Kurzanleitung für den VIP-Simulator

Teil 2: VIP-Erweiterung um Stack-Pointer, Mikroprogrammierung

Diese Anleitung ist als Ergänzung zur bereits verteilten Kurzanleitung (Seiten 1 bis 7) zu den Grundfunktionen des VIP-Simulators anzusehen.

6 Zusätzliche Kommandozeilenparameter

Zusätzlich zu den bereits bekannten Kommandozeilenparametern `–d` und `–e` unterstützt der VIP-Simulator die Folgenden:

- `-x`
- `-m`
- `-p <mikroprogrammdatei>`.

Die Option `–x` schaltet im VIP-Simulator die SP-Version (Stack-Pointer-Version) des VIP frei, die in Abschnitt 7 beschrieben wird. Diese kann dann im Einstellungen-Fenster aktiviert und deaktiviert werden (siehe Abschnitt 7.1).

Eine ähnliche Funktion besitzt auch die Option `-m`. Ist diese angegeben, so wird in der Menüleiste des Hauptfensters des VIP-Simulators das Menü „Mikroprogramm“ aktiviert. Seine Optionen werden in Abschnitt 8.3 behandelt.

Die Option `–p` schließlich ermöglicht es, bei Programmstart des VIP-Simulators das Mikroprogramm aus einer Datei einzulesen. Ohne diese Option verwendet der VIP-Simulator ein vorgegebenes Mikroprogramm. Weitere Information über die Mikroprogrammdateien finden sich ebenfalls in Abschnitt 8.3.

7 Die SP-Version des VIP-Simulators

Bei der SP-Version wird das Strukturbild des VIP um ein SP-Register (**S**tack **P**ointer) und die zugehörigen Leitungen und Steuersignale erweitert. Dies sind im einzelnen

- die Verbindungen des SP-Registers mit dem Adressbus,
- die Verbindungen des PC-Registers mit dem Datenbus, sowie
- die Steuersignale y_d und y_e.

Außerdem müssen die Steuersignale y_9 und y_a um jeweils einen Bit erweitert werden, um die neuen Möglichkeiten der entsprechenden Multiplexer ansteuern zu können. Das neue Strukturbild wird auch im Strukturbildfenster (Abbildung 8) angezeigt.

Das zusätzliche Register ermöglicht die Verwaltung eines einfachen Stacks im Speicher des VIP. Dazu sind allerdings einige neue Assemblerbefehle nötig. Diese sind normalerweise im Mikroprogramm des VIP **nicht** implementiert! Sie müssen vom Benutzer implementiert werden, oder es wird bei Programmstart mit Hilfe der Option `–p` ein Mikroprogramm geladen, welches die zusätzlichen Befehle enthält.

