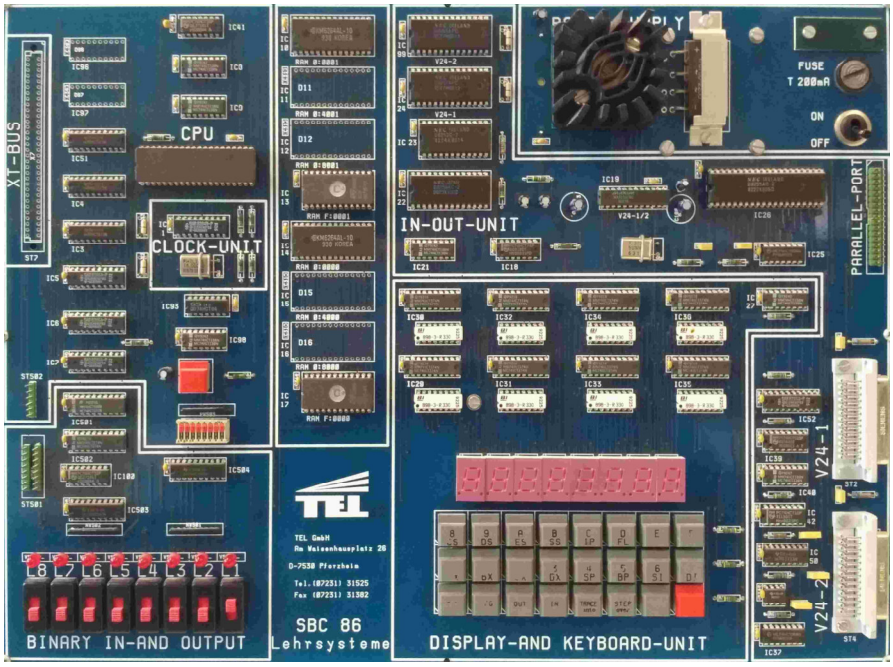
Assemblerprogrammierung mit dem Singleboard-Computer SBC86

Inhaltsverzeichnis:

1. Vorbemerkung

Der SBC86 ist ein Singleboard-Computer auf Basis der CPU Intel 8086. Die Hardware wird über die serielle Schnittstelle mit einem PC verbunden. Ein integriertes Systemprogramm realisiert den Datenaustausch und stellt grundlegende Möglichkeiten zum Austesten der Programme bereit.

Auf dem PC erfolgt die Entwicklung der Programme (Editieren und Assemblieren), auf dem SBC86 werden diese dann abgearbeitet und getestet.



In aktuellen Betriebssystem-Versionen wird der realisierte serielle Datenaustausch nicht mehr unterstützt, Hardware und Systemprogramm des Singelboardcomputers werden jetzt emuliert und stehen so weiterhin zur Verfügung.

2. Die Assemblersprache

2.1. Allgemeines

Die Assemblersprache ist eine symbolische Schreibweise für die Darstellung von Maschinenbefehlen, ein Assemblerbefehl entspricht genau einem Maschinenbefehl. Die Umsetzung eines Assemblerprogrammes in den Maschinencode übernimmt der sogenannte Assembler.

Die möglichenAssemblerbefehle werden durch den Befehlssatz eines konkreten Prozessors vorgegeben, die anderen Elemente der Assemblersprache sind prozessorunabhängig.

Assembler arbeiten zeilenorientiert. Sie unterscheiden i.d.R. bei Symboldefinitionen zwischen Groß- und Kleinschreibung, ignorieren dies jedoch für Assemblerbefehle.

2.2. Zahlen und Zeichen

Zahlen können im Assemblerprogramm als ganze Binär-, Dezimal- oder Hexadezimalzahlen angegeben werden. Sie müssen, außer Dezimalzahlen (Standardeinstellung), entsprechend gekennzeichnet sein. Dies erfolgt durch einen nachgestellten Kennbuchstaben. Steht auf der höchstwertigen Stelle einer Hexadezimalzahl A-F, so ist ihr weiterhin eine Null voranzustellen. Eine Zahl muss immer mit einer Dezimalziffer beginnen.

```
mov al,1110b      ;Binärzahl (0Eh, 15d)
mov ax,1110       ;Dezimalzahl (456h, 100 0101 0110b)
mov ax,1110d      ; …
mov ax,1110h      ;Hexadezimalzahl (4368d)
mov al,0ffh       ; … muss immer mit einer Dezimalziffer beginnen
mov al,0xff       ; … C-Syntax
```

Zeichen und Zeichenketten (Strings) werden durch einfache oder doppelte Hochkommas begrenzt:

```
mov al, 'A'          ;Zeichen (41h)
mov al, "a"          ;Zeichen (61h)
text: equ "Abcd"     ;Zeichenkette (41h,62h,63h,64h)
```

Symbole (Label, Marken, Namen) können Buchstaben, Ziffern oder die nachfolgenden Sonderzeichen _ $ # @ ~ . ? enthalten. Sie müssen immer mit einem Buchstaben, _ oder ? beginnen. Reservierte Worte (alle Befehle, Registerbezeichnungen und Anweisungen) dürfen nicht verwendet werden. Jedes Symbol darf nur einmal definiert werden.

