Handbuch

MCS51 Experimentierplatine

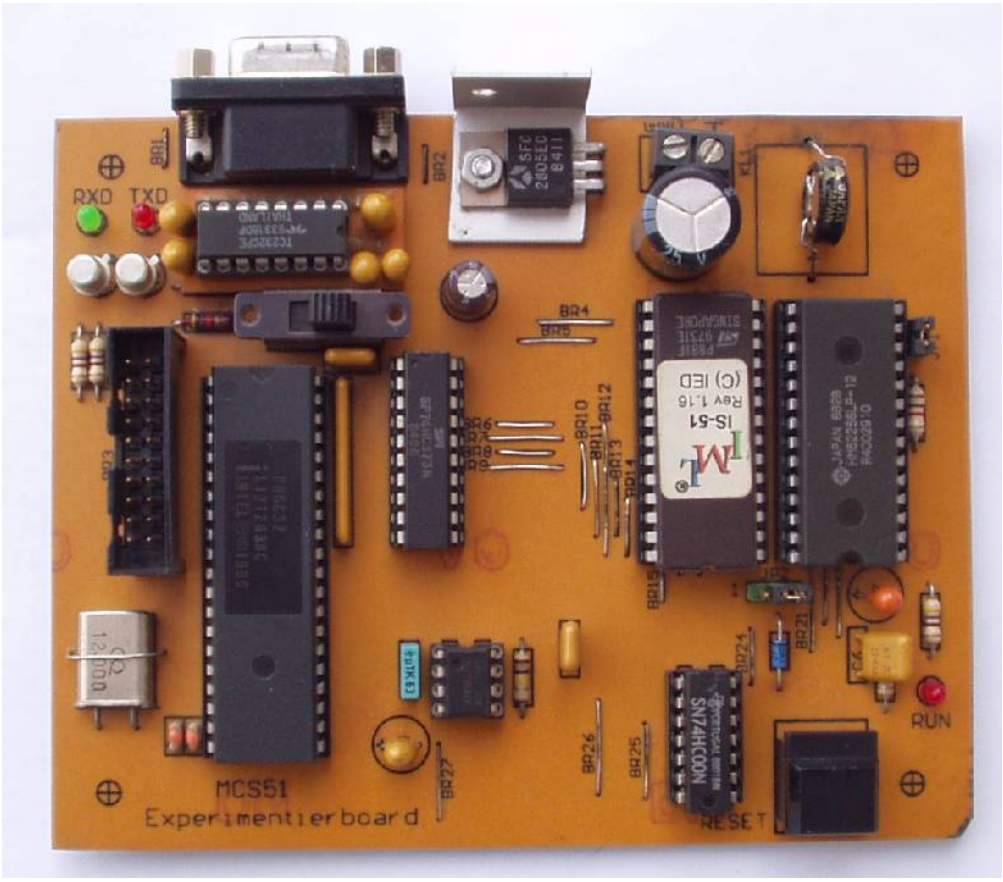
IS51

mit Monitorprogramm

Version 1.20

und Erweiterungsplatine

IS51-E/A1



Stand: 12. März 2009

Inhaltsverzeichnis

1 Allgemeine Beschreibung

Die IS51 ist geeignet, die Hard- und Software der MCS51 Mikrokontroller durch Versuche kennenzulernen. Es kann mit einem geeigneten Programm aber auch als Steuergerät eingesetzt werden. Es ist auch möglich, die IS51 als Erweiterung der Fähigkeiten des PC über eine serielle Schnittstelle zu nutzen. Dadurch wird es dem PC auf einfache Weise möglich, Messwerte zu erfassen oder Relais zu schalten. Das Board verfügt über insgesamt 64KB Speicher. Dieser ist aufgeteilt in 32KB EPROM für das Monitorprogramm und die Unterprogramme, und 32KB RAM für die Anwenderprogramme und als Datenspeicher. Die Arbeitsprogramme werden auf einem Leitrechner erstellt und über die serielle Schnittstelle direkt in den Speicher des Experimentierboards geladen. Befindet sich ein Programm im Speicher, kann es durch den Mikrokontroller ausgeführt werden, ohne daß der Leitrechner weiter gebraucht wird. Die Anwenderprogramme sind im RAM gespeichert. Dieses ist mit einem NiCd-Akku verbunden, so daß sein Inhalt beim Abschalten der Versorgungsspannung nicht verloren geht. Zum Anschluß der Versuchsschaltungen ist ein Erweiterungsstecker vorhanden. Auf diesen sind die digitalen Ein/Ausgänge und die Versorgungsspannung geführt. Der Spannungsregler für die Versorgungsspannung befindet sich mit auf der Platine, so daß von außen eine ungeregelte Spannung von 8V bis 12V (max.15V) angelegt werden kann.

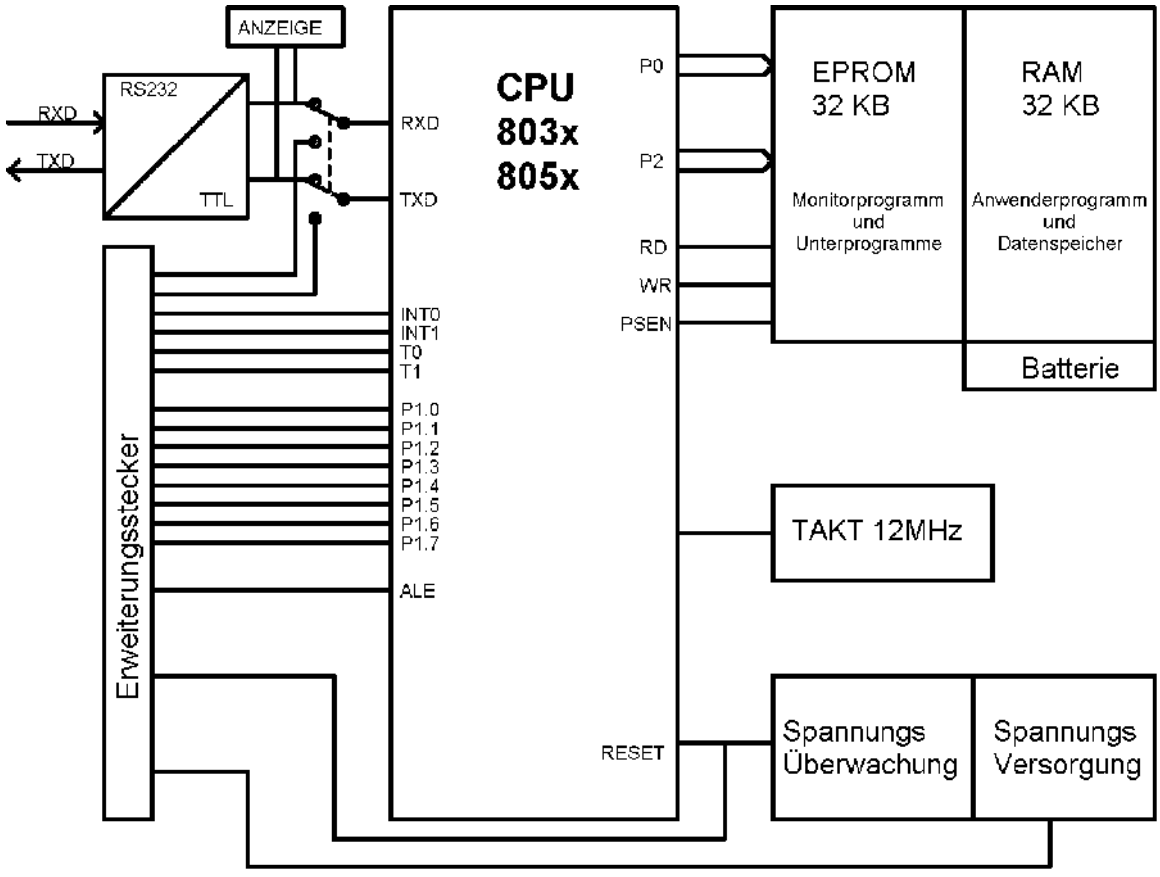


Bild 1 Blockschaltbild Experimentierboard IS51

2 Hardware

2.1 Stromversorgung

Das Experimentierboard arbeitet mit einer einfachen Betriebsspannung von 5 Volt. Zur Stabilisierung dieser Spannung befindet sich auf dem Board ein integrierter Spannungsregler. Die Eingangsspannung wird über Klemme KL1 zugeführt. Sie darf zwischen 8 Volt und 12 Volt liegen. Die Stromaufnahme des Boards ist abhängig vom eingesetzten Kontroller-Baustein und den angeschlossenen Zusatzplatinen. Durch den Spannungregler darf ein maximaler Strom von 0,25A fließen. Wird die Strombelastung durch eine Zusatzschaltung höher, ist ein zusätzlicher Kühlkörper zu montieren. Die Höhe der Betriebsspannung wird durch IC5 überwacht. Fällt sie unter ca. 4,5 Volt, so wird der Zugriff auf das RAM gesperrt und ein RESET des Prozessors ausgelöst. Der Akku zum Puffern der RAM-Spannung wird im eingeschalteten Zustand des Boards mit einem Strom von ca. 1mA geladen. Wird das Board länger als ein halbes Jahr nicht benutzt, ist durch Ziehen des Jumpers JP1 der Pufferakku von der Schaltung zu trennen. Zur Anzeige der anliegenden Betriebsspannung befindet sich auf dem Board die RUN-LED D1.

Stromaufnahme:	NMOS-Prozessor:	140mA / 8031 160mA / 8032
	CMOS-Prozessor:	60mA / 80C31-32

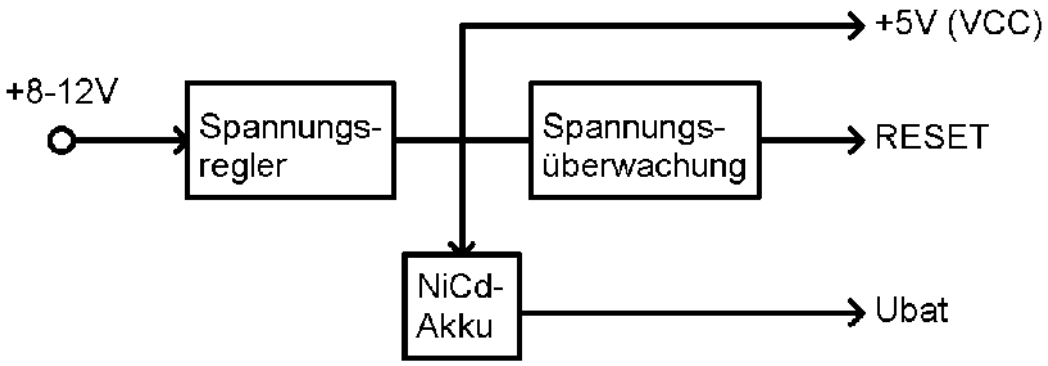


Bild 2.1 Spannungsversorgung

2.2 CPU

Als Mikrokontroller-Baustein können verschiedene Typen der MCS51 Familie eingesetzt werden. Durch den aktivierten Bausteinanschluß EA ist der interne Programmspeicher abgeschaltet. Dadurch ist es möglich auch die Bausteine mit internen Programmspeicher einzusetzen. Die Taktfrequenz der CPU beträgt 12MHz. In Anwenderprogrammen kann auf alle Bausteinfunktionen, die nicht zur Speicherbedienung benötigt werden, uneingeschränkt zugegriffen werden. Zur Speicherbedienung werden P0, P2, RD (P3.7) und WR (P3.6) benötigt.

Einsetzbare Mikrokontroller:

8031, 8032, 8051, 8052, 8751, 8752, 80C31, 80C32, 80C51, 80C52, 87C52, 89S52, 89S8252, 89S8253 und viele andere Bausteine im 40 poligen DIL Gehäuse.

2.3 Speicher

Das Experimentierboard verfügt über 64KB externen Speicher. Dieser ist aufgeteilt in 32KB EPROM und 32KB RAM. Das EPROM belegt den Adressbereich von 0000h bis 7FFFh. Es beinhaltet das Monitorprogramm und die Unterprogramme. Das RAM belegt den Adressbereich 8000h bis FFFFH. Es kann die Anwenderprogramme aufnehmen und als allgemeiner Datenspeicher verwendet werden. Das RAM kann wahlweise mit +5V (VCC) oder Ubat (Batteriespannung) verbunden sein. Mit Jumper JP2 in Position 1-2 ist das RAM mit VCC verbunden, in Position 2-3 wird das RAM mit der Batteriespannung versorgt. Wird das RAM mit VCC versorgt, sind die Inhalte aller Speicherzellen nach dem Abschalten der Versorgungsspannung undefiniert. Ist das RAM mit der Batteriespannung verbunden, bleiben die Inhalte der Speicherzellen über das Abschalten der Versorgungsspannung erhalten. Um ein Programm aus dem Speicher zu entfernen, ist Jumper JP2 in Position 1-2 zu stecken und die Versorgungsspannung abzuschalten. Befindet sich Jumper JP2 ständig in Position 1-2, ist durch ziehen von Jumper JP1 der Akku von der Schaltung zu trennen. Ohne diese Maßnahme ist, abhängig vom eingesetzten RAM, unter Umständen der Ruhestrom aus dem Pufferakku zu hoch. Die Prozessorsignale PSEN und RD sind beim Experimentierboard zu einem RAM Freigabesignal zusammengefasst. Dadurch ist ein gemischter Programm- Datenspeicher Betrieb im RAM möglich. Nur durch diese Schaltungsart ist es mit MCS51 Bausteinen möglich Programme im Datenspeicher auszuführen, wodurch das Download von Programmen durchgeführt werden kann. Beim unterschreiten von ca. 4,5V werden durch die Spannungsüberwachung Zugriffe zum RAM gesperrt. Dadurch wird ein unbeabsichtigtes Überschreiben der Speicherzellen beim Ein- und Ausschalten der Versorgungsspannung verhindert.

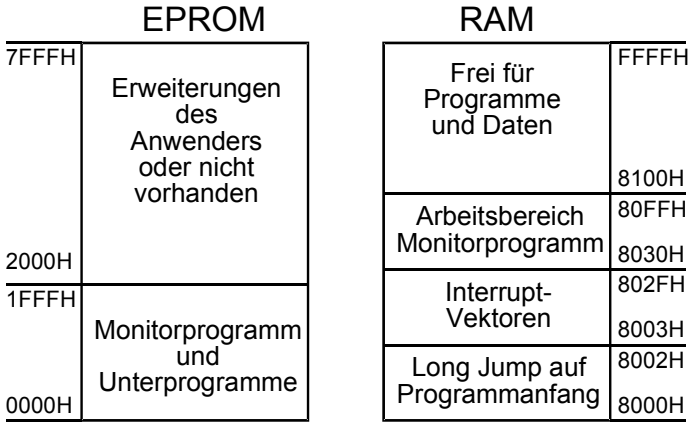


Bild 2.2 Speicher des Experimentierboard

JP1	JP2	Funktion
auf	1-2	Kein Datenerhalt im RAM
auf	2-3	Kein Datenerhalt im RAM
zu	1-2	Nur Kurzzeitig zum Löschen des RAM
zu	2-3	Datenerhalt im RAM

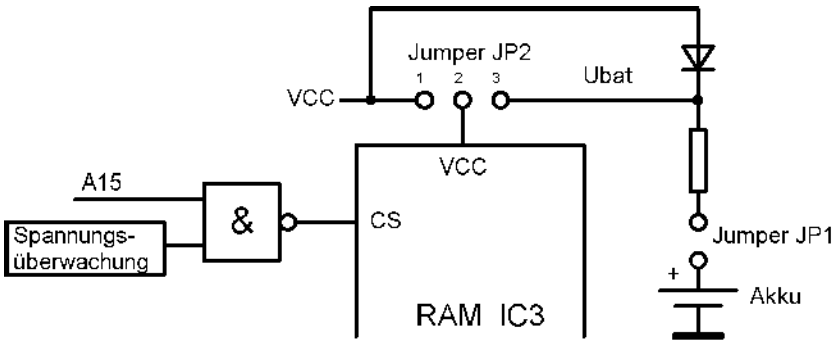


Bild 2.3 Anschaltung RAM

2.4 Ein/Ausgabe

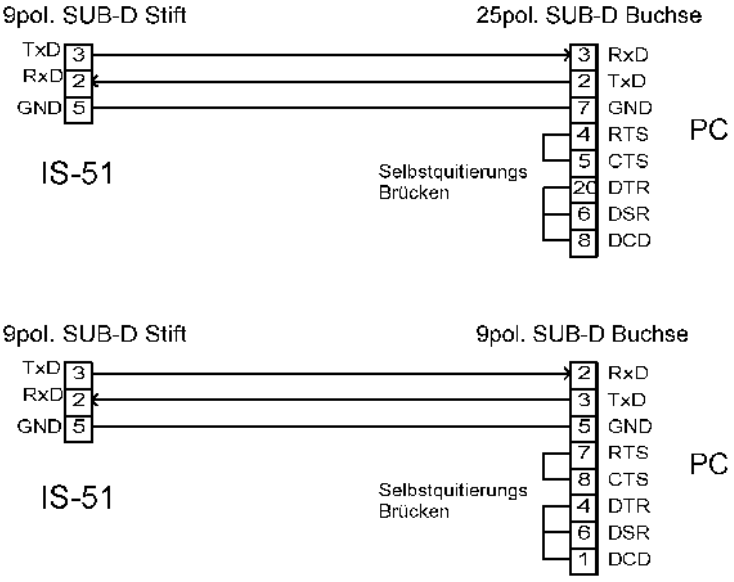
Die digitalen Ein/Ausgänge des Experimentierboards sind Portbit des Mikrokontrollers. Sie können als digitale Eingänge oder als digitale Ausgänge verwendet werden. Alle verfügbaren Portbit sind auf den Erweiterungsstecker ST1 geführt. Sie können zur Steuerung der Versuchsschaltungen benutzt werden. Zusätzlich stehen an ST1 die Signale ALE, RESET und die Versorgungsspannung für externe Schaltungen zur Verfügung. Wird die serielle Schnittstelle (SIO) nicht gebraucht, so können durch umschalten von SW1 in Position Port, die Signale RXD (P3.0) und TXD (P3.1) auf den Erweiterungsstecker geschaltet werden. Das Signal ALE kann als Taktsignal für Versuchsschaltungen herangezogen werden. Es hat eine Taktfrequenz von Quarz/6, beim Experimentierboard also 12MHz/6 ist 2MHz. Das Signal RESET dient als High aktives Rücksetzsignal für Versuchsschaltungen. Die Pegel der Signale sind, abhängig vom verwendeten Mikrokontroller, entweder TTL- oder CMOS-Pegel.