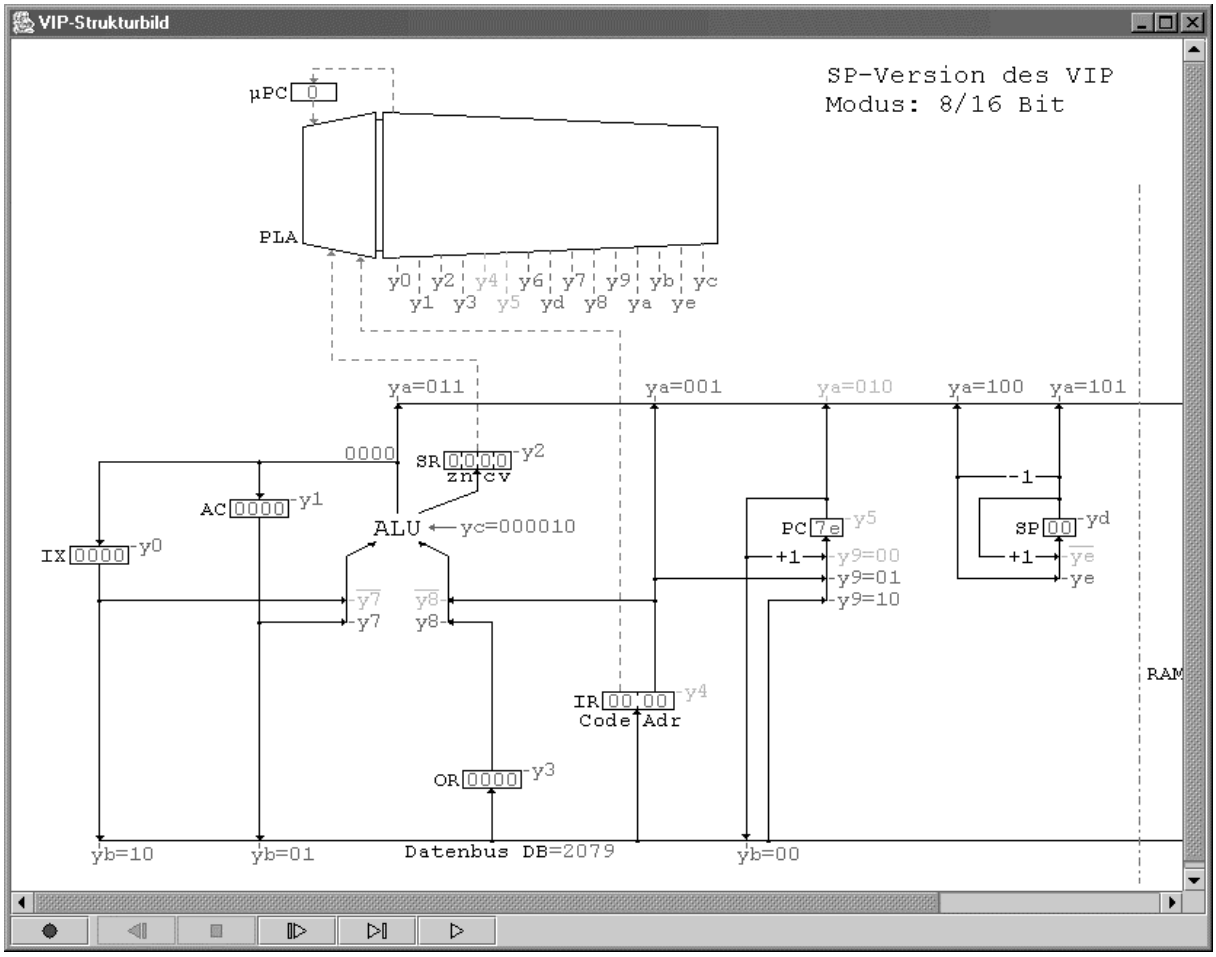


Abbildung 8: Strukturbildfenster bei aktivierter SP-Version

7.1 Aktivierung der SP-Version

Im Einstellungen-Fenster des VIP-Simulators wurde durch den Kommandozeilenparameter `-x` die Option „SP-Version des VIP aktivieren“ freigeschaltet. Diese Option bestimmt also die Version des VIP und ist mit den beiden Modi des VIP („8/16 Bit“ und „24/32 Bit“) kombinierbar, so dass insgesamt vier Konfigurationsmöglichkeiten des VIP im VIP-Simulator existieren.

Es ist natürlich möglich, dass die SP-Version durch eine in der Kommandozeile angegebene Einstellungendatei aktiviert wird. Ist aber dabei die Option `-x` nicht gesetzt, so wird die Version des VIP ohne Warnung zurückgesetzt. Dadurch kann eventuell eine an der Kommandozeile angegebene Mikroprogrammdatei ebenfalls nicht geladen werden (mehr dazu in Abschnitt 8.3), worüber der Benutzer allerdings informiert wird.

Die Version des VIP kann jederzeit geändert werden, allerdings muss dabei das aktuelle Assemblerprogramm vom VIP-Simulator neu übersetzt werden. Bei Erfolg wird es wieder in den VIP geladen.

8 Das Mikroprogramm des VIP

Über das durch die Option `-m` freigegebene Menü „Mikroprogramm“ erhält der Benutzer Zugriff auf das Mikroprogramm des VIP im VIP-Simulator. Die Option „Mikroprogramm

anzeigen“ blendet die neuen Reiter „PLA-Tabelle“ und „Code-Tabelle“ in der Tabulatorleiste ein. Diese beiden Tabellen stellen das Mikroprogramm des VIP dar.