2.3. Assemblerbefehle

Assemblerbefehle entsprechen Prozessorbefehlen und bilden nach der Assemblierung das ausführbare Programm (Maschinenprogramm, Objektcode), welches von der Ziel-CPU abgearbeitet werden kann. Alle Angaben in Klammern sind optional:

[Marke:] [Befehl] [Zieloperand,Quelloperand] [;Kommentar] ↵

Marke: Eine Marke (Label) ist ein Symbol gefolgt von einem Doppelpunkt. Die Marke repräsentiert die Speicheradresse, auf dem der folgende Befehl steht und kann von Sprungbefehlen als Operand benutzt werden (Sprungziel).

Befehl Ein Befehl aus dem Befehlsvorrat des Prozessors, für den das Programm geschrieben wird. Die Schreibweise ist der Befehlsliste zu entnehmen.

Operand(en) Je nach Art des Befehls werden keine, ein oder zwei Operand(en) benötigt. Zwei Operanden sind durch ein Komma zu trennen. Der Zieloperand steht vor dem Komma, der Quelloperand danach.

; Kommentar Erkennt der Assembler ein Semikolon, wird der Rest der Zeile als Kommentar gewertet. Kommentare sind in Assemblerprogrammen fast die einzige Möglichkeit, diese lesbar zu gestalten ...

2.4. Assembleranweisungen, Symbol- und Datendefinitionen

Assembleranweisungen (Pseudobefehle, Assemblerdirektiven) steuern die Arbeitsweise des Assemblers bei der Übersetzung des Quellprogrammes und erzeugen keine ausführbaren Befehle. Sie sind auch erforderlich, um Konstanten zu vereinbaren, Speicherplätze zu reservieren, Adressen zuzuweisen und dienen der Vereinfachung der Assemblerprogrammierung:

```
[Name:]   [Anweisung] [Operand(en)]  [;Kommentar]↵
```

Die Angaben in Klammern sind optional. Für Operanden und Kommentar gilt das bereits bei den Assemblerbefehlen gesagte.

Name: Einige Anweisungen benötigen einen Namen, bei anderen ist er wahlfrei.

Anweisung Eine Anweisung aus dem Vorrat des Assemblers. Assembleranweisungen können sich je nach verwendetem Assembler unterscheiden. Die Schreibweise ist der jeweiligen Dokumentation zu entnehmen.

```
;Ein Beispiel mit dem im Praktikum verwendeten Voreinstellungen
            org 100h        ;Festlegung der aktuellen Adresse im Codeabbild,
                            ;Startadresse des Propgramms
start:      in al,0         ;Einlesen der Schalter nach Register AL
            out 0,al        ;Ausgabe Inhalt AL nach den LED´s
            jmp start       ;ständig wiederholen
```

Die Assembleranweisungen zur Symboldefinition weisen Ausdrücken einen Namen zu, der dann im Quellprogramm anstelle der Ausdrücke benutzt werden kann. Ausdrücke können alphanumerische Ausdrücke, Adreßangaben, Register und Befehle sein.

```
name equ ausdruck

; Beispiele für Symbolzuweisungen
zahl:       equ 55h
leds:       equ 0
zeitkonst:  equ 8000h
            mov al,zahl       ;  der Assembler ersetzt die ver wendeten Symbole
            out leds,al       ;  durch die zugewiesenen Ausdrücke
            mov cx,zeitkonst
```

Datendefinitionen dienen dem Reservieren von Speicherplatz, insbesondere der Variablenvereinbarung und -initialisierung. Der Assembler stellt u.a. zur Verfügung:

```
[name:]   db ausdruck     ; ein Byte wird reserviert
[name:]   dw ausdruck     ; zwei Byte, ein Wort
[name:]   dd ausdruck     ; vier Byte, ein Doppelwort
```

<name> steht für die erste Speicheradresse, über ihn kann auf den reservierten Speicherplatz zugegriffen werden. Der reservierte Speicherplatz kann gleichzeitig initialisiert werden.

```
; Beispiele für Datendefinitionen
negativ:    db -1           ; negative Zahl, entspricht 0FFh
maske:      db 01011100b    ; Binärzahl
zeichen:    db 'A'          ; Zeichen im ASCII, entspricht 41h
wort:       dw 1234h        ; 2 Byte
doppel:     dd 12345678h    ; 4 Byte
```