Signal	PIN		Signal		Signal	PIN		Signal
+5V (VCC)	1	2	+5V (VCC)		T0 (P3.4)	11	12	P1.4
RXD (P3.0)	3	4	P1.0		T1 (P3.5)	13	14	P1.5
TXD (P3.1)	5	6	P1.1		ALE	15	16	P1.6
INT0 (P3.2)	7	8	P1.2		RESET	17	18	P1.7
INT1 (P3.3)	9	10	P1.3		GND	19	20	GND

Bild 2.4 Belegung Erweiterungsstecker

2.5 Serielle Schnittstelle

Die serielle Schnittstelle des Experimentierboards dient in erster Linie zum Download der Anwenderprogramme. Dazu wird die SIO (Serial Input Output) des Boards mit einer seriellen RS232C (COMx) Schnittstelle des Personal Computers (PC) verbunden. Die SIO wird von dem im Mikrokontroller integrierten UART gesteuert. Der Pegelwandler für RS232 befindet sich mit auf dem Board. Die Schnittstelle arbeitet im „Drei Draht Betrieb“, das bedeutet sie benutzt RXD, TXD und GND. Die Leitungssignale sind auf die 9 polige SUB-D Buchse ST2 geführt. Die Verdrahtung der Leitungsbuchse entspricht einer Daten-End-Einrichtung (DEE). Die Übertragungsparameter sind 4800 Baud, ohne Paritätsprüfung, 8 Datenbit und 1 Stopbit.



PIN	NAME	FUNKTION
2	RXD	Eingang
3	TXD	Ausgang
5	GND	Signalerde

Belegung ST2 DEE 9pol. SUB-D

Bild 2.5 Serielles Verbindungskabel zum PC

Die Signale RXD (P3.0) und TXD (P3.1) des Mikrokontrollers sind über Schalter SW1 geführt. Dadurch können sie wahlweise auf die Übertragungsleitung (Stellung SIO) oder den Erweiterungsstecker ST1 (Stellung Port) geschaltet werden. Eine Datenübertragung über die SIO ist nur in Stellung SIO möglich. Zur Anzeige der Datenübertragung befinden sich für beide Übertragungsrichtungen LED´s auf der Platine. Sie leuchten jeweils beim LOW-Pegel der entsprechenden Leitung auf. In Stellung Port können die Signale von einer Erweiterungsschaltung als Portbit oder Steuerleitungen für eine von RS232 abweichende SIO genutzt werden. Nach dem Download eines Anwenderprogramms kann SW2 in Stellung P gebracht werden. Das Programm ist in den Speicher geladen und bei Batteriepufferung gesichert, wodurch die SIO nun nicht mehr gebraucht wird.

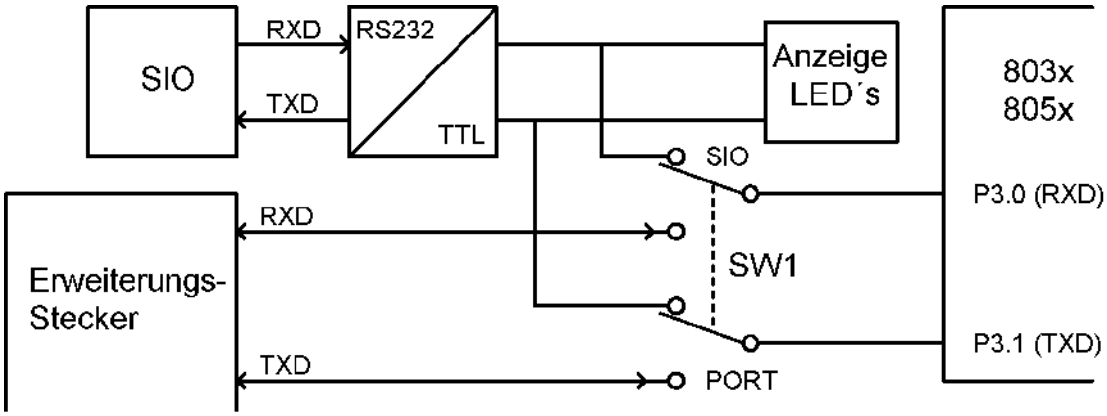


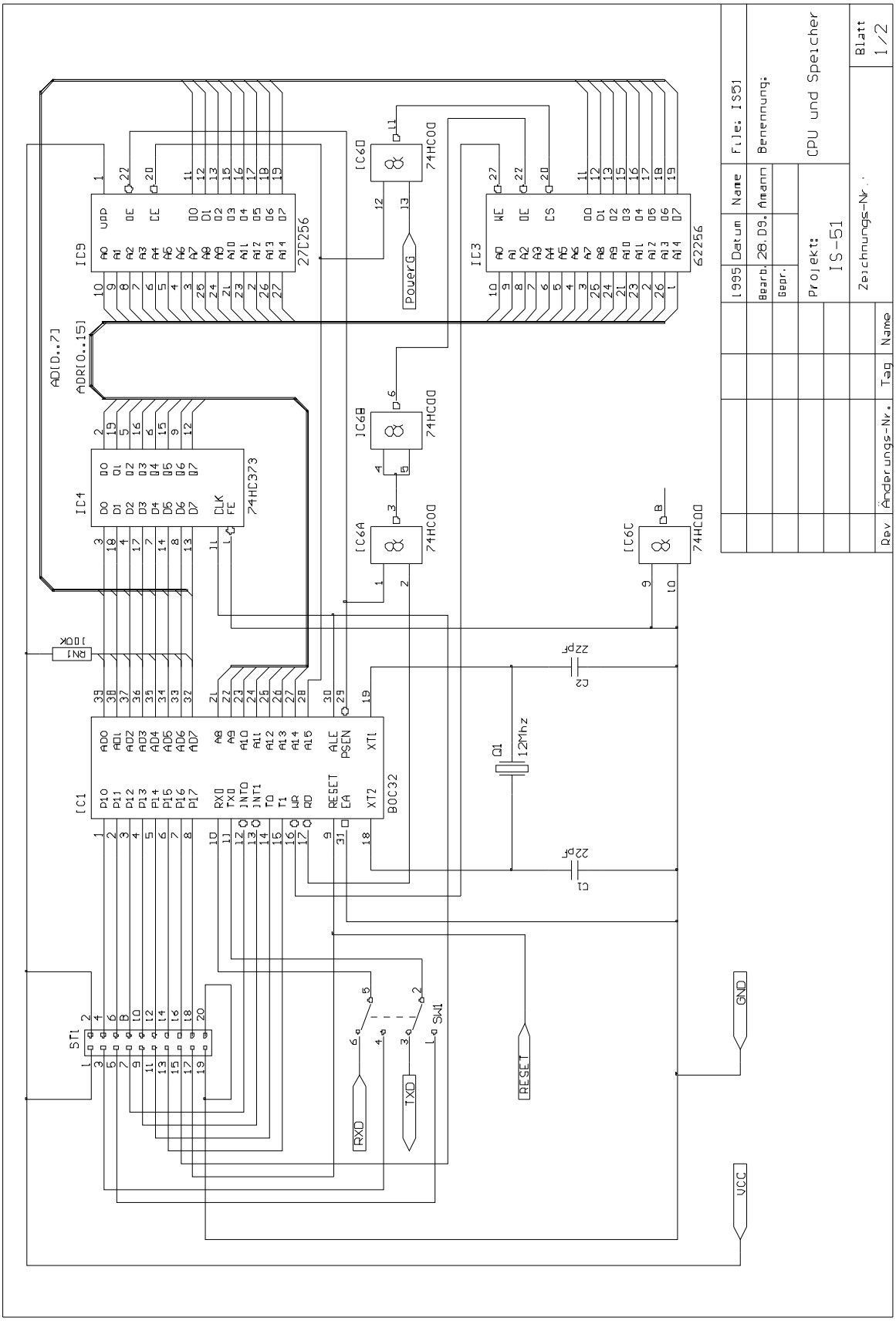
Bild 2.6 Funktion Schalter SW1

2.6 Technische Daten IS51

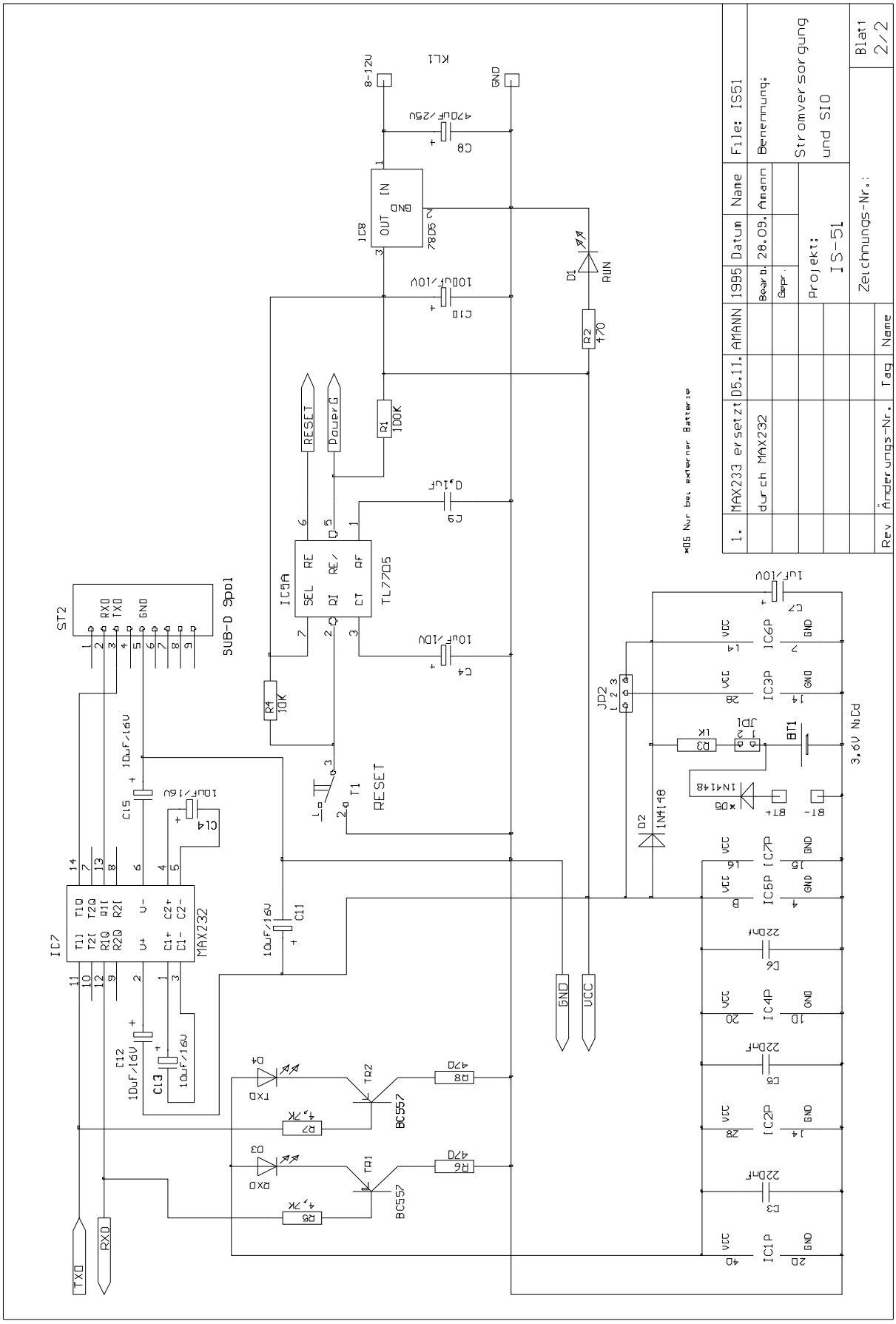
Takt	12MHz			
Versorgungsspannung	+8V bis +12V (maximal 15V)			
Stromaufnahme	ca. 180mA NMOS		ca. 60mA CMOS	
Strom im Spannungsregler	Maximal 0,25A ohne Zusatzkühlkörper			
Pufferakku	3,6V NiCd		Ladestrom ca. 1mA	
Ein/Ausgabe	12 (14) Digitale Ein/Ausgabebit			
Pegel der Ein/Ausgabe	80xx		80Cxx	
	U_{QL} = 0,45V	U_{QH} = 2,4V	U_{QL} = 0,45V	U_{QH} = 2,4V
	U_{IL} = 0,8V	U_{IH} = 2,0V	U_{IL} = 0,7V	U_{IH} = 1,9V
	I_{QL} = 2,4mA	I_{QH} = -80uA	I_{QL} = 1,6mA	I_{QH} = -60uA
	I_{IL} = -800uA	I_{IH} = 500uA	I_{IL} = -50uA	I_{IH} = 50uA
Serielle Schnittstelle	RS 232 C	Asynchron	Drei Draht Betrieb	
	4800 Baud, 8 Datenbit, 1Sopbit, ohne Paritätsprüfung			
	Verdrahtet nach DEE			
Sonstiges	Resettaster, Umschalter für Signale RXD und TXD, Anzeige LED´s für Datenübertragung, Spannungsregler für +5V auf Platine.			

2.7 Schalt- und Bestückungspläne

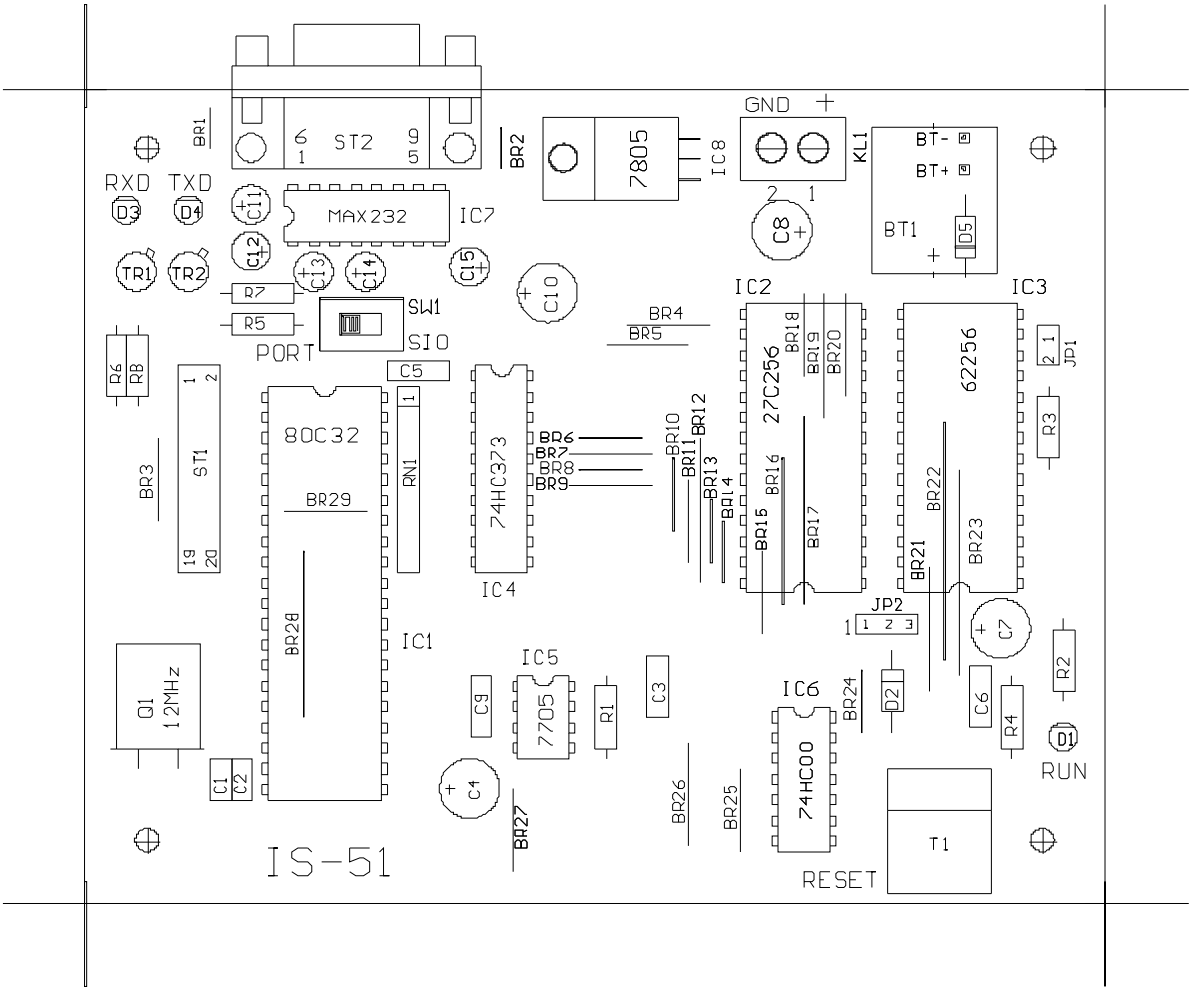
2.7.1 Stromlauf 1



2.7.2 Stromlauf 2



2.7.3 Bestückungsplan



3 Software

3.1 Monitorprogramm

Das Monitorprogramm befindet sich im EPROM der Experimentierplatine. Es hat hauptsächlich die Aufgabe, das Laden von Anwenderprogrammen zu ermöglichen. Das Download von Programmen und Daten geschieht im INTEL-HEX Format. Tritt bei der Übertragung ein Fehler auf, blinkt die rote TXD-LED solange bis ein RESET ausgelöst wird. Der interne Speicher der Mikrokontroller wird vom Monitorprogramm nicht benötigt, so daß er dem Anwender zur Verfügung steht. Der Stack beginnt auf Adresse 30H. Die Interrupt-Vektoren werden durch das Monitorprogramm in das RAM verlegt, wodurch der Anwender die Interrupts der Mikrocontroller nutzen kann. Anwenderprogramme beginnen immer ab RAM-Adresse 8000H mit einem LONG JUMP (OP-Code 02H). Befindet sich ein Anwenderprogramm im RAM, wird es nach RESET automatisch gestartet. Das Monitorprogramm wird unter diesen Umständen nicht weiter beachtet. Nach dem Einschalten der Versorgungsspannung wird ein Schreib-Lesetest des RAM durchgeführt, wird dabei ein Fehler festgestellt blinkt die rote TXD-LED.

