etwas schneller, als wenn man ihn in der „Micro“-Ebene laufen lässt da deutlich weniger visualisiert werden muss.

7 Spezielles

7.1 7.1 Parameterübergabe

Es kann an eine laufende Instanz ein Parameter, eine ASM Datei, übergeben werden. Diese Datei wird vom Stebs geladen und anschliessend automatisch assembliert. Falls bereits eine Datei geladen und in Bearbeitung ist, wird der Benutzer aufgefordert diese Datei zu speichern oder zu verwerfen, bevor die übergebene Datei geladen wird.

Der Befehl lautet: `steb.exe datei.asm`

Der Pfad zur Datei kann entweder relativ oder auch absolut sein.

7.2 IO-Gerät mehrfach Benutzung

Dasselbe IO-Gerät kann, dank eines Plug-In Systems, auch mehrmals geöffnet und verwendet werden. Einerseits kann das Gerät temporär hinzugefügt werden, indem das Gerät nochmals durch „Add New IO Device“ auf dem Ribbon „View“ hinzugefügt wird.

Es kann aber auch permanent hinzugefügt werden, indem man das Gerät, also die DLL Datei, in den „plugin“-Ordner kopiert. Falls dasselbe Gerät zweimal vorkommen soll, gibt man ihm einfach einen anderen Dateinamen. Danach ist ein Neustart des Programmes erforderlich.

7.3 IO-Geräte selber programmieren

IO-Geräte können, wie in dem separaten Dokument „Plugin Entwicklung“ erklärt, selber in C# programmiert werden und danach in „steb“ verwendet werden.

8 Konfiguration

8.1 Instruktionen und Microcodes

Jede Instruktion mit der im Assemblercode programmiert werden kann, besteht aus ein bis vier Macro-Steps und diese wiederum aus acht Micro-Steps. Die Instruktionen können über eine externe Excel-Datei „Microcode V4.0.xlsm“ verändert werden. Im Excel-Sheet „Decoder Table“ können neue Instruktionen definiert werden. Die genaue Beschreibung der Instruktion wird in einem Sheet mit dem jeweiligen Instruktionsnamen festgehalten.

Microcode V4.0.xlsm

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	MOV reg,const				Opcode: 0xD0				Microcode Addr: 0x2F0				last change: version 4.0		
2	Microcode Address (ADDR)	Next MIP Address (NA)	Clear Interrupt Flag (CIF)	Enable Value (EV)	Offset, Address or Value (VAL)	Affect Flag (AF)	Flag Criterion (CRIT)	ALU operation (ALU)	Destination Register (DEST)	Source Register/Data Bus (SRC)	R/W	IO/MEM	Pseudo Code	Description	
3	FETCH													FETCH	
4	OP													OPERAND	
5	2F0	Next							to MAR	from IP	R	MEM	MAR < IP	put IP value onto address bus	
6	2F1	Next							to MDR	from DATA	R	MEM	MDR < DATA	read register address from data bus	
7	2F2	Next							to SEL	from MDR	R	MEM	SEL < MDR	select register	
8	2F3	Next							to X	from IP	R	MEM	X < IP	feed IP value to X	
9	2F4	Next						INC	to RES	from IP	R	MEM	RES < ALU(INC)	increment IP value	
10	2F5	Next							to IP	from RES	R	MEM	IP < RES	update IP	
11	2F6	Next									R	MEM	nop	perform nop	
12	2F7	Next									R	MEM	nop	perform nop	
13	EXE													EXECUTE	
14	2F8	Next							to MAR	from IP	R	MEM	MAR < IP	put IP value onto address bus	
15	2F9	Next							to MDR	from DATA	R	MEM	MDR < DATA	read constant from data bus	
16	2FA	Next							to [SEL]	from MDR	R	MEM	[SEL] < MDR	move constant into register	
17	2FB	Next							to X	from IP	R	MEM	X < IP	feed IP value to X	
18	2FC	Next						INC	to RES	from IP	R	MEM	RES < ALU(INC)	increment IP value	
19	2FD	Next							to IP	from RES	R	MEM	IP < RES	update IP	
20	2FE	Next									R	MEM	nop	perform nop	
21	2FF	Fetch									R	MEM	MIP < FetchAddr	load fetch address into MIP	

Abbildung 27: Instruktionstyp MOV mit seinen Micro-Steps (von Adresse 2F0 bis 2FF)

Die Instruktionen können im Sheet „Readme“ mittels der Funktion „Export“ exportiert werden.