Werden in einer Datendefinition mehrere Werte, durch Komma getrennt, angegeben (Initialisierungsliste), erhält man ein Array des gewählten Datentyps. Es können soviele Zeichen angegeben werden, wie in eine Zeile passen. <name> steht für die Adresse des ersten Wertes.

```
Tabelle:    db 3fh,6,5bh,4fh,66h,6dh,7dh,7,7fh,6fh
array1:     dw 0,1,2,3,4,5      ; ein Array vom Typ Word
array2:     dw 0,1,2            ; das gleich Array,
            dw 3,4              ;  nur die Schreibweise
            dw 5                ;  ist eine andere
```

Zeichenketten werden mit der DB-Anweisung definiert. Für jedes (im ASCII kodierte) Zeichen wird ein Byte im Speicher benötigt. Auch hier steht der Bezeichner für die Adresse des ersten Zeichens.

```
text1:      db 'Das ist ein Text'
text2:      db 'D','a','s',' ','a','u','c','h'
string:     db 'Text und Steuerzeichen',0dh,0ah,00
```

2.5. Operatoren

Um das Schreiben von Assemblerprogrammen zu erleichtern, erlauben Assembler die Benutzung von Operatoren. Man kann Ausdrücke bilden, die als Operanden in einem Befehl oder einer Anweisung verwendet werden können. Die Ausdrücke werden während der Übersetzung ausgewertet und/oder berechnet und das Ergebnis anstelle des Ausdruckes in das Programm eingefügt.

Welche Operatoren in welcher Schreibweise genutzt werden können, hängt vom verwendeten Assembler ab und ist der entsprechenden Dokumentation zu entnehmen. Mindestens lässt ein Assembler die Benutzung der vier Grundrechenarten zu.

```
; Beispiel für die Operatoren + - * /
anzahl:     equ 4*1024          ; 4 KByte
anfang:     equ 0c00h
ende:       equ anfang+laenge-1
start:      mov cx,anzahl/2
            mov bx,anfang
            mov al,-1
            mov [bx],al
            mov bx,tabelle      ; Startadresse und
            mov cx,laenge       ; Länge der Tabelle
            …
            jmp $               ; Endlosschleife
                                ; ($ : Adresse dieses Befehls bzw. der Anweisung)
tabelle:    db 3fh
            db 5bh
            ...
laenge:     equ $-tabelle       ; Länge der Tabelle
```

3. Programmentwicklung im Praktikum

Im Praktikum wird die folgende Entwicklungsumgebung eingesetzt:

Texteditor	Geany (www.geany.org)
Assembler	Netwide Assembler NASM (www.nasm.us)
SBC86	Die Hardware steht als Emulator zur Verfügung (i8086emu)

Für das Schreiben der Assemblerprogramme kann prinzipiell jeder Texteditor verwendet werden. Zu empfehlen sind Editoren, die verschiedene Sprachelemente farblich hervorheben (Syntax-Highlighting). Geany steht sowohl unter Linux als auch für Windows zur Verfügung.

Entsprechend der Geschichte und aktuellen Bedeutung der Intel-Prozessoren gibt es eine große Anzahl entsprechender Assembler. Bei der Auswahl ist zu beachten, das sich die Syntax der Assemblersprache im Detail unterscheiden kann. Bei der Übernahme von Beispielen aus anderen Quellen ist dies zu berücksichtigen.

Der Emulator i8086emu basiert auf dem SBC86 (SingleBoard-Computer, siehe Seite 1). Es wird sowohl die Hardware mit allen Ein- und Ausgabemöglichkeiten als auch das (orginale) Systemprogramm bereit gestellt. Der SBC 86 ist im weiteren detailliert beschrieben.

Editieren

Bevor sie beginnen: erstellen sie unbedingt ein seperates Verzeichnis für ihre Assemblerprojekte:

Linux:	~/mpt86
Windows:	…\mpt86

Starten sie den Editor

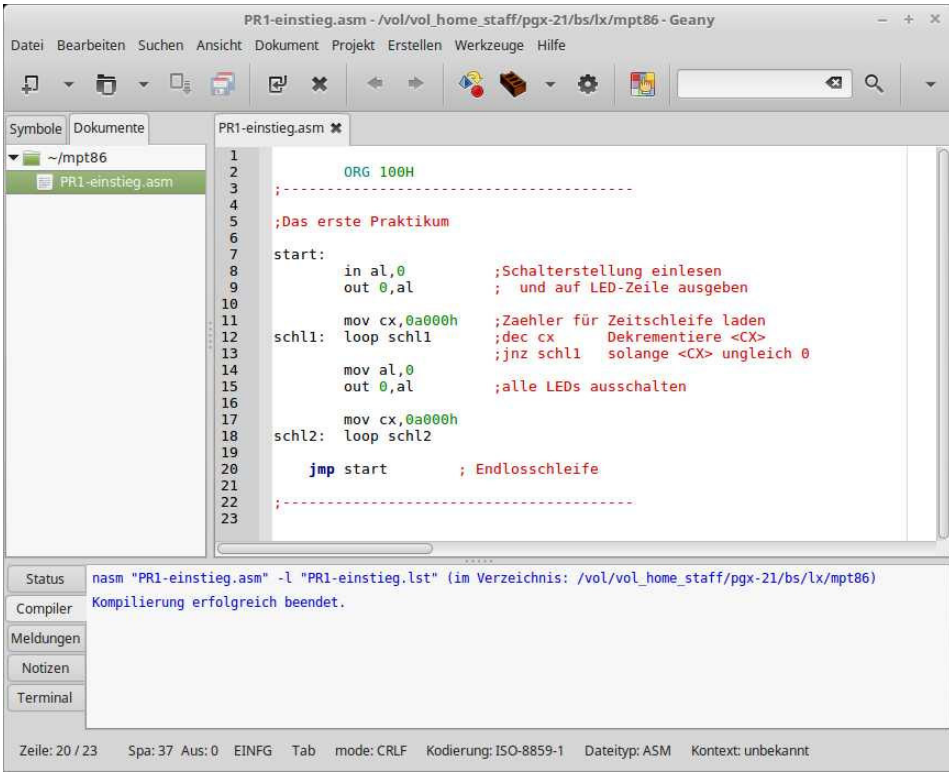
Linux:	Menü ⇨ Entwicklung ⇨ Geany
Windows:	Start ⇨ Geany ⇨ Geany

Speichern sie neue Dateien immer und sofort als Assemblerdatei <name.asm> ab. Nur dann funktionieren Syntax-Highlighting und die korrekte Übersetzung.

Übersetzen

Der Aufruf des Assemblers „nasm“ ist für Dateien <name.asm> voreingestellt und erfolgt über

Geany Menü ⇨ Erstellen ⇨ Kompilieren oder direkt: F8 (Funktionstaste)



Meldungen des Assemblers werden im Fenster „Compiler“ angezeigt.

Durch folgende Konfiguration können sie zusätzlich ein Assembler-Listing erzeugen:

Geany Menü ⇨ Erstellen ⇨ Kommandos zum Erstellen konfigurieren

Kommando unter Linux:

```
nasm "%f" -l "%e.lst"
```

Kommando unter Windows:

```
...\nasm\nasm "%f" -o "%e.com" -l "%e.lst"
```

(-l: erzeuge Listfile)

Nach erfolgreicher Übersetzung werden dann Maschinencode und Assemblerlisting im gleichen Verzeichnis abgelegt:

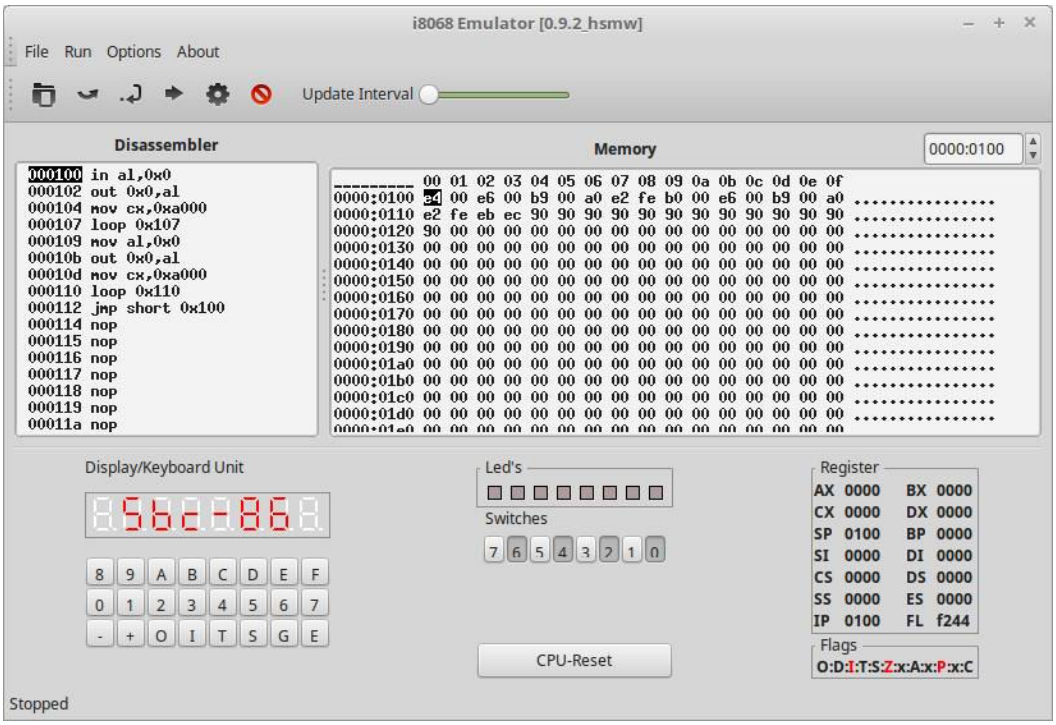
<name.asm>	Assembler-Datei (Quelltext)
<name.lst>	Assembler-Listing (Quelltext und erzeugter Opcode)
<name>	unter Linux (!) …
<name.com>	unter Windows (!) das fertige Programm (Maschinenprogramm, Opcode)

Ausführen und Testen

Starten sie den Emulator i8086emu und laden Sie den Opcode <name> bzw. <name.com>

File ⇨ Open oder Strg+O oder durch anklicken des Symbols „Open File“

Nach Quelltextänderungen und Neuübersetzung reicht dann ein schnelles „Reload“.