3.2 Speicherbelegung

Der Mikrokontroller interne Speicher ist für den Anwender frei Verfügbar. Der Stackpointer wird durch das Monitorprogramm auf 2FH gelegt, wodurch der Stack ab Adresse 30H beginnt. Er kann vom Benutzer, falls nötig, auf eine andere Adresse gestellt werden. Das EPROM belegt den Speicherbereich 0000H bis 7FFFH. Das RAM belegt den Adressbereich 8000H bis FFFFH. Im Eprom ist für Monitorprogramm und Unterprogramme der Adressbereich 0000H bis 1FFFH reserviert. Der Adressbereich 2000H bis 7FFFH ist für Erweiterungen des Anwenders freigehalten. Im RAM sind die Adressen 8000H, 8001H und 8002H für den LONG JUMP auf den Programmstart reserviert. Ab Adresse 8003H bis 802FH befinden sich die Einsprungadressen der Interrupts. Für jeden Interrupt sind 3 Adressen freigehalten. Der Adressbereich 8030H bis 80CFH ist der Arbeitsbereich des Monitorprogramms. Der Adressbereich 8100H bis FFFFH ist frei für Anwenderprogramme und Daten.

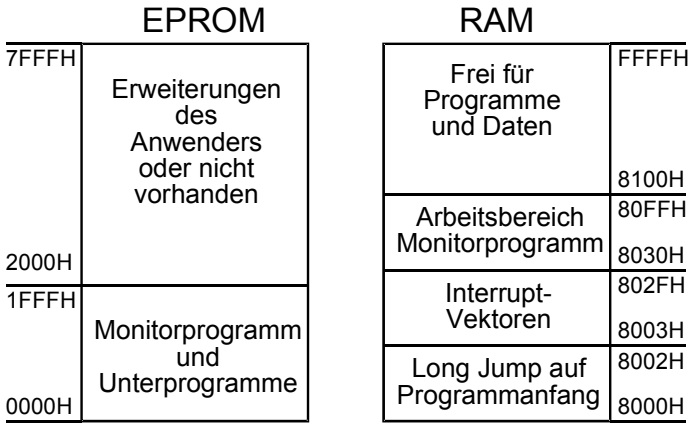


Bild 3.1 Speicherbelegung

3.2.1 Speicherbelegung im Adressbereich 8000H-80FFH

8000H - 8002H Autostartkennung und Sprung auf Programmanfang

8003H - 802FH Interrupt Vektoren

8030H	EOUTPORT1, Rettungszelle für Ausgabeerweiterungsport 1 (z.B: IS51-E/A1)
8031H	EOUTPORT2, Rettungszelle für Ausgabeerweiterungsport 2
8032H	EOUTPORT3, Rettungszelle für Ausgabeerweiterungsport 3
8033H	EOUTPORT4, Rettungszelle für Ausgabeerweiterungsport 4
8034H	EOUTPORT5, Rettungszelle für Ausgabeerweiterungsport 5
8035H	EOUTPORT6, Rettungszelle für Ausgabeerweiterungsport 6
8036H	EOUTPORT7, Rettungszelle für Ausgabeerweiterungsport 7
8037H	EOUTPORT8, Rettungszelle für Ausgabeerweiterungsport 8
8038H	EINPORT1, Rettungszelle für Eingabeerweiterungsport 1
8039H	EINPORT2, Rettungszelle für Eingabeerweiterungsport 2
803AH	EINPORT3, Rettungszelle für Eingabeerweiterungsport 3
803BH	EINPORT4, Rettungszelle für Eingabeerweiterungsport 4
803CH	EINPORT5, Rettungszelle für Eingabeerweiterungsport 5
803DH	EINPORT6, Rettungszelle für Eingabeerweiterungsport 6
803EH	EINPORT7, Rettungszelle für Eingabeerweiterungsport 7
803FH	EINPORT8, Rettungszelle für Eingabeerweiterungsport 8
8040H	TELEANZ, Anzahl der Byte im Telegrammpuffer
8041H- 806FH	TELEPUF, Telegrammpuffer für Datenaustausch mit Treiber
8070H	RETR0, Rettungszelle für Register 0
8071H	RETR1, Rettungszelle für Register 1
8072H	RETR2, Rettungszelle für Register 2
8073H	RETR3, Rettungszelle für Register 3
8074H	RETR4, Rettungszelle für Register 4
8075H	RETR5, Rettungszelle für Register 5
8076H	RETR6, Rettungszelle für Register 6
8077H	RETR7, Rettungszelle für Register 7
8078H	RETDPL, Rettungszelle für Datenpointer Low
8079H	RETDPH, Rettungszelle für Datenpointer High
807AH	RETPSW, Rettungszelle für Programm Status Wort
807BH	RETACC, Rettungszelle für Akku
807CH	RETB, Rettungszelle für Register B
807DH	CPUID, Typ des eingesetzten Mikrokontrollers 00H = 80x1, 01H = 80x2, 10H = 80Cx1, 11H = 80Cx2
807EH - 807FH	ADAW16, Arbeitszellen für Werte der externen A/D-Wandler
8080H - 808CH	BEFPUF, Befehlspuffer
808DH	ADOK, A/D-Auflösung: 00 = kein Meßwert, 08 = 8Bit, 0A = 10Bit, ... usw.
808EH - 808FH	ADWERT, A/D-Wandler Meßwert: 808EH = L, 808FH = H
8090H – 80AFH	Adressen der 16 Anwenderroutinen
80B0 - 80FFH	Reserviert

Die reservierten Speicherzellen sind für Erweiterungen des Monitorprogramms vorgesehen, sie sollten nicht für andere Zwecke benutzt werden.

3.3 Interrupt

Die Einsprungadressen der Mikrocontroller Interrupts werden durch das Monitorprogramm in den RAM-Bereich verlegt. Dadurch können die Interrupts in Anwenderprogrammen genutzt werden. Für jeden Interrupt sind 3 Adressen reserviert. Dies reicht aus, einen LJMP-Befehl auf jede beliebige Adresse aufzunehmen.

Adressbereich	Interrupt	Originalvektor
8003H - 8005H	INT0-Interrupt (INT0)	03H
8006H - 8008H	Timer 0 Überlauf (TF0)	0BH
8009H - 800BH	INT1-Interrupt (INT1)	13H
800CH - 800EH	Timer 1 Überlauf (TF1)	1BH
800FH - 8011H	UART (RI/TI)	23H
8012H - 8014H	Timer 2 Überlauf (TF2/EXF2 - NUR 80x2)	2BH
8015H – 8017H	Interrupt 6	33H
8018H – 801AH	Interrupt 7	3BH
801BH – 801DH	Interrupt 8	43H
801EH – 8020H	Interrupt 9	4BH
8021H – 8023H	Interrupt 10	53H
8024H – 8026H	Interrupt 11	5BH
8027H – 8029H	Interrupt 12	63H
802AH – 802CH	Interrupt 13	6BH
802DH – 802FH	Interrupt 14	73H

Tabelle 3.1 Einsprungadressen der Interrupts

3.4 Anwenderprogramme

Anwenderprogramme werden auf dem Leitrechner erstellt. Sie müssen im MCS51-Maschinencode als INTEL-HEX Datei vorliegen. Diese Datei wird über die serielle Schnittstelle in den RAM-Speicher des Experimentierboards geladen (Download). Anwenderprogramme müssen immer ins RAM ab Adresse 8000H geladen werden. Sofern sie mit einem LONG JUMP (Code 02H xxH xxH) auf den Programmcode beginnen beginnen, werden sie beim Neueinschalten der Spannungsversorgung oder RESET automatisch gestartet. Ein Programm kann nur durch löschen des RAM-Speichers aus diesem entfernt werden. Unter den Unterprogrammen befinden sich solche, die bei Programmbeendigung den LONG JUMP entfernen und dadurch die Autostartkennung löschen. Der LONG JUMP und die Interrupt Vektoren befinden sich im Quellcode der Programme und werden mit Übertragen. Benötigt ein Programm keine Interrupts, sollte der dafür vorgesehene Speicherplatz trotzdem reserviert bleiben.

Programmbeispiel:

```
0006  0000                    ZSECH         EQU    0E95h  ;UP-Zeit  1/10s
0007  0000                    OPLA1         EQU    0F36h  ;UP-Ausgabe  AW-51/1
0008  0000
0009  0000                    org  8000h           ;RAM Anfangsadresse
0010  8000  02  8100           ljmp  START         ;Sprung auf Programmcode
0011  8003                                         ;* kein Interrupt
0012  8003                    org  8100h           ;Anfang Programmcode
0013  8100                    START:
0014  8100  7F  01             mov  r7,#1          ;1.Ausgabewert
0015  8102                    PLOOP:
0016  8102  12  0F36           lcall  OPLA1        ;Ausgabe
0017  8105  7F  02             mov  r7,#02         ;Zeitwert  0,2 Sekunden
0018  8107  12  0E95           lcall  ZSECH        ;Warten
0019  810A  EF                 mov  a,r7
0020  810B  23                 rl  a               ;Neuer Ausgabewert
0021  810C  FF                 mov  r7,a
0022  810D  80  F3             sjmp  PLOOP         ;ständig wiederholen
0023  810F
```

3.5 Download

Vor dem Download ist die Schnittstelle des Leitrechners auf die erforderlichen Parameter einzustellen. Die Übertragungsparameter sind 4800Baud, 8Datenbit, 1Sopbit ohne Paritätsprüfung. Ist der Leitrechner ein IBM-kompatibler PC, geschieht dies durch das MODE-Kommando.

MODE COMx: 48,n,8,1 [Enter]

COMx ist die verwendete Schnittstelle COM1, COM2, COM3 oder COM4

Die Übertragung des Programms kann beim PC mit dem COPY-Kommando durchgeführt werden.

COPY name.HEX COMx: [Enter]

name ist der Name der INTEL-HEX Datei

Ist bei der Übertragung ein Fehler aufgetreten, so blinkt die TXD-LED des Experimentierboards solange bis ein RESET des Boards durchgeführt wird. War die Übertragung einwandfrei, wird das Anwenderprogramm, sofern es ein Autostartprogramm ist, unmittelbar nach dem Download ausgeführt.

3.6 Das INTEL-HEX Format

Das INTEL-HEX Format ist ein Dateiformat speziell zur Übertragung speicherorientierter Daten. Dabei werden nicht nur die Daten, sondern auch deren Speicheradresse mitübertragen. Die Übertragung geschieht im ASCII-Code in RECORDS (Blöcken). Jeder RECORD ist mit einem Startzeichen und einer Prüfsumme versehen, womit Übertragungsfehler aufgedeckt werden. Die Prüfsumme ist die Summe der Zweierkomplemente aller Byte eines RECORDS mit Ausnahme des Startzeichens. Startzeichen ist der Doppelpunkt (:). Der letzte Record (Endrecord) ist immer gleich aufgebaut (:00000001FF).

Beispieldatei von 3.4:

```
:03800000028100FA
:0D810000740112011A78021200CE2380F5DE
:00000001FF
```

Inhalt der Datei:Jedes Zeichen liegt innerhalb der Datei als ASCII-Zeichen.

:0380.....0FA = 3AH 30H 33H 38H 30H 30H 46H 41H 0DH 0Ah

:0000001FF = 3AH 30H 30H 30H 30H 30H 30H 31H 46H 46H 0DH 0AH

Aufbau des INTEL-HEX Formats: **: Anzahl Adresse Typ Daten Prüfsumme CR LF**

Bereich für Prüfsumme

:	= Startzeichen
Anzahl	= Anzahl der Datenbyte im Record. Maximal 255
Adresse	= 16 Bit Speicheradresse des 1. Datenbyte des Record
Typ	= Recordtyp 00 = Datenrecord, 01 = Endrecord
Daten	= Die einzelnen Datenbyte des Record
Prüfsumme	= Die Summe der Zweierkomplemente aller davorliegenden Byte des Record ausser ":".
CR	= Carriage Return (ASCII 0DH)
LF	= Line Feed (ASCII 0AH)

3.7 Unterprogramme

Im Eprom befinden sich eine Anzahl Unterprogramme, die in den Anwenderprogrammen aufgerufen werden. Der Aufruf erfolgt durch den LONG CALL Befehl (LCALL adr.). Werden Unterprogramme benötigt die nicht vorhanden sind, können diese jederzeit im RAM angelegt werden. Im Eprom ist der Speicherbereich von 2000H bis 7FFFH für Erweiterungen des Anwenders reserviert. Sofern ein EPROM-Programmiergerät verfügbar ist, können in diesen Bereich eigene Unterprogramme hinterlegt werden.

Die Unterprogramme können durch einen Aufruf mit ihrer Adresse (LCALL adr.) gestartet werden. Dieser Aufruf wird schnell ausgeführt und ist in seiner Laufzeit berechenbar. Bei Änderungen des Programms im EPROM können sich die Adressen der Unterprogramme ändern, wodurch dann die Unterprogrammaufrufe an falsche Adressen gerichtet sind.

Beispiel:

```
0000  11E0          OPLA1  EQU    0F36h
0001
0002  7F  3E        mov  R7,#3Eh        ;Ausgabewert  laden
0003  12  0F36      lcall  OPLA1        ;Ausgabe  aufrufen
```

In den folgenden Tabellen sind die einzelnen Unterprogramm im EPROM der IS51 aufgeführt. Die Tabelleneinträge haben folgende Bedeutung:

Name: Der Name des Unterprogramms

Nr.: Die Nummer des Unterprogramms

Adresse: Die Speicheradresse des Unterprogramms

Funktion: Eine kurze Funktionsbeschreibung des Unterprogramms

Aufruf: Die beim Aufruf erwarteten Parameter

Rückgabe: Die Rückgabewerte des Unterprogramms bei fehlerfreier Ausführung

Fehler: Die Rückgabe bei fehlerhafter Ausführung

Ändert: Die durch das Unterprogramm verursachten Änderungen

RegBank: Die Registerbank, die zum Aufruf aktiv sein muß

Ausgabefunktionen:

Name:	**OPLA1**	**Nr.: 00H**	**Adresse: 0F36H**
Funktion:	Inhalt des R7 zu den LED's der Erweiterungsplatine IS51-E/A1ausgeben. Der Wert wird zusätzlich in der externen Datenspeicherzelle EOUTPORT1 (8030H) abgelegt.		
Aufruf:	Ausgabebyte in R7.		
Rückgabe:	-		Fehler: -
Ändert:	Zustand der Ausgabe, EOUTPORT1 (8030H)		RegBank: 0,1,2 oder 3

Name:	**OPLA1BS**	**Nr.: 01H**	**Adresse: 17E9H**
Funktion:	Ein Bit am Ausgabeport der IS51-E/A1 setzen.		
Aufruf:	R7 = Bitmaske (2).		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Ausgabe, EOUTPORT1 (8030H)		RegBank: 0,1,2 oder 3

Name:	**OPLA1BR**	**Nr.: 02H**	**Adresse: 1804H**
Funktion:	Ein Bit am Ausgabeport der IS31-E/A1 rücksetzen.		
Aufruf:	R7 = Bitmaske (2).		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Ausgabe, EOUTPORT1 (8030H)		RegBank: 0,1,2 oder 3

Name:	**OPLA1BC**	**Nr.: 03H**	**Adresse: 1822H**
Funktion:	Ein Bit am Ausgabeport der IS51-E/A1 invertieren.		
Aufruf:	R7 = Bitmaske (2).		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Ausgabe, EOUTPORT1 (8030H)		RegBank: 0,1,2 oder 3

Name:	**SEROA**	**Nr.: 10H**	**Adresse: 16C6H**
Funktion:	Inhalt des R7 zum ersten seriellen Erweiterungsport ausgeben. Daten = P1.0, Load = P1.1, Takt = P1.2.		
Aufruf:	Ausgabebyte in Register 7.		
Rückgabe:	-		Fehler: -
Ändert:	Zustand der Ausgabe, EOUTPORT1		RegBank: 0,1,2 oder 3