8.1 Die Code-Tabelle

In der Code-Tabelle (siehe Abbildung 9) wird jeder (Assembler-)Anweisung ein 8 Bit breiter Code zugewiesen. Dabei erhält eine Anweisung für jede von ihr unterstützte Adressierungsart einen separaten Eintrag. Es ist zu empfehlen, sich dabei an die Konvention zu halten, dass die Adressierungsart in den untersten zwei Bits des Code kodiert wird, allerdings schreibt das der VIP-Simulator nicht vor.

Die Adressierungsarten werden direkt hinter der Anweisung durch die Zeichen „#“ für die Direktoperand-, „@“ für die speicherindirekte und „`[IX]`“ für die indizierte Adressierung angegeben. Für die Angabe der speicherdirekten Adressierung wird kein Zeichen angefügt. Um zu signalisieren, dass eine Anweisung gar keinen Operanden erwartet, wird das Minuszeichen („`-`“) verwendet.

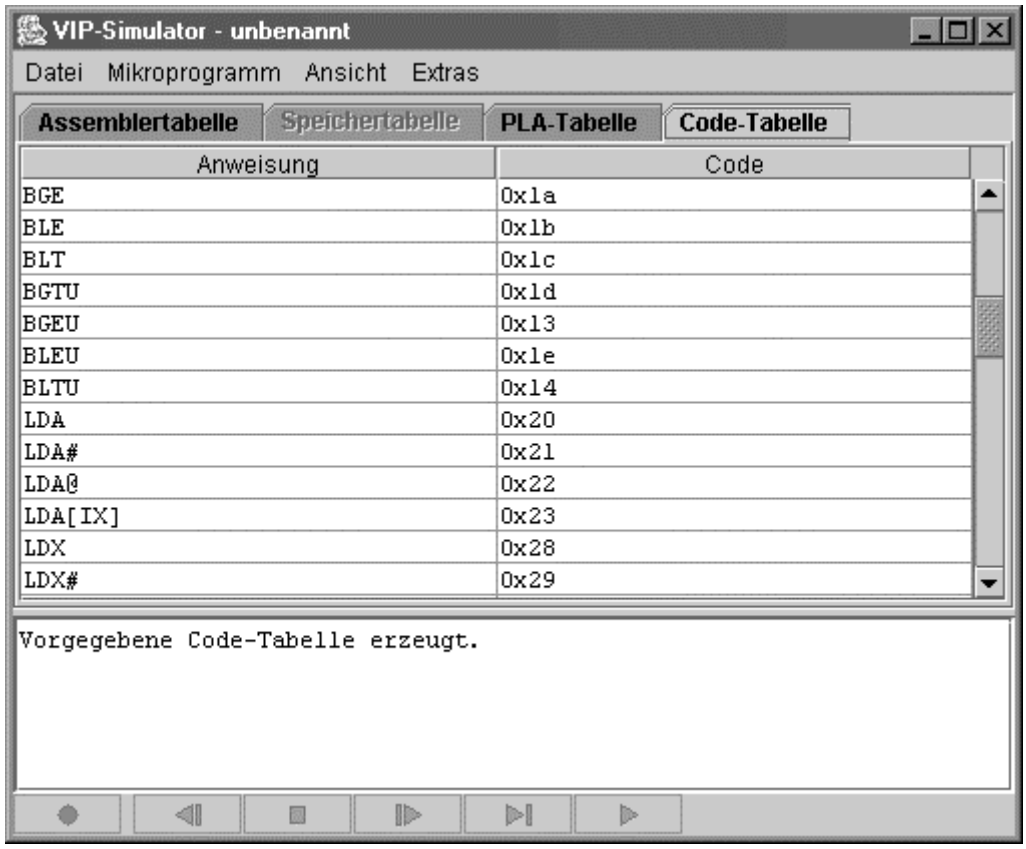


Abbildung 9: Die Code-Tabelle des VIP-Simulators