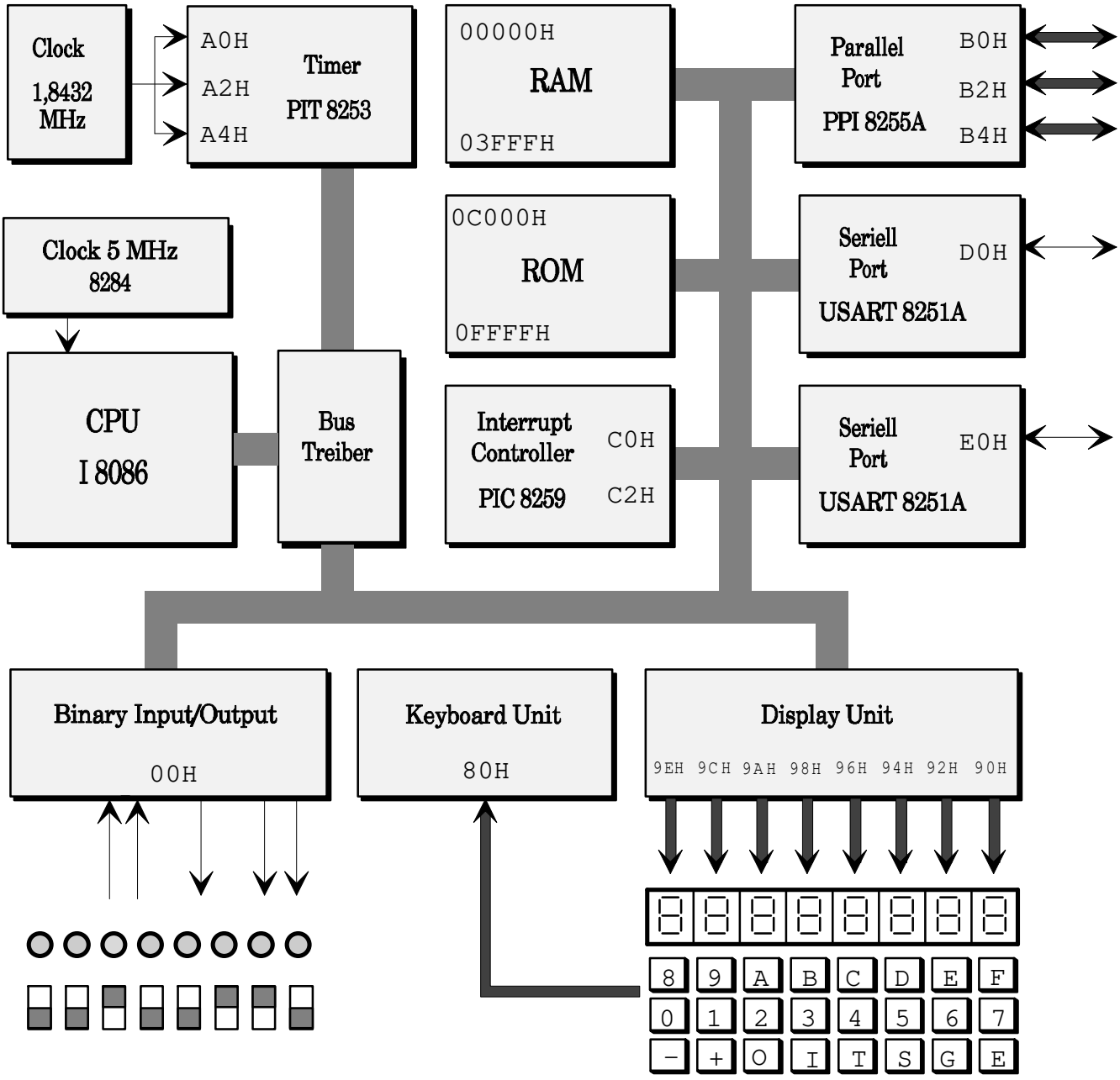


Folgende Test-Funktionen stehen zur Verfügung:

F7	Trace Into	Einzelschritt, Ausführung eines Befehls
F8	Step Over	im Unterschied zu Trace werden hier Unterprogramme und LOOP-Schleifen insgesamt abgearbeitet
F9	Run	das Programm wird an der aktuellen Position gestartet
ESC	Stop	das Programm wird unterbrochen.
Ctrl-R		Change Register, gezieltes Ändern von Registerinhalten
Ctrl-M		Change Memory, gezieltes Ändern von Speicherinhalten
Ctrl-B		Set Breakpoint, Setzen und Rücksetzen von Unterbrechungen