Name:	**SERAUSG**	**Nr.: 11H**	**Adresse: 1712H**
Funktion:	Serielle Ausgabeerweiterung bis max. 8 Ports. Beginnend mit dem Inhalt von EOUTPORT1 werden der Reihe nach die Inhalte der Rettungszellen zu den Erweiterungsports ausgegeben. Die Anzahl der Ports wird in R7 erwartet. Daten = P1.0, Load = P1.1, Takt = P1.2.		
Aufruf:	R7 = Anzahl 1-8, Ausgabewerte in den Rettungszellen EOUTPORT 1-8.		
Rückgabe:	-		Fehler: -
Ändert:	Zustand der Ausgabeports		RegBank: 0,1,2 oder 3

Name:	**SERPBS**	**Nr.: 12H**	**Adresse: 1841H**
Funktion:	Ein Bit in einer Ausgabe-Rettungszelle (EOUTPORT 1-8) setzen.		
Aufruf:	R5 = Portnummer 1-8, R7 = Bitmaske (2)		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Rettungszelle (EOUTPORT 1-8)		RegBank: 0,1,2 oder 3

Name:	**SERPBR**	**Nr.: 13H**	**Adresse: 185FH**
Funktion:	Ein Bit in einer Ausgabe-Rettungszelle (EOUTPORT 1-8) rücksetzen.		
Aufruf:	R5 = Portnummer 1-8, R7 = Bitmaske (2)		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Rettungszelle (EOUTPORT 1-8)		RegBank: 0,1,2 oder 3

Name:	**SERPBC**	**Nr.: 14H**	**Adresse: 187FH**
Funktion:	Ein Bit in einer Ausgabe-Rettungszelle (EOUTPORT 1-8) invertieren.		
Aufruf:	R5 = Portnummer 1-8, R7 = Bitmaske (2)		
Rückgabe:	-		Fehler: -
Ändert:	R7, Zustand der Rettungszelle (EOUTPORT 1-8)		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Eingabefunktionen:

Name:	**IPLA1**	**Nr.: 20H**	**Adresse: 0F52H**
Funktion:	Schalterstellung der Erweiterungsplatine IS51-E/A1 in Register 7 einlesen.		
Aufruf:	-		
Rückgabe:	Schalterstellung in R7		Fehler: -
Ändert:	R7		RegBank: 0,1,2 oder 3

Name:	**IPLA1BIT**	**Nr.: 21H**	**Adresse: 189CH**
Funktion:	Einzelbit am Eingabeport (Einzelschalter) IS51-E/A1 lesen.		
Aufruf:	R7 = Bitnummer 0-7.		
Rückgabe:	CY = Bit		Fehler: -
Ändert:	CY		RegBank: 0,1,2 oder 3

Name:	**SERIA**	**Nr.: 30H**	**Adresse: 16EEH**
Funktion:	Wert vom ersten seriellen Eingabe-Erweiterungsport in Register 7 einlesen. Takt = P1.3, Load = P1.4, Daten = P1.5.		
Aufruf:	-		
Rückgabe:	Eingabewert in Register 7.		Fehler: -
Ändert:	R7, EINPORT1 (8034H)		RegBank: 0,1,2 oder 3

Name:	**SEREING**	**Nr.: 31H**	**Adresse: 174CH**
Funktion:	Beginnend mit dem Erweiterungsport 1 werden die Eingabeports der Reihe nach in die Rettungszellen ab EINPORT1 eingelesen. Takt = P1.3, Load = P1.4, Daten = P1.5.		
Aufruf:	R7 = Anzahl Ports 1-8.		
Rückgabe:	Eingabewerte in den Rettungszellen.		Fehler: -
Ändert:	Rettungszellen EINPORT1-8 (8038H- 803FH)		RegBank: 0,1,2 oder 3

Name:	**EWIBIT**	**Nr.: 32H**	**Adresse: 18BCH**
Funktion:	Bit aus Eingabeerweiterungs- Rettungszelle lesen.		
Aufruf:	R7 = Bitnummer 0-7, R5 = Portnummer 1-8.		
Rückgabe:	CY = Bit		Fehler: -
Ändert:	CY		RegBank: 0,1,2 oder 3

SIO-Funktionen:

Name:	**SIOABLOCK**	**Nr.: 40H**	**Adresse: 129BH**
Funktion:	Einen Block Bytes aus dem Programm/Datenspeicher der IS51über die serielle Schnittstelle senden. Die Speicheradresse des Blocks wird in R6 (H) und R7 (L) erwartet, die Anzahl der Bytes in R5. (von 8000h-FFFFh sind Programm- und Datenspeicher identisch)		
Aufruf:	R6/R7 = Anfangsadresse, R5 = Anzahl, Daten im Speicher		
Rückgabe:	-		Fehler: -
Ändert:	R5		RegBank: 0,1,2 oder 3

Name:	**SIODBLOCK**	**Nr.: 41H**	**Adresse: 1276H**
Funktion:	Einen Block Bytes aus dem Programm/Datenspeicher der IS51 über die serielle Schnittstelle senden. Die Anfangsadresse des Blocks wird in R6 (H) und R7 (L) erwartet. Das Ende des Datenblocks wird am ASCII-Zeichen "$" (24H) erkannt.		
Aufruf:	R6/R7 = Anfangsadresse, Daten im Speicher mit Endezeichen $		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0,1,2 oder 3

Name:	**CHRDTO1**	**Nr.: 48H**	**Adresse: 12BEH**
Funktion:	Ein Zeichen von serieller Schnittstelle lesen mit Timeout. Die Timeoutzeit wird in 10ms Schritten in R7 hexadezimal übergeben (01-FF = 0,01-2,55s., 00 = 2,56s.)		
Aufruf:	Timeoutzeit in R7		
Rückgabe:	R7 = Zeichen, CY = 0		Fehler: R7 = 00, CY = 1
Ändert:	R7, CY		RegBank: 0,1,2 oder 3

Name:	**CHRDTO2**	**Nr.: 49H**	**Adresse: 12EEH**
Funktion:	Ein Zeichen von serieller Schnittstelle lesen mit Timeout. Die Timeoutzeit wird in 100ms Schritten in R7 hexadezimal übergeben (01-FF = 0,1-25,5s., 00 = 25,6s.)		
Aufruf:	Timeoutzeit in R7		
Rückgabe:	R7 = Zeichen, CY = 0		Fehler: R7 = 00, CY = 1
Ändert:	R7, CY		RegBank: 0,1,2 oder 3

Name:	**BYTERD**	**Nr.: 4AH**	**Adresse: 0E66H**
Funktion:	Zwei ASCII-Zeichen von SIO lesen und in ein HEX-Byte gewandelt in Register 7 zurückgeben. Kein Timeout, wartet bis zwei Zeichen empfangen wurden.		
Aufruf:	-		
Rückgabe:	HEX-Byte in R7		Fehler: -
Ändert:	R7		RegBank: 0,1,2 oder 3

Zeitfunktionen:

Name:	**MINH**	**Nr.: 50H**	**Adresse: 0F07H**
Funktion:	Kehrt nach der angegebenen Wartezeit in 1 Minuten Schritten zum Hauptprogramm zurück. Der Zeitwert wird im Register 7 hexadezimal erwartet.		
Aufruf:	Zeitwert in Register 7 hexadezimal (01-FF = 1 - 255 min, 00 = 256 min)		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0

Name:	**SECH**	**Nr.: 51H**	**Adresse: 0EB1H**
Funktion:	Kehrt nach der angegebenen Wartezeit in 1 Sekunden Schritten zum Hauptprogramm zurück. Der Zeitwert wird im Register 7 hexadezimal erwartet.		
Aufruf:	Zeitwert in R7 hexadezimal (01-FF = 1 - 255s, 00 = 256s).		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0

Name:	**ZSECH**	**Nr.: 52H**	**Adresse: 0E95H**
Funktion:	Kehrt nach der angegebenen Wartezeit in 0,1 Sekunden Schritten zum Hauptprogramm zurück. Der Zeitwert wird im Register 7 hexadezimal erwartet.		
Aufruf:	Zeitwert in Register 7 hexadezimal (01-FF = 0,1 - 25,5 s, 00 = 25,6s)		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0

Name:	**HSECH**	**Nr.: 53H**	**Adresse: 0ED7H**
Funktion:	Kehrt nach der angegebenen Wartezeit in 0,01 Sekunden Schritten zum Hauptprogramm zurück. Der Zeitwert wird im Register 7 hexadezimal erwartet.		
Aufruf:	Zeitwert in Register 7 hexadezimal (01-FF = 0,01 - 2,55 s, 00 = 2,56s)		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0

Name:	**MSECH**	**Nr.: 54H**	**Adresse: 0EF3H**
Funktion:	Kehrt nach der angegebenen Wartezeit in 1 Millisekunden Schritten zum Hauptprogramm zurück. Der Zeitwert wird im Register 7 hexadezimal erwartet.		
Aufruf:	Zeitwert in Register 7 hexadezimal (01-FF = 1 - 255 ms, 00 = 256ms)		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0

Name:	**T0PAUSE**	**Nr.: 5AH**	**Adresse: 0B3DH**
Funktion:	Warten bis das T0-Bit "1" und dann wieder "0" wird (T0-Taste gedrückt und losgelassen).		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0,1,2 oder 3

Name:	**INT0PAUSE**	**Nr.: 5BH**	**Adresse: 0B44H**
Funktion:	Warten bis das INT0-Bit "1" und dann wieder "0" wird (INT0-Taste gedrückt und losgelassen).		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	-		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Codewandel-Funktionen:

Name:	**ASCHEX**	**Nr.: 60H**	**Adresse: 0E00H**
Funktion:	Inhalt des Register 7 von ASCII (30h-46h) in eine HEX-Tetrade (00-0F) umwandeln.		
Aufruf:	ASCII-Zeichen in Register 7		
Rückgabe:	HEX-Tetrade in Register 7 Low-Tetrade, CY = 0		Fehler: R7 = FFH
Ändert:	R7, CY		RegBank: 0,1,2 oder 3

Name:	**HEXASC**	**Nr.: 61H**	**Adresse: 0E37H**
Funktion:	Die beiden Tetraden des R7 von HEX (0-F) in je ein ASCII-Zeichen (30-46H) umwandeln.		
Aufruf:	HEX-Byte in Register 7		
Rückgabe:	2 ASCII: R6 = H, R7 = L		Fehler: -
Ändert:	R7, R6		RegBank: 0,1,2 oder 3

Name:	**HEXDEZ**	**Nr.: 62H**	**Adresse: 0F74H**
Funktion:	Eine zweistellige HEX-Zahl (00-FF) aus Register 7 in eine dreistellige BCD-Zahl (000 – 255) umwandeln. Ergebnis High in R6 (0000 xxxx), Low in R7 (xxxx xxxx).		
Aufruf:	Hexzahl in Register 7.		
Rückgabe:	R6 = H, R7 = L		Fehler: -
Ändert:	R7, R6		RegBank: 0

Name:	**HEX16DEZ**	**Nr.: 63H**	**Adresse: 0FA5H**
Funktion:	Eine vierstellige HEX-Zahl (0000-FFFF) aus R6 (H) und R7 (L) in eine fünfstellige BCD-Zahl (00000-65535) umwandeln. Ergebnis High in R5 (0000 xxxx), Mid in R6 (xxxx xxxx), Low in R7 (xxxx xxxx). R4 = 0		
Aufruf:	Hexzahl R6 = H, R7 = L		
Rückgabe:	R4 = 0, R5 = H, R4 = M, R3 = L		Fehler: -
Ändert:	R7, R6, R5, R4		RegBank: 0

Name:	**DEZHEX**	**Nr.: 64H**	**Adresse: 10B3H**
Funktion:	Inhalt des Register 7 von BCD (00-99) in eine Hexzahl (00-63H) umwandeln.		
Aufruf:	Register 7 ist BCD-Zahl		
Rückgabe:	Hexzahl in R7, CY = 0		Fehler: R7 = 0, CY = 1
Ändert:	R7, CY		RegBank: 0,1,2 oder 3

Name:	**DEZ16HEX**	**Nr.: 65H**	**Adresse: 111BH**
Funktion:	Die vierstellige BCD-Zahl (0000-9999) aus R6 (H) und R7 (L) in eine vierstellige Hexzahl (0000-27F0) umwandeln. Das Ergebnis wird in R6 (H) und R7 (L) zurückgegeben.		
Aufruf:	BCD-Zahl, R6 = H, R7 = L		
Rückgabe:	Hexzahl, R6 = H, R7 = L, CY = 0		Fehler: R7 = 00, R6 = 00, CY = 1
Ändert:	R7, R6, CY		RegBank: 0

Name:	**BCH1TO7**	**Nr.:66H**	**Adresse: 1320H**
Funktion:	Umwandlung einer Hextetrade (00-0F) aus R7-Low in 7-Segment Code. Das Ergebnis wird im Register 7 zurückgegeben.		
Aufruf:	R7 = 00-0FH		
Rückgabe:	R7 = 7-Segment Code		Fehler: -
Ändert:	R7		RegBank: 0,1,2 oder 3

Name:	**BCH2TO7**	**Nr.: 67H**	**Adresse: 1349H**
Funktion:	Umwandlung eines Hexbyte (00-FF) aus Register 7 in zwei 7-Segment Codes. Das Ergebnis wird in R6 (H) und R7 (L) zurückgegeben.		
Aufruf:	Register 7 = 00-FFH		
Rückgabe:	7-Segment Codes: R6 = H, R7 = L		Fehler: -
Ändert:	R7, R6		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Arithmetikfunktionen:

Name:	**DIV16**	**Nr.: 80H**	**Adresse: 11A1H**
Funktion:	Die 16Bit Hexzahl aus R6(H) und R7 (L), durch die 8Bit Hexzahl in R5 ganzzahlig teilen. Das Ergebnis wird in R6(H) und R7 (L), der Divisionsrest in R5 zurückgegeben.		
Aufruf:	Dividend: R6 = H, R7 = L. Divisor: R5		
Rückgabe:	Quotient: R6 = H, R7 = L. Rest: R5, CY = 0		Fehler: CY = 1-
Ändert:	R7, R6, R5, CY		RegBank: 0

Name:	**DIV32**	**Nr.: 81H**	**Adresse: 13D6H**
Funktion:	Die 32Bit Hexzahl aus R4 (H), R5, R6 und R7 (L) wird durch die 16Bit Hexzahl in R2 (H) und R3 (L) geteilt. Das ganzzahlige Ergebnis wird in R4 (H), R5, R6 und R7 (L), der Divisionsrest in R2 (H) und R3 (L) zurückgegeben. (!!! Nicht für C-Programme verwenden !!!!)		
Aufruf:	Zahl A: R4, R5, R6, R7. Zahl B: R2, R3		
Rückgabe:	Quotient: R4, R5, R6, R7. Rest: R2,R3 CY = 0		Fehler: -CY = 1
Ändert:	R7, R6, R5, R4, R3, R2, CY		RegBank: 0

Name:	**MUL16**	**Nr.: 82H**	**Adresse: 1373H**
Funktion:	Die 16Bit Hexzahl aus R6 (H) und R7 (L) wird mit der 16Bit Hexzahl in R4 (H) und R5 (L) multipliziert. Das hexadezimale Ergebnis wird in R4 (H), R5, R6 und R7 (L) zurückgegeben.		
Aufruf:	Zahl A: R6 = H, R7 = L. Zahl B: R4 = H, R5 = L		
Rückgabe:	R4, R5, R6, R7		Fehler: -
Ändert:	R7, R6, R5, R4		RegBank: 0

Name:	**CMP16**	**Nr.: 83H**	**Adresse: 124AH**
Funktion:	Die 16Bit Hexzahl A in R6/R7 wird mit der 16Bit Hexzahl B in R4/R5 verglichen. A > B - CY=0, R7=1. A = B - CY=0, R7=0. A < B - CY=1, R7=2.		
Aufruf:	Zahl A: R6 = H, R7 = L. Zahl B: R4 = H, R5 = L		
Rückgabe:	Ergebnis: R7, CY		Fehler: -
Ändert:	R7, CY		RegBank: 0

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Beendigungsfunktionen:

Name:	**CLRAUTO**	**Nr.: 90H**	**Adresse: 135FH**
Funktion:	Autostart-Kennung löschen.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	Autostart-Kennung		RegBank: 0,1,2 oder 3

Name:	**T0END**	**Nr.: 91H**	**Adresse: 0F5CH**
Funktion:	Programm wird durch T0 = 1 beendet. Autostart-Kennung wird gelöscht.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	Autostart-Kennung		RegBank: 0,1,2 oder 3

Name:	**T1END**	**Nr.: 92H**	**Adresse: 0F68H**
Funktion:	Programm wird durch T1 = 1 beendet. Autostart-Kennung wird gelöscht.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	Autostart-Kennung		RegBank: 0,1,2 oder 3

Name:	**INT0END**	**Nr.: 93H**	**Adresse: 0F6CH**
Funktion:	Programm wird durch INT0 = 1 beendet. Autostart-Kennung wird gelöscht.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	Autostart-Kennung		RegBank: 0,1,2 oder 3

Name:	**INT1END**	**Nr.: 94H**	**Adresse: 0F70H**
Funktion:	Programm wird durch INT1 = 1 beendet. Autostart-Kennung wird gelöscht.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	Autostart-Kennung		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Befehlserweiterungs-Funktionen:

Name:	**PUSHREG**	**Nr.: A0H**	**Adresse: 1679H**
Funktion:	Die Inhalte von R0-7, DPTR, PSW, A und B in die Rettungszellen 8070-807CH schreiben. Es werden die Register 0-7 der aktiven Registerbank benutzt.		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	8070H-807CH		RegBank: 0,1,2 oder 3