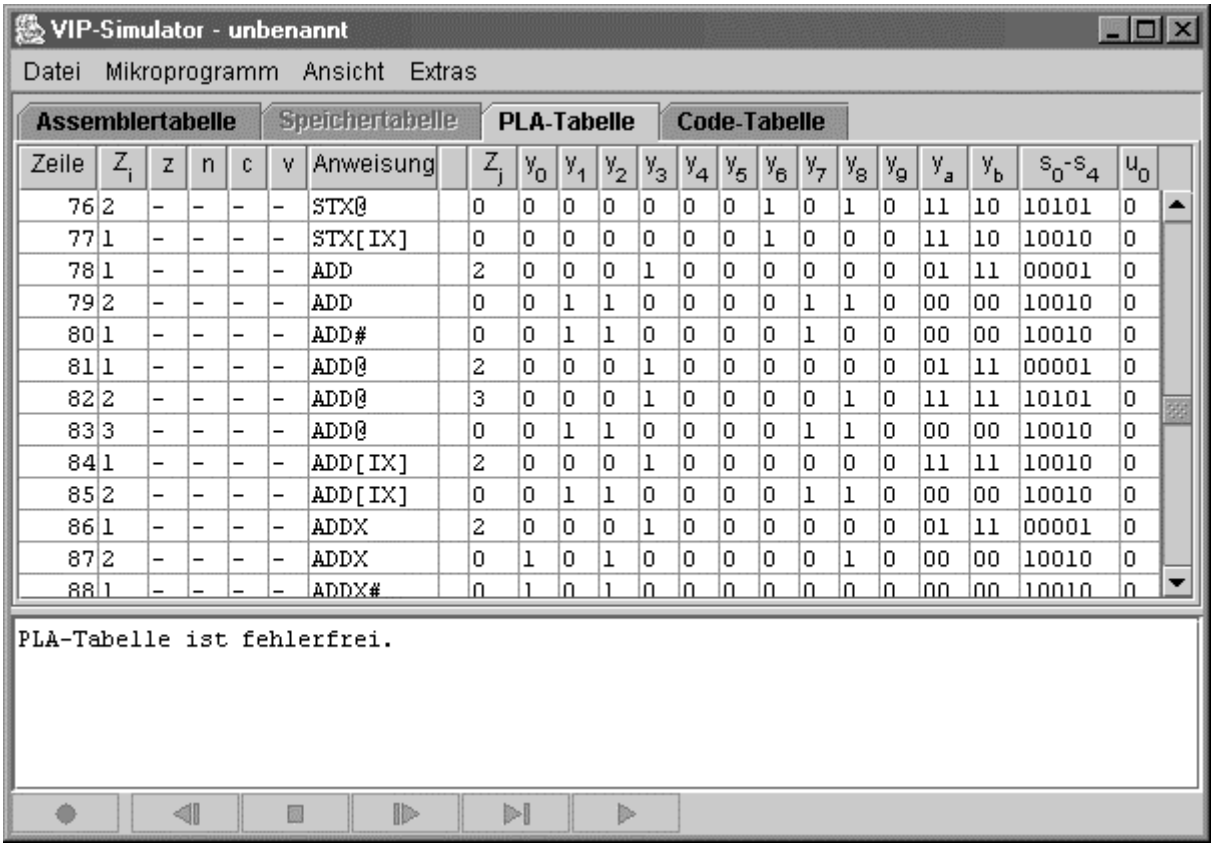
Die Einträge in der Code-Tabelle können genauso wie in der Assemblertabelle editiert werden. Auch hier können Zeilen mit der Taste „Einfg“ eingefügt und mit „Entf“ gelöscht werden.

Jede Anweisung muss in der Tabelle einmalig sein. Doppelte Einträge werden beim Beenden der Eingabe erkannt und nicht übernommen. In der Code-Spalte dagegen sind doppelte Einträge möglich, so dass Synonyme für eine Anweisung möglich sind (wie das zum Beispiel bei den Anweisungen „BCC“ und „BGEU“ mit dem Code „0x13“ der Fall ist).

8.2 Die PLA-Tabelle

Die in Abbildung 10 gezeigte Ansicht der PLA-Tabelle im VIP-Simulator entspricht ziemlich genau der aus der Vorlesung bekannten PLA. Für jede Kombination der Eingangssignale aus den Registern µPC (Z_i), SR (z, n, c und v) und IR.Code (Anweisung) werden die Pegel der Steuersignale und der Folgezustand (Z_j) angegeben.