4. Der Singleboard-Computer SBC 86

4.1. Blockschaltbild



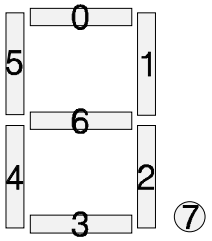
4.2. Speicheraufteilung

0000H - 003FH	Interrupt Vektortabelle
0040H - 004BH	System Zellen
004CH - 00FFH	Systemstack
0100H - 2FFFH	Anwenderprogramme
C000H - CFFFH	Systemprogramm

4.3. Interruptvektortabelle, Hardware-Interrupts

0000H	INT 0		Divisionsüberlauf	nicht genutzt
0004H	INT 1		Single Step Interrupt	Systemprogramm
0008H	INT 2		NMI	nicht genutzt
000CH	INT 3		Breakpoint Interrupt	Systemprogramm
0010H	INT 4		INTO	nicht genutzt
0014H	INT 5		Tastaturzugriff	Systemprogramm
0018H	INT 6		Displayzugriff	Systemprogramm
001CH	INT 7			frei
0020H	INT 8	IRQ0	PIT Kanal 1	nicht genutzt
0024H	INT 9	IRQ1	PIT Kanal 2	nicht genutzt
0028H	INT 10	IRQ2	USART TxRdy (D0)	Systemprogramm
002CH	INT 11	IRQ3	USART RxRdy (D0)	Systemprogramm
0030H	INT 12	IRQ6	PPI Port A (PC3)	nicht genutzt
0034H	INT 13	IRQ7	PPI Port B (PC0)	nicht genutzt
0038H	INT 14			frei
003CH	INT 15			frei
---	---	IRQ4	USART TxRdy (E0)	nicht genutzt
---	---	IRQ5	USART RxRdy (E0)	nicht genutzt

5. Direkter Display- und Tastaturzugriff



Auf die acht 7-Segment-Anzeigen kann über die Adressen 90h-9eh (siehe 4.1) zugegriffen werden. Dabei ist jedes Segment mit einer der acht Ausgangsleitungen des entsprechenden Ports verbunden. Die Ausgabe einer '1' läßt das zugehörige Segment aufleuchten. Durch interne Latches bleibt die Anzeige bis zum nächsten Zugriff erhalten. Nebenstehendes Bild zeigt die Zuordnung der Ausgangsleitungen zu den einzelnen Segmenten.

Die Tastaturmatrix kann über das Port 80h abgefragt werden. Bleibt während mehrerer Zugriffe der eingelesene Wert gleich, liegt eine Tastenbetätigung vor. Die betätigte Taste ist wie folgt kodiert:

0	0	S	S	S	Z	Z	Z	
					1	1	0	Zeile 0 (obere Zeile)
					1	0	1	Zeile 1 (mittlere Zeile)
					0	1	1	Zeile 2 (untere Zeile)
		0	0	0				Spalte 0 (linke Spalte)
		0	0	1				Spalte 1
		0	1	0				Spalte 2
			bis					bis
		1	1	1				Spalte 7 (rechte Spalte)

6. Display- und Tastaturzugriff über Systemaufrufe

Vom Systemprogramm des SBC werden über die Software-Interrupts 5 und 6 Routinen bereitgestellt, die die Arbeit mit Tastatur und Display vereinfachen.

INT 5 - Tastaturstatus ermitteln

Parameter:	Rückgabe:
AH = 0	AL = 00 keine Taste betätigt sonst Taste betätigt

INT 5 - Warten auf Tastenbetätigung

Parameter:	Rückgabe:
AH = 1	AL = Tastencode 0..F 00..FF ENTER 10 GO 11 STEP 12 usw.

INT 5 - Eingabe einer 16-Bit-Hexadezimalzahl

Parameter:	Rückgabe:
AH = 2 BX = Vorgabewert DL = Displaystelle	AX = eingegebene Hexadezimalzahl DL = Tastaturcode der Quittung (10h, 16h oder 17h)

Der Vorgabewert wird ab der gewünschten Stelle nach rechts ins Display geschrieben und kann dann über die Tastatur geändert werden.

INT 6 - Display löschen

Parameter:
AH = 0

Das gesamte Display wird gelöscht.

INT 6 - Ausgabe eines ASCII-Zeichens

Parameter:
AH = 1 AL = Zeichen DL = Displaystelle

Das ASCII-Zeichen wird in die gewünschte Displaystelle geschrieben.
Die Displaystellen werden von rechts mit 0 beginnend numeriert.

INT 6 - Ausgabe eines ASCII-Zeichenkette

Parameter:
AH = 2 BX = Adresse des 1.Zeichens DL = Displaystelle

Die Zeichenkette wird ab der gewünschten Stelle nach rechts ins Display geschrieben. Als Textendekennung wird 00 erwartet.