Name:	**POPREG**	**Nr.: A1H**	**Adresse: 1642H**
Funktion:	Die Inhalte der Rettungszellen 8070-807CH in die Register R0-7, DPTR, PSW, A und B schreiben. Es werden die Register 0-7 der aktiven Registerbank benutzt.		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	R0-7, DPTR, PSW, A und B		RegBank: 0,1,2 oder 3

Name:	**XCHREG**	**Nr.: A2H**	**Adresse: 15D0H**
Funktion:	Die Inhalte der Rettungszellen 8070-807CH mit den Inhalten von R0-7, DPTR, PSW, A und B tauschen. Es werden die Register 0-7 der aktiven Registerbank benutzt.		
Aufruf:	.		
Rückgabe:	.		Fehler: -
Ändert:	R0-7, DPTR, PSW, A, B, 8070H-807CH		RegBank: 0,1,2 oder 3

Name:	**PUSHDPTR**	**Nr.: A3H**	**Adresse: 1555H**
Funktion:	Den DPTR in die Rettungszellen 8078H (DPL) und 8079H (DPH) schreiben.		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	8078H, 8079H		RegBank: 0,1,2 oder 3

Name:	**POPDPTR**	**Nr.: A4H**	**Adresse: 1574H**
Funktion:	Den Inhalt der Rettungszellen 8078H und 8079H in den DPTR schreiben.		
Aufruf:			
Rückgabe:			Fehler: -
Ändert:	DPTR		RegBank: 0,1,2 oder 3

Name:	**XCHDPTR**	**Nr.: A5H**	**Adresse: 1589H**
Funktion:	Die Inhalte der Rettungszellen 8078H und 8079H und des DPTR tauschen.		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	DPTR, 8078H, 8079H		RegBank: 0,1,2 oder 3

Name:	**DECDPTR**	**Nr.: A6H**	**Adresse: 15B6H**
Funktion:	Den Inhalt des DPTR dekrementieren (DPTR - 1).		
Aufruf:	-		
Rückgabe:	-		Fehler: -
Ändert:	DPTR		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Sonstige:

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Displayfunktionen:

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

A/D-D/A Funktionen:

Name:	**ADWI8**	**Nr.: C0H**	**Adresse: 1914H**
Funktion:	Eine 8Bit Messung mit dem A/D-Wandler ADWI/1 durchführen. Der Messwert wird in Register 7 zurückgegeben und in ADWERT (807E/7Fh) abgelegt. Messbereich -1,28V bis +1,28V=-128 bis +128, Auflösung 10mV.		
Aufruf:			
Rückgabe:	R7		Fehler: -
Ändert:	Timer 0, A		RegBank: 0,1,2 oder 3

Name:	**ADWI11**	**Nr.: C1H**	**Adresse: 1973H**
Funktion:	Eine 11Bit Messung mit dem A/D-Wandler ADWI/1 durchführen. Der Messwert wird in R6 (H) und R7 (L) zurückgegeben und in ADWERT (807E/7Fh) abgelegt. Messbereich -1,28V bis +1,28V=-1280 bis +1280, Auflösung 1mV.		
Aufruf:	-		
Rückgabe:	R6 = H, R7 = L		Fehler: -
Ändert:	Timer 0, R7, R6		RegBank: 0,1,2 oder 3

Name:	**ADWI14**	**Nr.: C2H**	**Adresse: 19D4H**
Funktion:	Eine 14Bit Messung mit dem A/D-Wandler ADWI/1 durchführen. Der Messwert wird in R6 (H) und R7 (L) zurückgegeben und in ADWERT (807E/7Fh) abgelegt. Messbereich -1,28V bis +1,28V=-12800 bis +12800- Auflösung 0,1mV.		
Aufruf:	-		
Rückgabe:	R7 = H, R6 = L		Fehler: -
Ändert:	Timer 0, R7, R6		RegBank: 0,1,2 oder 3

Name:	**SAR8B2**	**Nr.: C3H**	**Adresse: 1783H**
Funktion:	Eine 8Bit Messung mit dem A/D-Wandler SAR8B2 durchführen. Der Messwert wird in Register 7 zurückgegeben, und am Ausgabeport der Platine angezeigt.		
Aufruf:	-		
Rückgabe:	Messwert in Register 7.		Fehler: -
Ändert:	R7, Anzeige-LED's		RegBank: 0

Name:	**AD3162INST**	**Nr.: C8H**	**Adresse: 1AAEH**
Funktion:	Die Interruptsteuerung für den A/D-Wandler AD3162 einrichten. Es wird der externe Interrupt INT0 verwendet. In R7 wird die Priorität (0=niedrig, 1=hoch) erwartet.		
Aufruf:	Priorität: R7		
Rückgabe:	-		Fehler: -
Ändert:	R7, A, DPTR		RegBank: 0,1,2 oder 3

Name:	**ADREAD**	**Nr.: CFH**	**Adresse: 18E8H**
Funktion:	Einen Messwert aus ADWERT (807E/7Fh) lesen und in R6/R7 zurückgeben. Ist kein Messwert vorhanden wird 8000h zurückgegeben. Die Wandlerroutinen legen ihren Messwert auch in ADWERT ab.		
Aufruf:	-		
Rückgabe:	R6 = H, R7 = L		Fehler: -
Ändert:	R7, R6		RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

Name:		**Nr.:**	**Adresse:**
Funktion:			
Aufruf:			
Rückgabe:			Fehler: -
Ändert:			RegBank: 0,1,2 oder 3

3.7.1 Kurzübersicht der Unterprogramme

Arithmetik:

Programmbeendigung:

Befehlserweiterung:

Display-Funktionen:

A/D-D/A Wandlung:

Sonstige:

3.8 Das Treiberprogramm IS51.COM

Mit Hilfe dieses Treibers kann die Hardware der IS51 von einem PC, mit Betriebssystem MS-DOS, bedient werden. Der Treiber wird auf dem PC installiert. Er benutzt den PC-Interrupt mit der Nummer 60H.

Installation:

IS51/I	Treiber installieren (Grundeinstellung)
IS51/I /P=<Schnittstelle> /B=<Baudrate> /S=<Stopbit>	Treiber installieren mit Parametern
IS51/E	Treiber aus dem Speicher entfernen
IS51/?	Hilfe zur Installation

Es dürfen ein oder mehrere Parameter angegeben werden. Wird kein Parameter angegeben, so wird in Grundeinstellung (COM1, 4800Baud, 8Datenbit und 1Stopbit) installiert.

Parameter darf sein:

<Schnittstelle>:	COM1, COM2, COM3 oder COM4
<Baudrate>:	110, 150, 300, 600, 1200, 2400, 4800 oder 9600 Baud
<Stopbit>:	1 oder 2

Beispiele:

IS51/I	(COM1, 4800 Baud, 1 Stopbit = Grundeinstellung)
IS51/I/P=COM2	(COM2, 4800 Baud, 1 Stopbit)
IS51/I/B=2400/S=2	(COM1, 2400 Baud, 2 Stopbit)

Nach erfolgreicher Installation des Treibers erscheint am Bildschirm eine Meldung mit Angabe der eingestellten Schnittstellenparameter. Die Funktionen des Treiberprogramms werden über den Softwareinterrupt INT 60H aufgerufen. Dabei erfolgt die Übergabe der Funktionsparameter in den Registern und Speicherzellen des PC. Die Funktionen des Treibers sind im folgenden Abschnitt detailliert beschrieben.

Beispielprogramm:

Ein Byte zum Ausgabeport der Erweiterungsplatine IS51-E/A1 ausgeben.

```
MOV  AX,2200h        ;Funktionsnummer laden
MOV  BL,5Eh          ;Ausgabebyte laden
MOV  DH,00h          ;Port wählen
INT  60h             ;Treiber aufrufen (Gewählte Funktion ausführen)
INT  20h             ;Programm beenden
```

3.9 Die Funktionen des Treibers IS51.COM

Systemsteuerung

Funktion: 01H	Identifizierung
Ermittelt den Typ des eingesetzten Mikrokontrollers. Liefert als Ergebnis den CPU-Typ und die Monitorprogramm Version zurück. Version: 110d = Version 1.10, 115d = Version 1.15 usw. Typ: 00h = 80x1, 01h = 80x2, 10h = 80Cx1, 11h = 80Cx2	
Aufruf:	Rückgabe:
AX = 0100H	Kein Fehler: BH = CPU-Typ, BL = Version, CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:

Speicher

11H	Externen Speicher lesen	1AH	Indirekt Adresse lesen
12H	Externes RAM schreiben	1BH	Indirekt Adresse schreiben
14H	Programm starten	1CH	Byte lesen
15H	Unterprogramm aufrufen	1DH	Byte schreiben
18H	Direkt Adresse lesen		
19H	Direkt Adresse schreiben		

Funktion: 11H	Externen Speicher lesen
Aus dem externen Speicher der IS51 werden Daten gelesen. Der Speicher der IS51 erstreckt sich von 0000H - FFFFH. Die gelesenen Daten werden im Speicher des PC abgelegt.	
Aufruf:	Rückgabe:
AX = 1100h BX = Quelladresse im IS51 RAM CX = Anzahl der Bytes DX = Zieladresse im PC-RAM	Kein Fehler: CY = 0, Daten im PC-Speicher Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 12H	Externes RAM schreiben
In den externen RAM-Speicher der IS51 werden die übertragenen Daten geschrieben. Der RAM-Bereich der IS51 erstreckt sich von 8000H - FFFFH. Die Daten werden aus dem Speicher des PC geholt.	
Aufruf:	Rückgabe:
AX = 1200h BX = Zieladresse im IS51 RAM CX = Anzahl der Bytes DX = Quelladresse im PC-RAM	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 14H	Programm starten
Ein Anwenderprogramm, das sich im Arbeitsspeicher der IS51 befindet starten.	
Aufruf:	Rückgabe:
AX = 1400h BX = Startadresse des Programms	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 15H	Unterprogramm aufrufen
Ein Unterprogramm, das sich im Arbeitsspeicher der IS51 befindet aufrufen. Es wird die Adresse und die Funktionsnummer übergeben. Die Funktionsnummer wird nur bei Adresse 0100H benötigt.	
Aufruf:	Rückgabe:
AX = 1500h DL = Funktionsnummer des Unterprogramms BX = Adresse des Unterprogramms	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 18H	Direkt Adresse lesen
Aus dem internen direkt adressierbaren RAM-Speicher und den SFR des Mikrokontrollers werden Daten gelesen (Adr. 00-FFH). Der Stackpointer (81H) liefert undefinierte Werte.	
Aufruf:	Rückgabe:
AX = 1800H BX = Zieladresse der Daten im PC CL = Anzahl der Datenbytes DH = Quelladresse in IS51	Kein Fehler: CY = 0, Daten im PC-Speicher Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 19H	Direkt Adresse schreiben
In den Mikrokontroller internen direkt adressierbaren RAM-Speicher und die SFR werden die übertragenen Daten geschrieben (Adr 00-FFH). Der Stackpointer (81H) kann nicht verändert werden.	
Aufruf:	Rückgabe:
AX = 1900H BX = Quelladresse der Daten im PC CL = Anzahl der Datenbytes DH = Zieladresse in IS51	Kein Fehler: CY = 0, Daten im Mikrokontroller Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 1AH	Indirekt Adresse lesen
Aus dem internen indirekt adressierbaren RAM-Speicher des Mikrokontrollers werden Daten gelesen.	
Aufruf:	Rückgabe:
AX = 1A00H BX = Zieladresse der Daten im PC CL = Anzahl der Datenbytes DH = Quelladresse in IS51	Kein Fehler: CY = 0, Daten im PC-Speicher Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 1BH	Indirekt Adresse schreiben
In den internen indirekt adressierbaren RAM-Speicher des Mikrokontrollers werden Daten geschrieben (Adr. 80-FFH).	
Aufruf:	Rückgabe:
AX = 1B00H BX = Quelladresse der Daten im PC CL = Anzahl der Datenbytes DH = Zieladresse in IS51	Kein Fehler: CY = 0, Daten im Mikrokontroller Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 1CH	Byte lesen
Ein Byte aus einer direkt adressierbaren Datenspeicherzelle oder ein SFR des Mikrokontrollers lesen (Adr. 00-FFH). Der Stackpointer (81H) liefert undefinierte Werte.	
Aufruf:	Rückgabe:
AX = 1C00h DH = Adresse (00-FFH)	Kein Fehler: CY = 0, BL = Byte Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 1DH	Byte schreiben
Ein Byte in eine direkt adressierbare Datenspeicherzelle oder ein SFR des Mikrokontrollers schreiben. (Adr. 00-FFH). Der Stackpointer (81H) kann nicht beschrieben werden.	
Aufruf:	Rückgabe:
AX = 1D00h DH = Adresse (00-FFH) BL = Byte	Kein Fehler: CY = 0, Byte im Mikrokontroller Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 1EH	-
Aufruf:	Rückgabe:

Funktion: 1FH	-
Aufruf:	Rückgabe:

Ports

21H	Byte lesen	25H	Bit setzen
22H	Byte schreiben	26H	Bit rücksetzen
		27H	Bit invertieren

Funktion: 21H	**Port Byte lesen**
Ein Byte vom gewählten Port lesen.	
Aufruf:	Rückgabe:
AX = 2100h DH = Portnummer (1)	Kein Fehler: CY = 0, BL = Byte Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 22H	**Port Byte ausgeben**
Ein Byte zum gewählten Port ausgeben.	
Aufruf:	Rückgabe:
AX = 2200h BL = Byte DH = Portnummer (1)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)
Funktion: 25H	**Port Bit setzen**
Ein oder mehrere Bit am gewählten Ausgabeport setzen. Die Bit werden über die Bitmaske ausgewählt.	
Aufruf:	Rückgabe:
AX = 2500h BL = Bitmaske (2) DH = Portnummer (1)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 26H	**Port Bit rücksetzen**
Ein oder mehrere Bit am gewählten Ausgabeport zurücksetzen. Die Bit werden über die Bitmaske ausgewählt.	
Aufruf:	Rückgabe:
AX = 2600h BL = Bitmaske (2) DH = Portnummer (1)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 27H	**Port Bit invertieren**
Ein oder mehrere Bit am gewählten Ausgabeport invertieren. Die Bit werden über die Bitmaske ausgewählt.	
Aufruf:	Rückgabe:
AX = 2700h BL = Bitmaske (2) DH = Portnummer (1)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

A/D-Wandler

45H A/D starten 46H A/D lesen

Funktion: 45H	A/D starten
Wandlung des angegebeben A/D-Wandlers starten. Darf nur aufgerufen werden, wenn der Wandler angeschlossen ist. Liefert den Meßwert oder FFFFH zurück. Bei langsamen Wandlern wird FFFFH zurückgegeben und wenn fertig, der Meßwert im externen Speicherwort ADWERT (807EH)abgelegt.	
Aufruf:	Rückgabe:
AX = 4500h BH = Wandlernr. (5)	Kein Fehler: CY = 0, DX = Rückgabe Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 46H	A/D Wert lesen
Einen Meßwert aus dem externen Speicherwort ADWERT (807EH) abholen. Ist ein Meßwert vorhanden wird dieser, sonst FFFFh zurückgegeben.	
Aufruf:	Rückgabe:
AX = 4600h	Kein Fehler: CY = 0, DX = Meßwert Fehler: CY = 1, AH = Fehlernummer (4)

Interrupt und Timer

51H Interrupt Vektor lesen
52H Interrupt Vektor setzen
53H Interrupt sperren
54H Interrupt freigeben

55H Zähler 0 lesen
56H Zähler 1 setzen
57H Zähler 0 sperren
58H Zähler 0 freigeben
5FH Zähler 0 programmieren

Funktion: 51H	Interrupt Vektor lesen
Liefert den Vektor des angegebenen Interrupt der IS51 zurück.	
Aufruf:	Rückgabe:
AX = 5100H DH = IntNr. (3)	Kein Fehler: CY = 0, BX = Interrupt Vektor Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 52H	Interrupt Vektor setzen
Setzt den Vektor des angegebenen Interrupt der IS51.	
Aufruf:	Rückgabe:
AX = 5200H BX = Vektor DH = IntNr. (3)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 53H	Interrupt sperren
Sperrt den angegebenen Interrupt der IS51.	
Aufruf:	Rückgabe:
AX = 5300H DH = IntNr. (3)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 54H	Interrupt freigeben
Gibt den angegebenen Interrupt der IS51 frei.	
Aufruf:	Rückgabe:
AX = 5400H DH = IntNr. (3)	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 55H	Zähler 0 lesen
Stand des Timer 0 Zählregisters auslesen.	
Aufruf:	Rückgabe:
AX = 5500H	Kein Fehler: CY = 0, BX = Zählerwert Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 56H	Zähler 0 setzen
Stand des Timer 0 Zählregisters setzen.	
Aufruf:	Rückgabe:
AX = 5600H BX = Zählerwert	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 57H	Zähler 0 sperren
Sperrt den Zähltakt für Zähler 0.	
Aufruf:	Rückgabe:
AX = 5700H	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion: 58H	Zähler 0 freigeben
Gibt den Zähltakt für Zähler 0 frei.	
Aufruf:	Rückgabe:
AX = 5800H	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion 5FH:	**Zähler 0 programmieren**
Den Timer 0 des Mikrokontrollers als Zähler programmieren. Die Betriebsart kann auf 16- oder 13Bit, oder als 8Bit autoreload Zähler eingestellt werden. ModeByte: 01 = 16Bit, 02 = 13Bit, 03 = 8Bit autoreload	
Aufruf:	Rückgabe:
AX = 5F00H BL = ModeByte	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:

Funktion:	
Aufruf:	Rückgabe:

Funktion:	
Aufruf:	Rückgabe:

Sonstige

Funktion:	
Aufruf:	Rückgabe:
	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:
	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:
	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:
	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

Funktion:	
Aufruf:	Rückgabe:
	Kein Fehler: CY = 0 Fehler: CY = 1, AH = Fehlernummer (4)

3.9.1 Erläuterungen zu den Anmerkungen (x)

(1) Portnummer
- 00H = Ausgabeport IS51-E/A1
- 01H = Eingabeport IS51-E/A1
- 02H = Port P1
- 03H = Port P3

(2) Bitmaske

0000 0001 = Bit0	0001 0000 = Bit4
0000 0010 = Bit1	0010 0000 = Bit5
0000 0100 = Bit2	0100 0000 = Bit6
0000 1000 = Bit3	1000 0000 = Bit7

Beispiel: 0010 0110 = Bit 1, 2 und 5

(3) IntNr.