Z_i und Z_j werden dezimal angegeben, wobei Werte von „`0`“ bis „`255`“ angenommen werden. Bei den Statusbits sind nur die Einträge „`0`“ und „`1`“ erlaubt. Die Anweisung muss einem Eintrag aus der Spalte „Anweisung“ in der Code-Tabelle entsprechen. Außerdem ist in allen Eingangsspalten bis auf Z_i das Minuszeichen „`-`“ erlaubt, um den Fall „don’t care“ zu symbolisieren.



Zeile	Z_i	z	n	c	v	Anweisung	Z_j	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_a	y_b	s_0-s_4	u_0
76	2	-	-	-	-	STX@	0	0	0	0	0	0	0	1	0	1	0	11	10	10101	0
77	1	-	-	-	-	STX[IX]	0	0	0	0	0	0	0	1	0	0	0	11	10	10010	0
78	1	-	-	-	-	ADD	2	0	0	0	1	0	0	0	0	0	0	01	11	00001	0
79	2	-	-	-	-	ADD	0	0	1	1	0	0	0	0	1	1	0	00	00	10010	0
80	1	-	-	-	-	ADD#	0	0	1	1	0	0	0	0	1	0	0	00	00	10010	0
81	1	-	-	-	-	ADD@	2	0	0	0	1	0	0	0	0	0	0	01	11	00001	0
82	2	-	-	-	-	ADD@	3	0	0	0	1	0	0	0	0	1	0	11	11	10101	0
83	3	-	-	-	-	ADD@	0	0	1	1	0	0	0	0	1	1	0	00	00	10010	0
84	1	-	-	-	-	ADD[IX]	2	0	0	0	1	0	0	0	0	0	0	11	11	10010	0
85	2	-	-	-	-	ADD[IX]	0	0	1	1	0	0	0	0	1	1	0	00	00	10010	0
86	1	-	-	-	-	ADDX	2	0	0	0	1	0	0	0	0	0	0	01	11	00001	0
87	2	-	-	-	-	ADDX	0	1	0	1	0	0	0	0	0	1	0	00	00	10010	0
88	1	-	-	-	-	ADDX#	0	1	0	1	0	0	0	0	0	0	0	00	00	10010	0

Abbildung 10: Die PLA-Tabelle des VIP-Simulators

Auch in der PLA-Tabelle können Zeilen eingefügt und gelöscht werden. In den meisten Spalten werden unsinnige Eingaben automatisch rückgängig gemacht. Trotzdem fehlerhafte Zeilen (z. B. nach dem Laden des Mikroprogramms aus einer Datei) werden im Statusbereich gemeldet und bei der Ausführung von Assemblerprogrammen schlicht ignoriert.

Semantische Fehler, wie während der Ausführung nicht erreichbare Zeilen, Endlosschleifen im Mikroprogramm etc. werden vom VIP-Simulator natürlich nicht erkannt. Außerdem sollte darauf geachtet werden, dass die Bedingungen in der PLA-Tabelle eindeutig sind, da der VIP-Simulator bei der Auswahl einer passenden PLA-Zeile nichtdeterministisch vorgeht, das heißt, es ist nicht vorhersagbar, welche Zeile zuerst geprüft wird[1].

[1] In der Hardware werden alle PLA-Zeilen gleichzeitig überprüft. Dies ist in der Software nicht möglich, und eine Simulation dieses Verhaltens wäre laufzeitaufwändiger als die implementierte sequentielle Lösung.

8.3 Das Menü „Mikroprogramm“

In dem Menü „Mikroprogramm“ sind außer der bereits weiter oben erwähnten Option „Mikroprogramm anzeigen“ die vier Optionen „Öffnen...“, „Speichern“, „Speichern unter...“ und „Zurücksetzen“ enthalten.