INT 6 - Ausgabe einer 16-Bit-Hexadezimalzahl

Parameter:
AH = 3 BX = Hexadezimalzahl DL = Displaystelle

Die angegebene Zahl (4 Hexzeichen) wird ab der gewünschten Stelle nach rechts ins Display geschrieben.

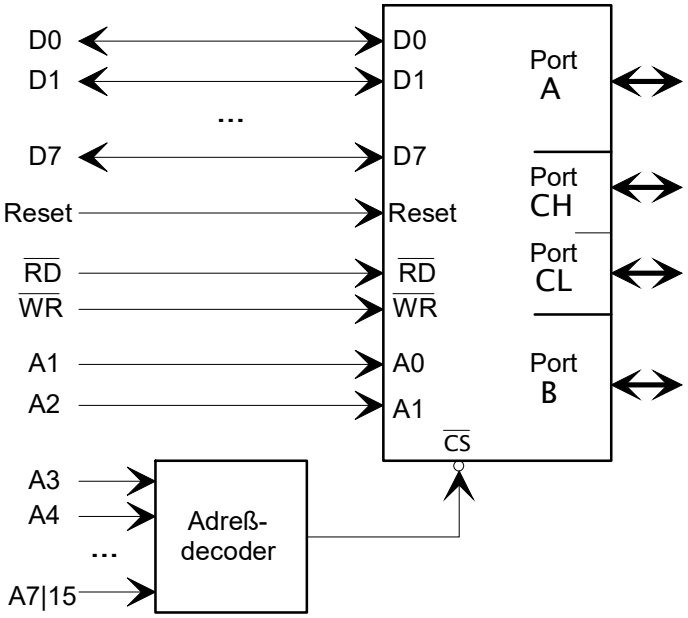
INT 6 - Ausgabe einer 8-Bit-Hexadezimalzahl

Parameter:
AH = 4 BL = Hexadezimalzahl DL = Displaystelle

Die angegebene Zahl (2 Hexzeichen) wird ab der gewünschten Stelle nach rechts ins Display geschrieben.

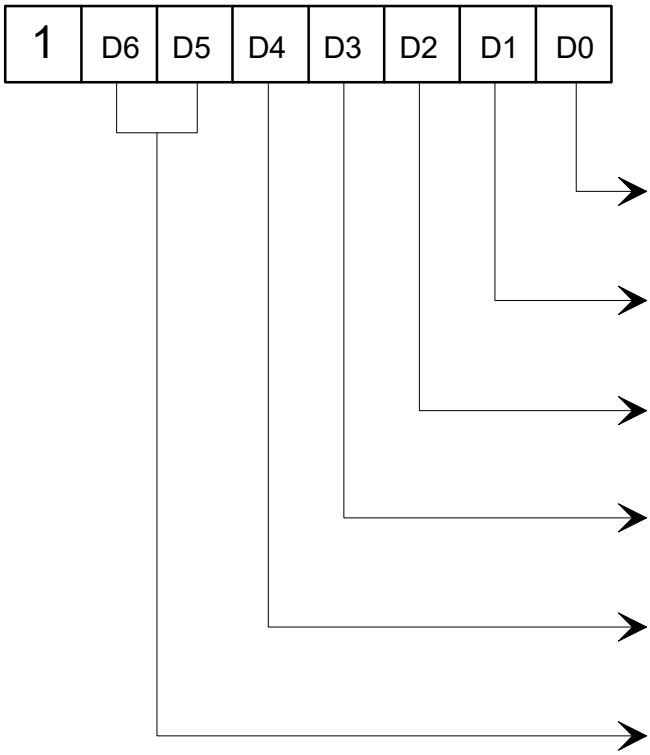
7. Programmable Peripheral Interface PPI 8255

7.1. Adressierung



A7 | 15 ... A3	A2	A1	
Adreß-	0	0	DATA Port A
auswahl	0	1	DATA Port B
über	1	0	DATA Port C
Decoder	1	1	Control