00H = Externer Interrupt 0 (INT0)	03H = Timer 1 Overflow (TF1)
01H = Timer 0 Overflow (TF0)	04H = UART Interrupt (RI/TI)
02H = Externer Interrupt 1 (INT1)	05H = Timer 2 Overflow (TF2/EXF2)

(4) Fehlernummer

Nr.	Beschreibung	Abhilfe
00H	Kein Fehler aufgetreten	-
01H	Timeout / Zeitüberschreitung der SIO	Serielle Verbindung prüfen
03H	Anzahl Wiederholversuche überschritten	Kleinere Baudrate einstellen
10H	PC-SIO: Frame Error	PC neu booten
11H	PC-SIO: Overrun Error	Aufruf wiederholen
12H	PC-SIO: Parity Error	PC neu booten
13H	PC-SIO: Break Interrupt	Serielle Verbindung prüfen
20H	Unbekannte Funktionsnummer	Funktionsnummer in AH prüfen
22H	Adresse im PC-RAM zu hoch	Adresse + Anzahl größer FFFFH
23H	Adresse im IS51-RAM zu hoch	Adresse + Anzahl größer FFFFH
24H	Falscher Port-Parameter	Nur 00H und 01H erlaubt
25H	Port nicht definiert	Nur 00H, 01H, 02H und 03H erlaubt
26H	Port nicht definiert	Nur 00H, 02H und 03H erlaubt
27H	Port nicht definiert	Nur 02H und 03H erlaubt
28H	Falsche Wandlernummer	Nur 00H - 05H erlaubt
40H	Interrupt-Nr. nicht definiert	Nur 00H - 08H erlaubt
50H	Adresse im internen Speicher zu groß	Adresse prüfen
52H	Mode-Byte nicht definiert	Nur 01H, 02H und 03H erlaubt

(5) Wandlernr.

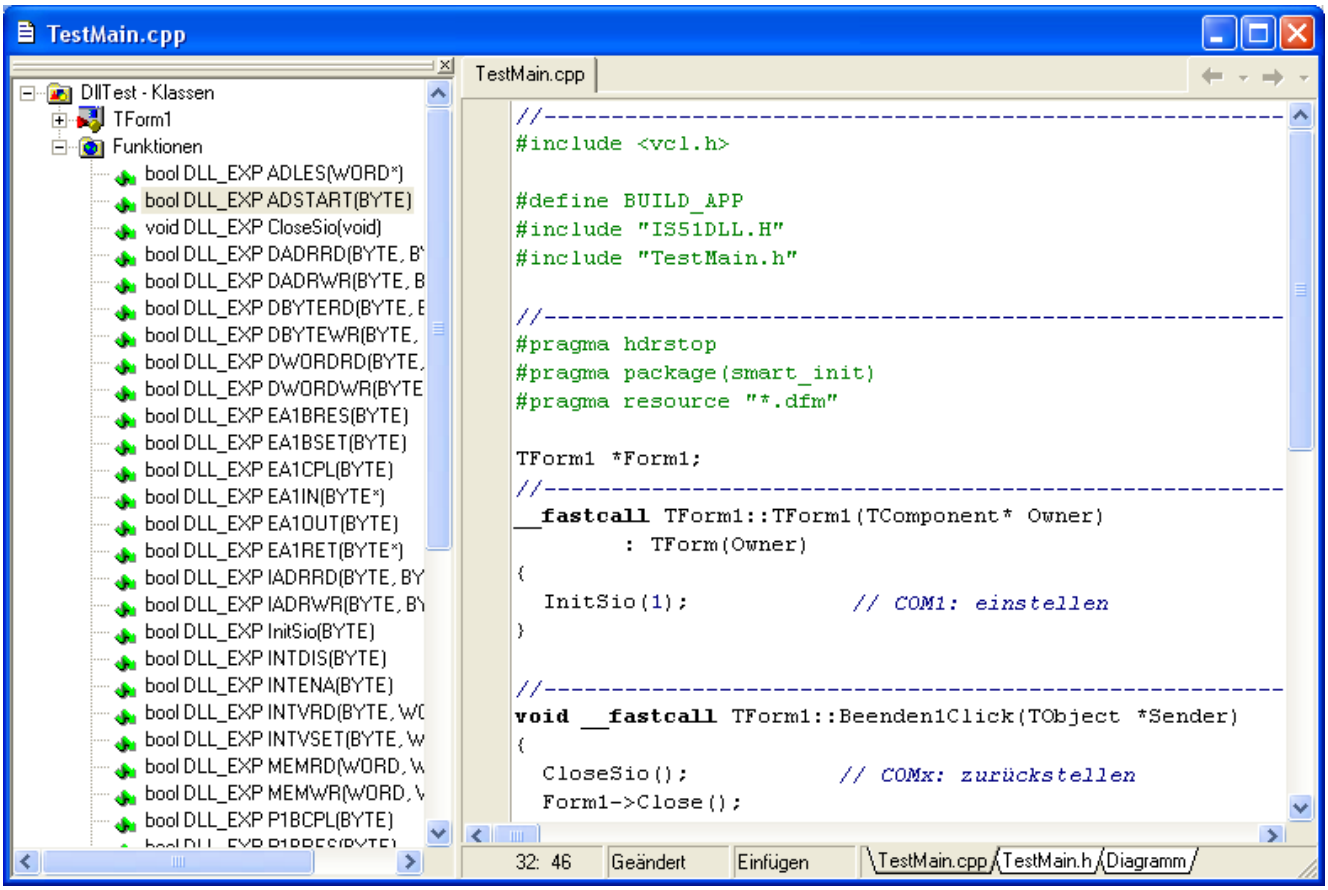
00 = ADWI8	04 =
01 = ADWI11	05 = CA3162
02 = ADWI14	06 =
03 = SAR8B2	07 =

3.9.2 Kurzübersicht der Treiberfunktionen

4 Die Bibliothek IS51DLL.LIB

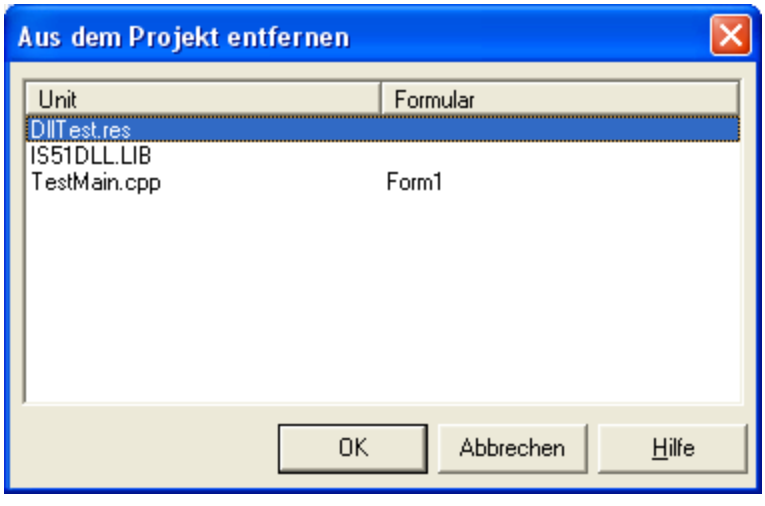
4.1 Allgemein

Die Bibliothek IS51DLL.LIB ermöglicht den Zugriff auf die Funktionen der IS51 von Programmen unter Windows. Sie übernimmt die Aufgaben des Treibers IS51.COM, welcher unter 3.8 beschrieben ist, auf Betriebssystem Ebene. Alle dort beschriebenen Funktionen sind implementiert und können genutzt werden. Um diese Funktionen nutzen zu können, ist die Bibliotheksdatei „IS51DLL.LIB“ mit in das Projekt des Anwenderprogramms des PC zu übernehmen und für die Namen und Parameter die Headerdatei „IS51DLL.H“ durch #include „IS51DLL.H“ ins Programm aufzunehmen. Neu hinzugekommen sind die Funktionen „InitSio“ und „CloseSio“ zum einstellen und wieder freigeben der benutzten seriellen PC-Schnittstelle.



Mit Hilfe der Bibliothek wird es Programmen möglich, die Funktionen der IS51 in PC-Programmen zu nutzen. So können beispielsweise mit dem PC Relais gesteuert oder Meßwerte erfasst werden. Alle Fähigkeiten der IS51 sind damit auch dem PC zugänglich.

Die Bilder zeigen ein Projekt in C++Builder von Borland, es ist auch möglich die Bibliothek in anderen Programmen zu nutzen.



4.1.1 Die Funktionen der Bibliothek

Alle Funktionen liefern als Funktionsergebnis einen boolschen Wert zurück. Wurde die Funktion fehlerfrei ausgeführt ist das Funktionsergebnis „true“ (1), sonst „false“ (0). Zum Datenaustausch werden Variablen und der Speicher des PC benutzt, welche genau, ist der Beschreibung der Funktionen zu entnehmen. Die Beschreibung ist in Funktionsgruppen gegliedert, welche die einzelnen Funktionen beinhalten.

Systemsteuerung

Name: **INITSIO** (Serielle Schnittstelle initialisieren)

Richtet die gewünschte serielle Schnittstelle mit den erforderlichen Parametern für die IS51 ein.

Definition: bool InitSio(BYTE)

Parameter: Nummer der gewünschten Schnittstelle. 1=COM1, 2=COM2....

Rückgabe: true/false

Beispiel:
```
if(InitSio(1))          // COM1: für IS51 einrichten
{ evtl. Fehlerbehandlung
}
```

Name: **CLOSESIO** (Serielle Schnittstelle schließen)

Stellt die ursprünglichen Einstellungen der seriellen Schnittstelle wieder her und gibt sie frei.

Definition: bool CloseSio(void)

Parameter: Nummer der gewünschten Schnittstelle. 1=COM1, 2=COM2....

Rückgabe: true/false

Beispiel:
```
if(CloseSio())          // COM: restaurieren und freigeben
```

Name: **ReadID** (Identifikation lesen)

Ermittelt den Typ des eingesetzten Mikrokontrollers. Liefert als Ergebnis den CPU-Typ und die Monitor-programm Version zurück. Version: 119 = Version 1.19, 120 = Version 1.20 usw. Typ: 00h = 80x1, 01h = 80x2, 10h = 80Cx1, 11h = 80Cx2.

Definition: bool ReadID(WORD*)

Parameter: Adresse der Zielvariable

Rückgabe: true/false, ID in Zielvariable

Beispiel:
```
WORD wID;               // Variable für Rückgabe
if(ReadID(&wID))        // ID in die Variable lesen
{ evtl. Fehlerbehandlung
}
```

Speicher

Name: **MEMRD** (Externen Speicher lesen)

Aus dem externen Speicher der IS51 werden Daten gelesen. Der Speicher der IS51 erstreckt sich von 0000H - FFFFH. Die gelesenen Daten werden im Speicher des PC abgelegt.

Definition: bool MEMRD(WORD wAdr, WORD wCount, BYTE *bTarget)

Parameter: 1.wAdr - Speicheradresse der IS51.

2. wCount - Anzahl Datenbyte

3. *bTarget - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
//64Byte aus IS51Adr. 8800h in Puffer lesen
      BYTE bPuffer[64];
      MEMRD(0x8800, 64, &bPuffer[0]);
```

Name: **MEMWR** (Externes RAM schreiben)

In den externen RAM-Speicher der IS51 werden die übertragenen Daten geschrieben. Der RAM-Bereich der IS51 erstreckt sich von 8000H - FFFFH. Die Daten werden im Speicher des PC erwartet.

Definition: bool MEMWR(WORD wAdr, WORD wCount, BYTE *bSource)

Parameter: 1.wAdr - Speicheradresse der IS51.

2. wCount - Anzahl Datenbyte

3. *bSource - Zeiger auf Quelladresse im PC

Rückgabe: true/false, -

Beispiel:
```
// 5Byte aus Puffer in IS51Adr. 8800h schreiben
      BYTE bPuffer[] = { 0,1,2,3,4 };
      MEMRD(0x8800, 5, &bPuffer[0]);
```

Name: **PRGSTART** (Programm starten)

Ein Anwenderprogramm, das sich im Arbeitsspeicher der IS51 befindet starten. Hiermit können beispielsweise Fremdstartprogramme gestartet werden. Die Funktion prüft nicht, ob ein Programm vorhanden ist!

Definition: bool PRGSTART (WORD *wPtr)

Parameter: 1.*wPtr – Zeiger auf Startadresse der IS51.

Rückgabe: true/false, -

Beispiel:
```
// Programm in IS51Adr. 8100h starten
    PRGSTART((WORD*)0x8100);
```

Name: **UPSTART** (Unterprogramm aufrufen)

Ein Unterprogramm, das sich im Arbeitsspeicher der IS51 befindet aufrufen. Es wird die Adresse des Unterprogramms übergeben.

Definition: bool UPSTART (WORD *wPtr)

Parameter: *wPtr – Zeiger auf SpeicherAdresse des Unterprogramms.

Rückgabe: true/false, -

Beispiel:
```
// Unterprogramm in IS51Adr. 1783h (SAR8B2) starten
    UPSTART((WORD*)0x1783);
```

Name: **DADRRD** (Direktadresse lesen)

Aus dem internen direkt adressierbaren RAM-Speicher und den SFR des Mikrocontrollers werden Daten gelesen (Adr. 00-FFH). Der Stackpointer (81H) liefert undefinierte Werte.

Definition: bool DADRRD (BYTE bAdr, BYTE bCount, BYTE *bPtr)

Parameter: 1.bAdr - Speicheradresse der IS51.

2. bCount - Anzahl Datenbyte

3. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
// 5 Byte ab Adresse 60 lesen
BYTE bData[5];
DADRRD(0x60,5,&bData);
```

Name: **DADRWR** (Direktadresse schreiben)

In den Mikrokontroller internen direkt adressierbaren RAM-Speicher und die SFR werden die übertragenen Daten geschrieben (Adr 00-FFH). Der Stackpointer (81H) kann nicht verändert werden.

Definition: bool DADRWR(BYTE bAdr, BYTE bCount, BYTE *bSource)

Parameter: 1.bAdr - Speicheradresse der IS51.

2. bCount - Anzahl Datenbyte

3. *bSource - Zeiger auf Quelladresse im PC

Rückgabe: true/false, -

Beispiel:
```
// 10 Byte ab Adresse 60h schreiben
BYTE bData[] = { 0,1,2,3,4,5,6,7,8,9 };
DADRWR(0x60,10,&bData);
```

Name: **IADRRD** (Indirektadresse lesen)

Aus dem internen indirekt adressierbaren RAM-Speicher des Mikrokontrollers werden Daten gelesen.

Definition: bool IADRRD(BYTE bAdr, BYTE bCount, BYTE *bPtr)

Parameter: 1.bAdr - Speicheradresse der IS51.

2. bCount - Anzahl Datenbyte

3. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
// 5 Byte ab Adresse A0h lesen
BYTE bData[5];
IADRRD(0xA0,5,&bData);
```

Name: **IADRWR** (Indirektadresse schreiben)

In den internen indirekt adressierbaren RAM-Speicher des Mikrokontrollers werden Daten geschrieben (Adr. 80-FFH).

Definition: bool IADRWR(BYTE bAdr, BYTE bCount, BYTE *bSource)

Parameter: 1.bAdr - Speicheradresse der IS51.

2. bCount - Anzahl Datenbyte

3. *bSource - Zeiger auf Quelladresse im PC

Rückgabe: true/false, -

Beispiel:
```
// 5 Byte ab Adresse A0h schreiben
BYTE bData[] = { 0,1,2,3,4 };
IADRWR(0xA0,5,&bData);
```

Name: **DBYTERD** (Direktadresse ein Byte lesen)

Ein Byte aus einer direkt adressierbaren Datenspeicherzelle oder ein SFR des Mikrokontrollers lesen (Adr. 00-FFH). Der Stackpointer (81H) liefert undefinierte Werte.

Definition: bool DBYTERD (BYTE bAdr, BYTE *bPtr)

Parameter: 1.bAdr - Speicheradresse der IS51.

2. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
// Ein Byte von Direktadresse 42h lesen
BYTE bData[5];
DBYTERD(0x42,5,&bData);
```

Name: **DBYTEWR** (Direktadresse ein Byte schreiben)

Ein Byte in eine direkt adressierbare Datenspeicherzelle oder ein SFR des Mikrokontrollers schreiben. (Adr. 00-FFH). Der Stackpointer (81H) kann nicht beschrieben werden.

Definition: bool DBYTEWR(BYTE bAdr, BYTE bByte)

Parameter: 1.bAdr - Direktadresse der IS51.