Die ersten drei Optionen verhalten sich genau wie die gleichnamigen Optionen im Menü „Datei“, nur dass jeweils das Mikroprogramm in eine ASCII-Datei gespeichert bzw. aus einer Datei eingelesen wird. Die Option „Zurücksetzen“ stellt das vorgegebene Mikroprogramm des VIP-Simulators wieder her. Der Benutzer erhält immer, wenn Änderungen an dem aktuellen Mikroprogramm durch eine der Optionen verloren gehen könnten, die Gelegenheit, die Änderungen zu sichern.

Ein Mikroprogramm im ASCII-Format besteht immer aus zwei Dateien: eine PLA-Datei (Endung „.pla“) und eine Code-Datei (Endung „.oct“, **O**p**C**ode-**T**abelle). Beim Laden und Speichern des Mikroprogramms wird immer nur die PLA-Datei angegeben, die Code-Datei wird immer implizit geöffnet beziehungsweise erzeugt. Dabei wird zur Erzeugung des Dateinamens der Code-Datei die Endung „.pla“ durch „.oct“ ersetzt.

Der jeweilige Aufbau der Mikroprogrammdateien kann eingesehen werden, nachdem das vorgegebene Mikroprogramm des VIP-Simulators gespeichert wurde. Im Wesentlichen werden die Spalten der Code-Tabelle durch einen Doppelpunkt getrennt, die Spalten der PLA-Datei durch Tabulatoren oder Leerzeichen. Kommentarzeilen werden in beiden Dateien durch das Zeichen „#“ eingeleitet.

Da sich die Anzahl der Steuersignale je nach gerade verwendeter Version des VIP unterscheidet, gibt es auch zwei verschiedene Formate für die PLA-Datei. Das jeweilige Format einer PLA-Datei wird vom VIP-Simulator anhand der ersten vier Bytes der Datei erkannt. Im Falle der Standard-Version des VIP sind das die Zeichen „`#pla`“, bei der SP-Version die Zeichen „`#plx`“.

Kurzanleitung für den VIP-Simulator

Anhang: Funktionalität der ALU

Tabelle 1: Funktionalität der im VIP-Simulator implementierten ALU

				arithmetische Operationen	logische Operationen
s_0	s_1	s_2	s_3	$s_4 = 0$	$s_4 = 1$
0	0	0	0	$Z = -1 + u_0$	$Z = 0$
0	0	0	1	$Z = X \vee Y + u_0$	$Z = \overline{X \vee Y}$
0	0	1	0	$Z = X \vee \overline{Y} + u_0$	$Z = \overline{X} \wedge Y$
0	0	1	1	$Z = X + u_0$	$Z = \overline{X}$
0	1	0	0	$Z = X \wedge \overline{Y} - 1 + u_0$	$Z = X \wedge \overline{Y}$
0	1	0	1	$Z = (X \vee Y) + (X \wedge \overline{Y}) + u_0$	$Z = \overline{Y}$
0	1	1	0	$Z = X - Y - 1 + u_0$	$Z = X \not\equiv Y$
0	1	1	1	$Z = (X \wedge \overline{Y}) + X + u_0$	$Z = \overline{X \wedge Y}$
1	0	0	0	$Z = (X \wedge Y) - 1 + u_0$	$Z = X \wedge Y$
1	0	0	1	$Z = X + Y + u_0$	$Z = X \equiv Y$
1	0	1	0	$Z = (X \vee \overline{Y}) + (X \wedge \overline{Y}) + u_0$	$Z = Y$
1	0	1	1	$Z = (X \wedge Y) + X + u_0$	$Z = \overline{X} \vee Y$
1	1	0	0	$Z = X - 1 + u_0$	$Z = X$
1	1	0	1	$Z = X \cdot 2^{-1}$	$Z = X \vee \overline{Y}$
1	1	1	0	$Z = X \triangleright 1$	$Z = X \vee Y$
1	1	1	1	$Z = X \cdot 2 + u_0$	$Z = -1$

Anmerkungen:

1. Die Funktionen, die von der Vorgabe im Skript der Vorlesung abweichen, sind grau unterlegt.
2. Der Operator $\triangleright$ soll eine Verschiebung der Bits des ersten Operanden um Anzahl des zweiten Operanden Stellen nach rechts symbolisieren, wobei die „nachrückenden“ oberen Bits mit Nullen gefüllt werden.