7.2. Modesteuerwort

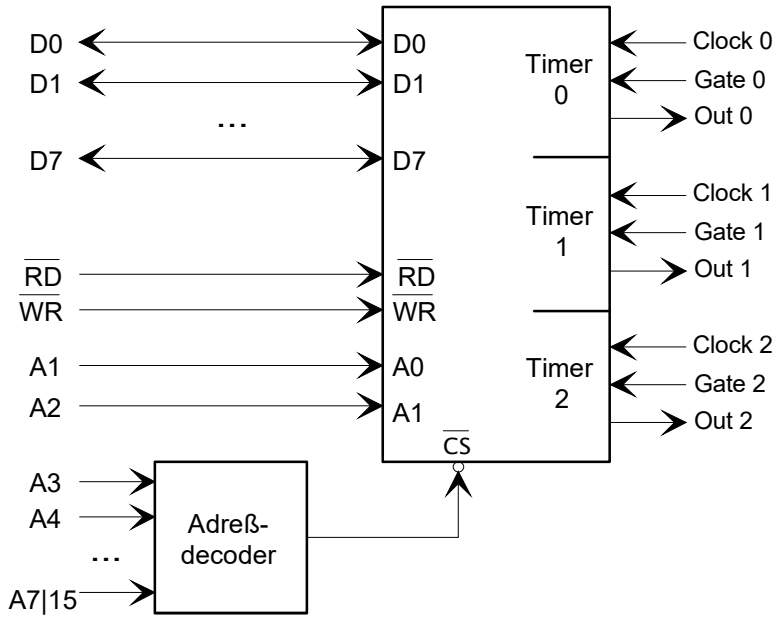


Port C Low	0	=	Output
	1	=	Input
Port B	0	=	Output
	1	=	Input
Port B	0	=	Mode 0
	1	=	Mode 1
Port C Hi	0	=	Output
	1	=	Input
Port A	0	=	Output
	1	=	Input
Port A	00	=	Mode 0
	01	=	Mode 1
	1x	=	Mode 2

Mode 0: Basic Input/Output (Port A, B, C)
Mode 1: Strobed Input/Output (Port A, B)
Mode 2: Bi-Directional Input/Output (Port A)

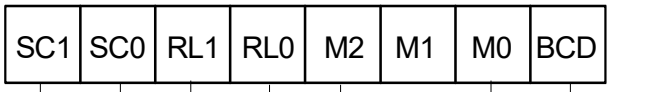
8. Programmable Interval Timer PIT 8253 / 8254

8.1. Adressierung



A7|15...A3	A2	A1	
Adress-	0	0	Timer 0
auswahl	0	1	Timer 1
über	1	0	Timer 2
Decoder	1	1	Steuerwort

8.2. Steuerwort



BCD 0 = Binärzähler, 16 Bit
1 = Dezimalzähler, 4 Dekaden

Betriebsart (siehe 8.3)
000 = Mode 0
001 = Mode 1
x10 = Mode 2
x11 = Mode 3
100 = Mode 4
101 = Mode 5

Lesen / Schreiben
00 = akt. Zählerstand zwischenspeichern
01 = nur das Low-Byte
10 = nur das Hi-Byte
11 = erst Low-Byte, dann Hi-Byte

Das Steuerwort bezieht sich auf
00 = Timer 0
01 = Timer 1
10 = Timer 2
11 = Read Back (nur 8254)

Nach Laden des Steuerwortes wird der Anfangswert des Zählers erwartet. Ein 16-Bit-Wert wird dann in der Reihenfolge Low-Byte vor Hi-Byte in das Zählregister geladen.

8.3. Betriebsarten

Mode 0: Interrupt bei Nulldurchgang

Nach Laden des Steuerwortes wird OUT auf „0“ gesetzt, nach Laden des Anfangswertes beginnt der Zählvorgang. Bei Nulldurchgang wird dann eine „1“ ausgegeben. Eine „0“ an GATE unterbricht den Zählvorgang. Mode 0 ist nicht repetierend.

Mode 1: Monoflop

Eine steigende Flanke von GATE startet den Zähler und setzt OUT auf „0“. Beim Zählerwert 0 wird eine „1“ ausgegeben und der Zähler stoppt. Eine steigende Flanke von GATE vor Ablauf des Zählers setzt das Zählregister neu, der Ausgang bleibt auf „0“. Mode 1 ist nicht repetierend.

Mode 2: Frequenzgenerator (Teiler durch N)

Die an CLK anliegende Frequenz wird durch den Anfangswert des Zählers geteilt. Bei Nulldurchgang geht OUT für eine Taktperiode von CLK auf „0“. Eine fallende Flanke an GATE bricht den Zählvorgang ab und setzt OUT auf „1“, die steigende Flanke startet den Zählvorgang neu. Mode 2 ist repetierend.

Mode 3: Generator mit symmetrischem Ausgangssignal

Ähnlich Mode 2. Der Zähler dekrementiert immer um zwei, zu einem vollständigen Zählvorgang gehören zwei Nulldurchgänge. Nach dem ersten wird eine „0“ ausgegeben und nach dem zweiten eine „1“. Bei ungeraden Anfangswerten liegt OUT eine Taktperiode länger auf „1“. Mode 3 ist repetierend.

Mode 4: Softwaregesteuerter Impuls (strobe)

Der Zähler startet, sobald der Anfangswert geladen ist. Bei Nulldurchgang wird „0“ für die Dauer einer Taktperiode ausgegeben. Eine „0“ an GATE unterbricht den Zählvorgang. Mode 4 ist nicht repetierend.

Mode 5: Hardwaregesteuerter Impuls (strobe)

Der Zähler startet mit einer steigenden Flanke von GATE, sonst wie Mode 5.