2. bByte - Zu schreibendes Datenbyte

Rückgabe: true/false, -

Beispiel:
```
// Byte 3Ah in Direktadresse 57h schreiben
BYTE bData[] = { 0,1,2,3,4 };
IADRWR(0x57,0x3A);
```

Ports

Name: **EA1IN** (Ein Byte vom Eingabeport der E/A-1 lesen)

Ein Byte von den Schaltern der Erweiterungsplatine E/A-1 lesen.

Definition: bool EA1IN(BYTE *bPtr)

Parameter: 1. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
//Schalterstellung E/A-1 lesen
BYTE bData;
EA1IN(&bData);
```

Name: **EA1OUT** (Ein Byte zum Ausgabeport der E/A-1 schreiben)

Ein Byte zu den Leuchtdioden der Erweiterungsplatine E/A-1 schreiben.

Definition: bool EA1OUT(BYTE bByte)

Parameter: 1.bByte - Zu schreibendes Datenbyte

Rückgabe: true/false, -

Beispiel:
```
// C9h zu den Leuchtdioden E/A-1 schreiben
EA1OUT(0xC9);
```

Name: **EA1RET** (Letzte Ausgabe zum Ausgabeport der E/A-1 zurücklesen)

Ließt die letzte Ausgabe zu den Leuchtdioden der Erweiterungsplatine E/A-1 zurück.

Definition: bool EA1RET(BYTE *bPtr)

Parameter: 1.*bPtr – Zeiger auf Zieladresse.

Rückgabe: true/false, -

Beispiel:
```
// Letzte Ausgabe zu bVar zurücklesen
BYTE bVar;
EA1BSET(&bVar);
```

N ame: **EA1BSET** (Bit am Ausgabeport der E/A-1 setzen)

Ein oder mehrere Leuchtdioden der Erweiterungsplatine E/A-1 einschalten.

Definition: bool EA1BSET(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=ein, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// alle roten Leuchtdioden einschalten
EA1BSET(0xC9);
```

Version 1.20

Name: **EA1BRES** (Bit am Ausgabeport der E/A-1 rücksetzen)

Ein oder mehrere Leuchtdioden der Erweiterungsplatine E/A-1 ausschalten.

Definition: bool EA1RES(BYTE *bPtr)

Parameter: 1.bByte – Bitmaske 1=aus, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// alle roten Leuchtdioden ausschalten
EA1BRES(0xC9);
```

Name: **EA1BCPL** (Bit am Ausgabeport der E/A-1 invertieren)

Ein oder mehrere Leuchtdioden der Erweiterungsplatine E/A-1 umschalten.

Definition: bool EA1CPL(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=aus, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// alle roten Leuchtdioden umschalten
EA1BCPL(0xC9);
```

Name: **P1IN** (Ein Byte vom Port 1 lesen)

Ein Byte von IS51-Port 1 in den Speicher des PC lesen.

Definition: bool P1IN(BYTE *bPtr)

Parameter: 1. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
// Port 1 nach bData lesen
BYTE bData;
P1IN(&bData);
```

Name: **P1OUT** (Ein Byte zum Port 1 schreiben)

Ein Byte zum IS51-Port 1 schreiben.

Definition: bool P1OUT(BYTE bByte)

Parameter: 1.bByte - Zu schreibendes Datenbyte

Rückgabe: true/false, -

Beispiel:
```
// 5Ah zum Port 1 schreiben
P1OUT(0x5A);
```

Name: **P1BSET** (Bit am Port 1 setzen)

Ein oder mehrere Bit am IS51-Port 1 einschalten.

Definition: bool P1BSET(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=ein, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// Die 4 Low-Bit am Port 1 einschalten
P1BSET(0x0F);
```

Name: **P1BRES** (Bit am Port 1 rücksetzen)

Ein oder mehrere Bit am IS51-Port 1 ausschalten.

Definition: bool P1BRES(BYTE *bPtr)

Parameter: 1.bByte – Bitmaske 1=aus, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// Die 4 Low-Bit am Port 1 ausschalten
P1BRES(0x0F);
```

Name: **P1BCPL** (Bit am Port 1 invertieren)

Ein oder mehrere Bit am IS51-Port 1 umschalten.

Definition: bool P1CPL(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=um, 0=unverändert

Rückgabe: true/false, -

Beispiel:
```
// Die 4 Low-Bit am Port 1 umschalten
P1BCPL(0x0F);
```

Name: **P3IN** (Ein Byte vom Port 3 lesen)

Ein Byte vom IS51-Port 3 in den Speicher des PC lesen.

Definition: bool P3IN(BYTE *bPtr)

Parameter: 1. *bPtr - Zeiger auf Zieladresse im PC

Rückgabe: true/false, Daten im Speicher des PC

Beispiel:
```
// Port 3 nach bData lesen
BYTE bData;
P3IN(&bData);
```

Version 1.20

Name: **P3OUT** (Ein Byte zum Port 3 schreiben)

Ein Byte zu IS51-Port 3 schreiben.

Definition: bool P3OUT(BYTE bByte)

Parameter: 1.bByte - Zu schreibendes Datenbyte

Rückgabe: true/false, -

Beispiel: // 5Ah zum Port 3 schreiben
P3OUT(0x5A);

N ame: **P3BSET** (Bit am Port 3 setzen)

Ein oder mehrere Bit am IS51-Port 3 einschalten.

Definition: bool P3BSET(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=ein, 0=unverändert

Rückgabe: true/false, -

Beispiel: // Die 4 Low-Bit am Port 3 einschalten
P3BSET(0x0F);

Name: **P3BRES** (Bit am Port 3 rücksetzen)

Ein oder mehrere Bit am IS51-Port 3 ausschalten.

Definition: bool P3RES(BYTE *bPtr)

Parameter: 1.bByte – Bitmaske 1=aus, 0=unverändert

Rückgabe: true/false, -

Beispiel: // Die 4 Low-Bit am Port 3 ausschalten
P3BRES(0x0F);

Name: **P3BCPL** (Bit am Port 3 invertieren)

Ein oder mehrere Bit am IS51-Port 3 umschalten.

Definition: bool P3CPL(BYTE bByte)

Parameter: 1.bByte – Bitmaske 1=um, 0=unverändert

Rückgabe: true/false, -

Beispiel: // Die 4 Low-Bit am Port 1 umschalten
P3BCPL(0xC9);

A/D-Wandler

Name: **ADSTART** (A/D-Wandler starten)

Wandlung des angegebeben A/D-Wandlers starten. Darf nur aufgerufen werden, wenn der Wandler angeschlossen ist. Liefert den Meßwert oder FFFFH zurück. Bei langsamen Wandlern wird FFFFH zurückgegeben und wenn fertig, der Meßwert im externen Speicherwort ADWERT (807EH)abgelegt.

Definition: bool ADSTART(BYTE bByte)

Parameter: 1.bByte – Nummer des A/D-Wandler

00 = ADWI8, 01 = ADWI11,

02 = ADWI14, 03 = SAR8B2,

04 = , 05 = CA3162

Rückgabe: true/false, -

Beispiel:
```
// ADWI14 Wandler starten
ADSTART(02);
```

Name: **ADLES** (A/D-Wandler Wert lesen)

Einen Meßwert aus dem externen Speicherwort ADWERT (807EH) abholen. Ist ein Meßwert vorhanden wird dieser, sonst 8000h zurückgegeben.

Definition: bool ADLES(WORD *wPtr)

Parameter: 1.bByte – Zeiger auf Zieladresse

Rückgabe: true/false, -

Beispiel:
```
// A/D-Wert lesen
int iWert;
ADLES(&iWert);
if(iWert == 0xFFFF)
{ Fehlerbehandlung }
```

Interrupt und Timer

Name: **INTVRD** (Interrupt Vektor lesen)

Liefert den Vektor des angegebenen Interrupt der IS51 zurück.

Definition: bool INTVRD (BYTE bINum, WORD *ptrVekt)

Parameter: 1. bINum – Interruptnummer

2. *ptrVekt – Zeiger auf Zieladresse

bINum:	00H = Externer Interrupt 0 (INT0),	01H = Timer 0 Overflow (TF0),
	02H = Externer Interrupt 1 (INT1),	03H = Timer 1 Overflow (TF1)
	04H = UART Interrupt (RI/TI),	05H = Timer 2 Overflow (TF2/EXF2)
	06H = Interrupt 6,	07H = Interrupt 7
	06H = Interrupt 8,	07H = Interrupt 9
	06H = Interrupt 10,	07H = Interrupt 11
	06H = Interrupt 12,	07H = Interrupt 12
	06H = Interrupt 14	

Rückgabe: true/false, -

Beispiel:
```
// Vektor Interupt 1 lesen
WORD wVektor;
INTVRD(0x02,&wVektor);
```

Name: **INTVSET** (Interrupt Vektor setzen)

Setzt den Vektor des angegebenen Interrupt der IS51.

Definition: bool INTVSET (BYTE bINum, WORD *ptrVekt)

Parameter: 1. bINum – Interruptnummer

2. *ptrVekt – Zeiger auf Quelladresse

Rückgabe: true/false, -

Beispiel:
```
// Vektor Timer 0 Overflow auf 8C00h setzen
WORD wVektor = 0x8C00;
INTVSET(0x01,&wVektor);
```

Name: **INTDIS** (Interrupt sperren)

Sperrt den angegebenen Interrupt der IS51.

Definition: bool INTDIS(BYTE bINum)

Parameter: 1. bINum – Interruptnummer

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 Overflow Interrupt sperren
INTDIS(0x01);
```

Name: **INTENA** (Interrupt freigeben)

Gibt den angegebenen Interrupt der IS51 frei.

Definition: bool INTENA (BYTE bINum)

Parameter: 1. bINum – Interruptnummer

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 Overflow Interrupt freigeben
INTENA(0x01);
```

Name: **ZLR0LES** (Zähler 0 lesen)

Stand des Timer 0 Zählregisters auslesen.

Definition: bool ZLR0LES (WORD *wPtr)

Parameter: 1. *wPtr – Zeiger auf Zieladresse im PC

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 Zählregister auslesen
WORD wVar;
ZLR0LES(&wVar);
```

Name: **ZLR0SET** (Zähler 0 setzen)

Stand des Timer 0 Zählregisters setzen.

Definition: bool ZLR0SET (WORD wWert)

Parameter: 1. wWert – Ladewert für Zählregister.

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 Zählregister mit 4000h setzen
ZLR0SET(0x4000);
```

Version 1.20

Name: **ZLR0PROG** (Zähler 0 programmieren)

Den Timer 0 des Mikrokontrollers als Zähler programmieren. Die Betriebsart kann auf 16Bit, 13Bit, oder als 8Bit autoreload Zähler eingestellt werden. ModeByte: 01 = 16Bit, 02 = 13Bit, 03 = 8Bit autoreload

Definition: bool ZLR0PROG (BYTE bByte)

Parameter: 1. bByte – Modebyte.

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 als 16Bit Zähler programmieren
ZLR0PROG(0x01);
```

Name: **ZLR0DIS** (Zähler 0 sperren)

Sperrt den Zähler 0 der IS51.

Definition: bool ZLR0DIS (void)

Parameter: -

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 sperren
ZLR0DIS();
```

Name: **ZLR0ENA** (Zähler 0 freigeben)

Gibt den Zähler 0 der IS51 frei.

Definition: bool ZLR0ENA (void)

Parameter: -

Rückgabe: true/false, -

Beispiel:
```
// Timer 0 freigeben
ZLR0ENA();
```

4.1.2 Kurzübersicht der DLL-Funktionen

5 Ein/Ausgabeplatine IS51-E/A1

Die Platine IS51- E/A1 ist als einfache Ein/Ausgabe für das MCS51 Experimentierboard konzipiert. Sie verfügt über einen 8Bit Ausgabeport an dem Leuchtdioden angeschlossen sind, einen 8 Bit Eingabeport mit Schaltern und zwei entprellte Tasten für die Signale INT0 und T0. Die Verbindung zum Experimentierboard wird mit einem 20poligen Flachbandkabel über den Erweiterungsstecker hergestellt. Da beim Experimentierboard nur Port P1 als 8Bit Ein-/Ausgabeport frei ist, wird die Signalverteilung zu den Schaltern und Leuchtdioden auf der IS51-E/A1 vorgenommen. Für Ausgaben ist der Ausgabewert am Kontrollerport P1 auszugeben und anschließend das Bit INT1 zu löschen und gleich wieder zu setzen. Dadurch wird der Ausgabewert in das Ausgaberegister 74HC374 übernommen. An Port P1 muß jetzt wieder FFH ausgegeben werden, um Eingaben zu ermöglichen. Zu Beachten ist die inverse Anzeige der Leuchtdioden, bei 0 leuchten die Dioden, bei 1 sind sie dunkel. Für Eingaben ist, um den Eingabebaustein 74HC244 freizuschalten das T1-Bit zu löschen, der Wert einzulesen und anschließend das T1-Bit wieder zu setzen. Zur Bedienung der Ein- und Ausgabe mit der Platine IS51-E/A1 sind im Monitorprogramm des Experimentierboards die Unterprogramme IPLA1 (Input E/A-Platine 1) und OPLA1 (Output E/A-Platine 1) vorgesehen. Die beiden Tasten T0 und INT0 sind entprellt, sie werden in Ruhelage als 0 und gedrückt als 1 gelesen. Sie sind auf die beiden Signale T0 (P3.4) und INT0 (P3.2) des Mikrokontrollers geführt. Eine gedrückte Taste wird durch die entsprechende Kontroll-LED auf der IS51-E/A1 angezeigt.

Bild 5.2 Ausgabeschema

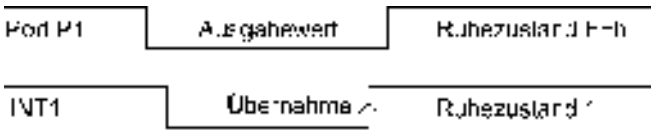
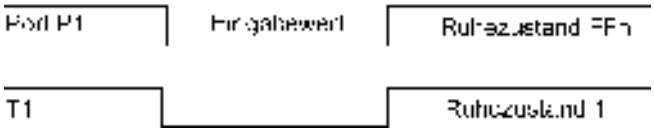


Bild 5.3 Eingabeschema



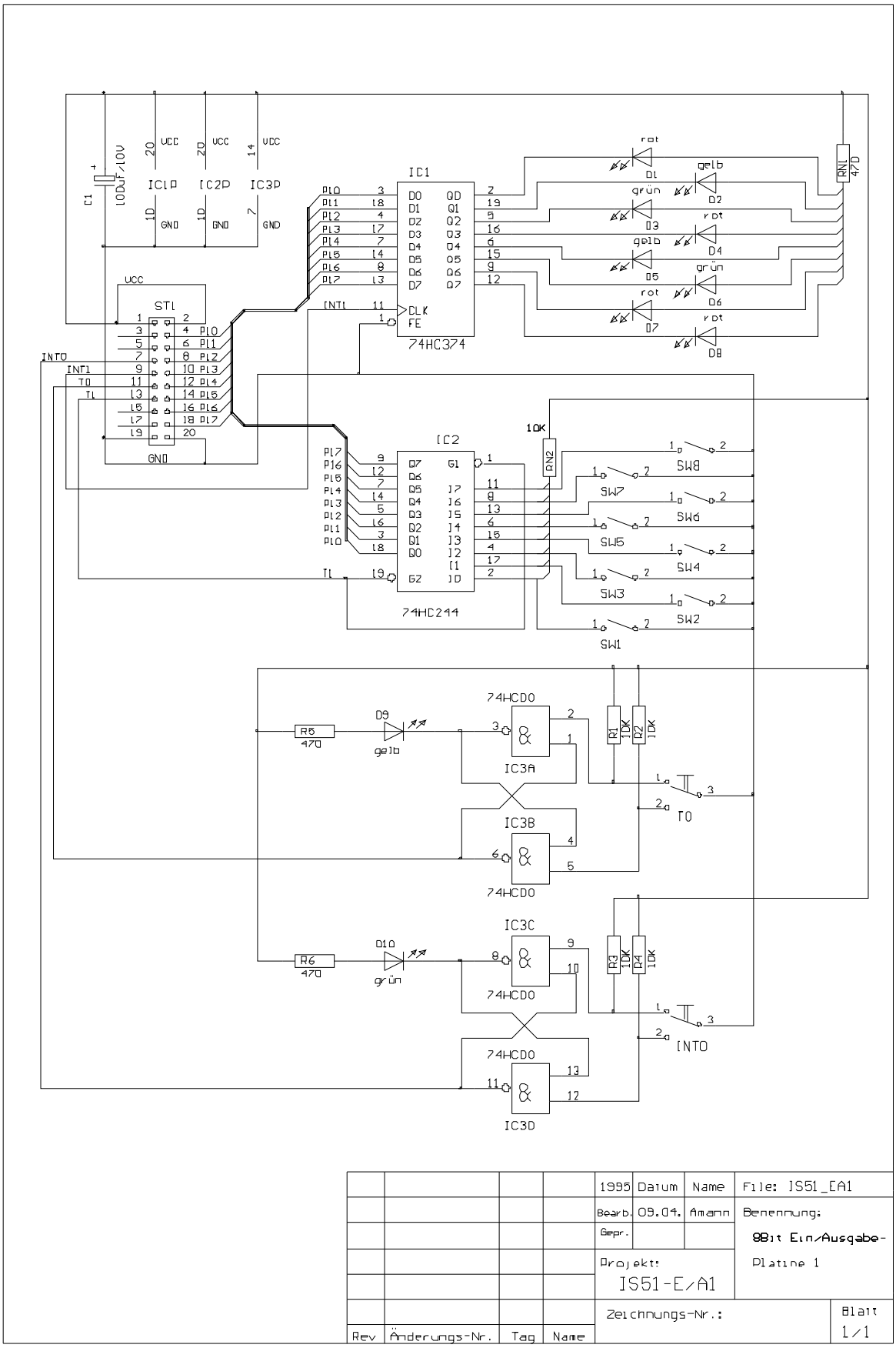
Das Unterprogramm OPLA1 gibt den Inhalt des R7 über die Leuchtdioden aus. Vor der Ausgabe wird der Ausgabewert invertiert, so daß eine 1 zu einer leuchtenden LED führt.