ROM1 Groups	Abbreviation	Dropdown Items	Individual value	Number of bits	Range	Compacted to Hex Cipher/s
Microcode Address	ADDR	"Hex value"	000 .. FFF	12	000 .. FFF	3 Nibbles (Hex)
Next MIP Address	NA	NEXT	3	2	0 .. 3	1 Nibble (Hex)
		DECODE	2			
		FETCH	1			
Clear Interrupt Flag	CIF		1	1	0, 1	
		CLEAR	1			
		"Empty"	0			
Enable Value	EV		1	1	0, 1	
		ENABLE	1			
		"Empty"	0			
Offset, Address or Value	VAL		000 .. FFF	12	000 .. FFF	3 Nibbles (Hex)
		"Hex value"	000 .. FFF			
Affect Flag	AF		1	1	0, 1	1 Nibble (Hex)
		AFFECT	1			
		"Empty"	0			

Abbildung 28: Exportfunktion der Instruktionen

Dabei werden im gleichen Verzeichnis wie die Excel-Datei die Dateien ROM1.data, ROM2.data und INSTRUCTION.data angelegt. Die Dateien müssen in das Verzeichnis von „C:\Users\<BENUTZERNAME>\AppData\Local\StebS\res“ kopiert werden und werden zu Programmstart von stebs einmalig eingelesen.

9 Tastenkombinationen

Folgende Tastenkombinationen können in der gesamten Applikation „stebs“ verwendet werden.

Tastekombination	Aktion
Ctrl+S	Datei speichern (die aktuelle Assembler Datei)
Alt+S	Datei Speichern unter
Ctrl+N	Neue Datei erstellen
Ctrl+O	Datei öffnen
Ctrl+C	Kopieren vom RAM oder Registerwerten
Alt+F4	Programm beenden
F1	Help per PDF Datei
F2	Reset von Ram und Registern
F3	Auto Run: Restart
F4	Assemblieren des Source Codes
F5	Auto Run: Instruction
F6	Auto Run: Macro
F7	Auto Run: Micro
F8	Auto Run: Pause / Continue
F9	Instruction Step
F10	Macro Step
F11	Micro Step
F12	Hint (Tipps) im Programm

Tabelle 5: Tastenkombinationen

10 Anhang

10.1 Abbildungsverzeichnis

Abbildung 1: Willkommens-Bildschirm	3
Abbildung 2: Selektion der Features	4
Abbildung 3: Installationsverzeichnis setzen	4
Abbildung 4: Desktopverknüpfung	5
Abbildung 5: Startmenü	5
Abbildung 6: Assembler Code Window	6
Abbildung 7: Output Window	7
Abbildung 8: Registers Window	7
Abbildung 9: RAM Window	8
Abbildung 10: Architektur Window	8
Abbildung 11: Home Ribbon	10
Abbildung 12: View Ribbon	11
Abbildung 13: Help Ribbon	11
Abbildung 14: Hauptmenu	11
Abbildung 15: IO Geräte	12
Abbildung 16: Interrupt Window	13
Abbildung 17: Interrupt auf Architecture Window	13
Abbildung 18: Layout Manager Buttons	14
Abbildung 19: Datei öffnen	17
Abbildung 20: All Layout	17
Abbildung 21: Beispiel einer Fehlermeldung	18
Abbildung 22: RAM Inhalt von Heater.asm	18
Abbildung 23: Schritte	19
Abbildung 24: Assembler Code Editor auf aktueller Zeile	19
Abbildung 25: Heizung aus und eingeschaltet	20
Abbildung 26: Auto Run	20
Abbildung 27: Instruktionstyp MOV mit seinen Micro-Steps (von Adresse 2F0 bis 2FF)	22
Abbildung 28: Exportfunktion der Instruktionen	23

10.2 Tabellenverzeichnis

Tabelle 1: Spalten des Micro Program Memory	10
Tabelle 2: Spalten des Opcode Decoders	10
Tabelle 3: Arten von Darstellungen der Fenster	14
Tabelle 4: Vorgefertigte Layouts	15
Tabelle 5: Tastenkombinationen	24