```
0F36h  OPLA1:
                PUSH  ACC                ;Akku retten
                MOV  A,R7                ;Ausgabewert umladen
                MOV  DPTR,#EOUTPORT1     ;Zeiger auf Rettungszelle
                MOVX  @DPTR,A            ;Ausgabewert retten
                CPL  A                   ;Invertieren für 74HC374
                MOV  P1,A                ;Wert anlegen
                CLR  INT1                ;Übernahmeimpuls
                SETB  INT1               ;erzeugen
                MOV  P1,#0FFh            ;Port 1 Ruhelage
                POP  ACC                 ;Akku wiederherstellen
                RET                      ;Zurück
```

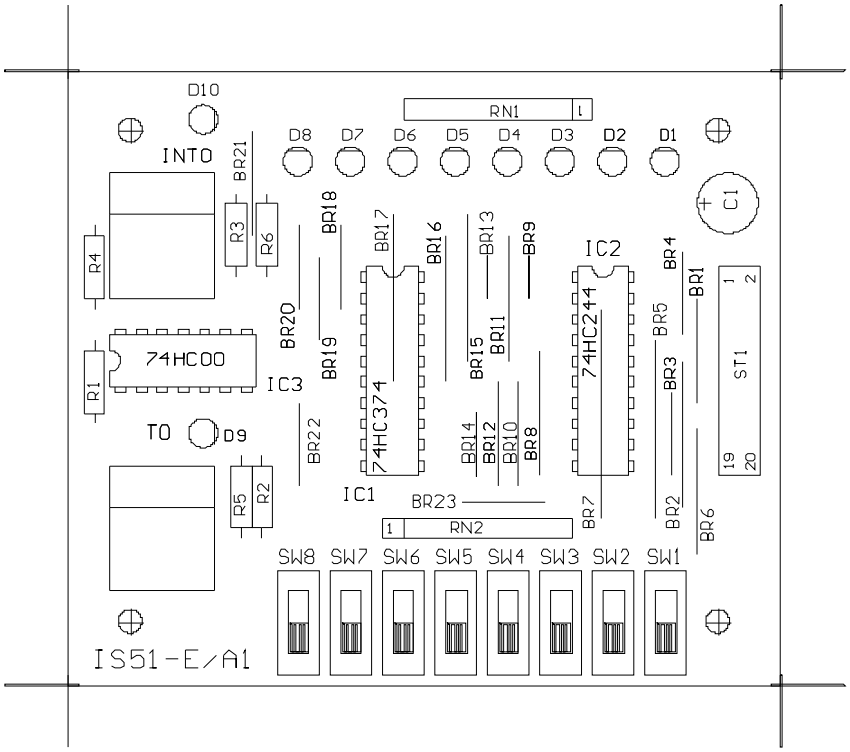
Das Unterprogramm IPLA1 liest die Schalterstellung der IS51-E/A1 in R7 ein.

```
0F52h  IPLA1:
                CLR  T1                  ;Eingabebaustein freigeben
                MOV  R7,P1               ;Wert einlesen
                SETB  T1                 ;Eingabebaustein sperren
                RET                      ;Zurück
```

5.1 Stromlaufplan IS51-E/A1

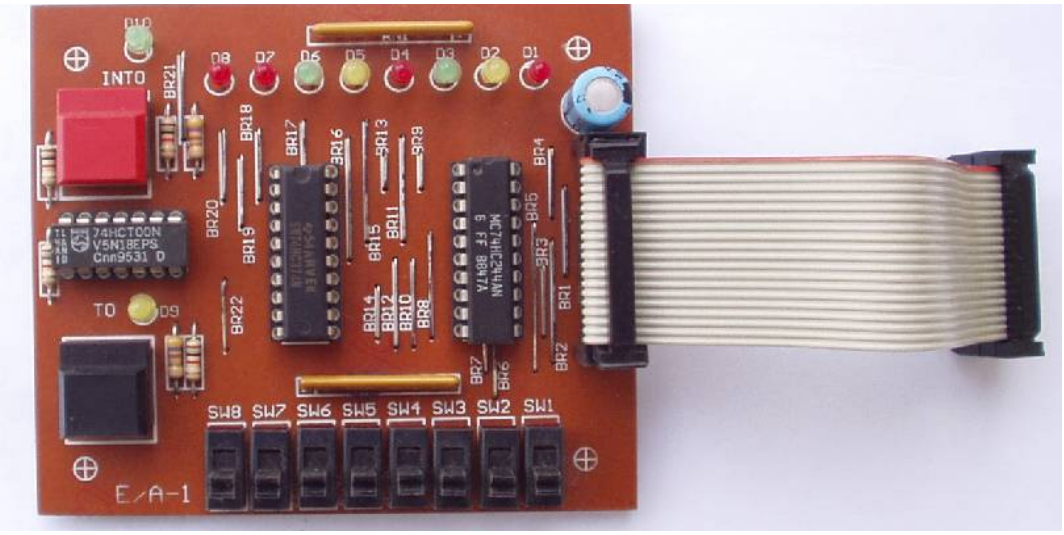


5.2 Bestückungsplan IS51-E/A1



5.3 Technische Daten

Versorgungsspannung:	5V
Stromaufnahme:	Max. 60mA
Ausgabe:	8 LED´s
Eingabe:	8 Schalter, 2 entprellte Taster
Sonstiges:	2 Kontroll-LED´s für Taster



6 Programmbeispiele - Assembler

6.1 Programmbeispiele (Unterprogramme mit Adressangabe

6.1.1 Lauflicht

Das Programm steuert ein Lauflicht für den Ausgabeport der Ein/Ausgabeplatine IS51-E/A1. Zur Zeitverzögerung wird das systemeigene Unterprogramm ZSECH (Zehntel SECunden Hexadezimal) verwendet und mit einer Zeitverzögerung von 2 Zehntelsekunden programmiert. Das Programm kann durch drücken der T0-Taste beendet werden. Bei Beendigung wird die Autostartkennung des Programms durch das Unterprogramm CLRAUTO entfernt, wodurch es nach Reset nicht wieder gestartet wird. Das Programm als solches bleibt im Speicher bis es überschrieben oder gelöscht wird. Es könnte also auch nach dem Löschen der Autostart-Kennung durch die Funktion 14h "Programm starten" des Treibers (IS51.COM) noch ausgeführt werden. Es werden keine Interrupts benutzt, weshalb der dafür reservierte Speicher unbeschrieben bleiben kann.

Zu Beginn werden dem Assembler die Adressen der benutzten Unterprogramme durch EQU-Anweisungen mitgeteilt. Die Anweisung ORG 8000h teilt dem Assembler mit welcher Adresse die Übersetzung beginnt. Der Befehl LJMP START springt zum eigentlichen Anfang des Programms und stellt gleichzeitig die Autostart-Kennung dar. Die Anweisung ORG 8100h veranlasst den Assembler die folgenden Befehle ab dieser Adresse zu übersetzen (siehe Speicherbelegung).

Das Programm wird mit einem geeigneten Assembler übersetzt und über serielle Schnittstelle in den Arbeitsspeicher der IS51 geladen. Durch die Autostart-Kennung wird es nach erfolgreicher Übertragung automatisch gestartet. Zu Beachten ist, daß ein eventuell im Speicher vorhandenes Autostart-Programm vor der Übertragung des neuen Programms gestopt werden muß.

```
0001  0000                        ;*  Lauflicht   *
0002  0000
0003  0000                        $p  8051  $s
0004  0000
0005  0000                        ZSECH          EQU    0E95h  ;Deklaration
0006  0000                        OPLA1          EQU    0F36h  ;der  Unterprogramm-
0007  0000                        CLRAUTO        EQU    135Fh  ;Adressen
0008  0000
0009  0000                        org  8000h
0010  8000           02  8100    ljmp  START           ;Autostart-  Kennung
0011  8003
0012  8003                        ;*  keine  Interrupts
0013  8003
0014  8003                        org  8100h            ;Anfangsadr.  Anwenderprogr.
0015  8100                        START:
0016  8100           74  01       mov  a,#01h           ;Startwert
0017  8102                        SCHLEIFE:
0018  8102           FF           mov  r7,a
0019  8103           12  0F36     lcall  OPLA1          ;Wert  anzeigen
0020  8106           7F  02       mov  r7,#2            ;Zeitwert  0,2  s.
0021  8108           12  0E95     lcall  ZSECH          ;UP-Wartezeit
0022  810B           23           rl  a                 ;Neuer  Anzeigewert
0023  810C           30  B4  F3   jnb  T0,SCHLEIFE     ;Wiederholen  bis  T0=1
0024  8113           12  135F     lcall  CLRAUTO        ;Autostart-  Kennung  löschen
0025  8116           02  0000     ljmp  0               ;Neustart  Monitorprogramm
0026  8119
0027  8119
0028  8119                        END                   ;Ende
```

6.1.2 Schalter lesen

Die Schalterstellung der Erweiterungsplatine IS51-E/A1 werden mit Hilfe systemeigener Unterprogramme gelesen und zu den Leuchtdioden wieder ausgegeben. Das Programm wird durch drücken der T0-Taste beendet, wobei auch die Autostart-Kennung entfernt wird.

```
0001  0000                    ;*  Schalter  E/A-1 lesen  *
0002  0000                    $p  8051
0003  0000                    OPLA1          EQU    0F36h
0004  0000                    IPLA1          EQU    0F52h
0005  0000                    CLRAUTO        EQU    135Fh
0006  0000
0007  0000                    org  8000h
0008  8000          02  8100  ljmp  START
0009  8003
0010  8003                    org  8100h
0011  8100                    START:
0012  8100          12  0F52  lcall  IPLA1          ;Schalterstellung  lesen
0013  8103          12  0F36  lcall  OPLA1          ;Zum  Ausgabeport  schreiben
0014  8106          30  B4  F7  jnb  T0,START      ;Wiederholen  bis  T0=1
0015  8109          7F  00    mov  r7,#0
0016  810B          12  20C5  lcall  OPLA1          ;Zuletzt  00h  ausgeben
0017  810E          12  135F  lcall  CLRAUTO        ;Autostart-  Kennung  löschen
0018  8111          02  0000  ljmp  0              ;Neustart
0019  8114
0020  8114                    END
```

6.1.3 Bit abfragen

Wird die INT0-Taste gedrückt, leuchten nur die grünen LED´s, sonst leuchten nur die gelben LED´s. Das Programm wird durch die T0-Taste beendet.

```
0001  0000                    ;*  INT0-Bit  abfragen  *
0003  0000                    $p  8051
0005  0000                    OPLA1          EQU    0F36h
0006  0000                    CLRAUTO        EQU    135Fh
0007  0000
0008  0000                    org  8000h
0009  8000          02  8100  ljmp  START
0012  8003
0013  8003                    org  8100h
0014  8100                    START:
0015  8100          74  12    mov  a,#12h          ;Ausgabewert  gelbe  LED's
0016  8102          30  B2  01  jnb  INT0,AUSGABE  ;Sprung  falls  INT0  =  0
0017  8105          23        rl  a                ;Nur  grüne  LED's
0018  8106                    AUSGABE:
0019  8106          FF        mov  r7,a
0020  8107          12  0F36  lcall  OPLA1          ;Zum  Ausgabeport  schreiben
0021  810A          30  B4  F3  jnb  T0,START      ;Wiederholen  bis  T0=1
0022  810D          12  135F  lcall  CLRAUTO        ;Autostart-  Kennung  löschen
0023  8110          02  0000  ljmp  0              ;Neustart
0024  8113
0025  8113                    END
```

6.1.4 Interrupt

Der INT0-Interrupt wird freigegeben, auf fallende Flanke programmiert und es wird ihm eine hohe Priorität zugewiesen. Der Interrupt-Vektor wird in Adresse 8003h eingerichtet und zeigt auf die Interrupt-Service-Routine. Das Hauptprogramm steuert ein Lauflicht der Erweiterungsplatine IS51-E/A1, das beim Loslassen der INT0-Taste (fallende Flanke) das Hauptprogramm unterbricht und für eine halbe Sekunde nur die roten LED´s einschaltet. Anschließend wird das Hauptprogramm fortgesetzt.

```
0001  0000                    ;*  INT0-Interrupt   *
0002  0000                    $p  8051
0003  0000
0004  0000                    OPLA1          EQU    0F36h
0005  0000                    CLRAUTO        EQU    135Fh
0006  0000                    ZSECH          EQU    0E95h
0007  0000
0008  0000                    org  8000h
0009  8000          02  8100  ljmp  START
0010  8003          02  8120  ljmp  INT0ISR
0011  8006
0012  8006                    org  8100h
0013  8100                    START:
0014  8100          D2  B8    setb  PX0              ;Hohe  Priorität
0015  8102          D2  88    setb  IT0              ;INT0  Flankengesteuert
0016  8104          C2  89    clr  IE0               ;Alte  Anforderung  löschen
0017  8106          75  A8  81  mov  IE,#81h          ;Nur  INT0  freigeben
0018  8109          74  01    mov  a,#1              ;1.  Ausgabewert
0019  810B                    SCHLEIFE:
0020  081B          FF        mov  r7,a              ;Ausgabe  vorbereiten
0021  810C          12  0F36  lcall  OPLA1          ;Ausgabe
0022  810F          7F  05    mov  r7,#5
0023  8111          12  0E95  lcall  ZSECH
0024  8114          23        rl  a                  ;Neuer  Ausgabewert
0025  8115          30  B4  F4  jnb  T0,SCHLEIFE    ;Wiederholen  bis  T0=1
0026  8118          12  135F  lcall  CLRAUTO        ;Autostart-Kennung  löschen
0027  811B          75  A8  00  mov  IE,#0
0028  811E          02  0000  ljmp  0                ;Neustart
0029  8121
0030  8121                    INT0ISR:               ;INT0-Service-Routine
0031  8121          C0  07    push  07               ;Ausgabewert  retten
0032  8123          7F  C9    mov  R7,#0C9h          ;Alle  roten  LED's
0033  8125          12  0F36  lcall  OPLA1          ;Einschalten
0034  8128          7F  05    mov  r7,#05            ;Für  0,5  Sekunden
0035  812A          12  0E95  lcall  ZSECH
0036  812D          D0  07    pop  07                ;Ausgabewert  restaurieren
0037  812F          12  0F36  lcall  OPLA1
0038  8132          32        reti                   ;Zurück  zum  Hauptprogramm
0039  8133
0040  8133                    END                    ;Ende
```

6.3 Programmbeispiele (Treiber IS51.COM)

Vor Ausführung der Programme für den Treiber-Interrupt muß dieser zwingend installiert sein (siehe Abschnitt 3.8). Die Programme werden für den PC geschrieben (8086 Maschinencode) und auf diesem ausgeführt. Sie steuern die Hardware der IS51 über die serielle Schnittstelle.

6.3.1 Lauflicht

Das Programm steuert ein Lauflicht für den Ausgabeport der Ein/Ausgabeplatine IS51-E/A1. Es benutzt die Funktionsaufrufe des Treiber-Interrupt (Funktion 22H, Port Byte ausgeben). Das Programm läuft auf dem PC und bedient die Funktionen der IS51 über die serielle Schnittstelle. Zur Zeitverzögerung wird eine BIOS-Routine verwendet, die nur auf AT-Computern verfügbar ist. Das Programm kann durch die ESC-Taste beendet werden.

Der Funktion 22H muß in AH die Funktionsnummer, in AL 00H, in BL der Ausgabewert und in DH die Portnummer übergeben werden. Durch den Befehl INT 60H wird die Funktion ausgeführt.

```
0001  0000                    ;* Lauflicht   *
0002  0000                    $p 8086 $b
0003  0000
0004  0000                    org 100h
0005  0100                    START:
0006  0100          B3 01      mov bl,01h          ;Startwert für Ausgabe
0007  0102                    SCHLEIFE:
0008  0102          B8 0022    mov ax,2200h        ;Fnr. Byte ausgeben
0009  0105          B6 00      mov dh,0            ;Port = E/A-1 Ausgabe
0010  0107          CD 60      int 60h             ;Funktion ausführen
0011  0109          D0 C3      rol bl,1            ;Neuer Ausgabewert
0012  010B          B4 86      mov ah,86h          ;BIOS - Wartezeit
0013  010D          B9 0500    mov cx,5            ;5 x 0,1 Sekunden = 0,5s.
0014  0110          BA 0000    mov dx,0            ;(Nur bei AT möglich)
0015  0113          CD 15      int 15h
0016  0115          B4 06      mov ah,6            ;DOS - Taste lesen
0017  0117          B2 FF      mov dl,0FFh
0018  0119          CD 21      int 21h
0019  011B          3C 1B      cmp al,1Bh          ;ESC - Taste ?
0020  011D          75 E3      jnz SCHLEIFE        ;Falls nein, wiederholen
0021  011F          B8 004C    mov ax,4C00h        ;Falls ja, Ende
0022  0122          CD 21      int 21h
0023  0124
0024  0124                    END
```

6.3.2 Schalter lesen

Die Schalterstellung der Erweiterungsplatine E/A-1 wird gelesen und zu den LED´s der gleichen Platine ausgegeben. Das Programm kann durch die ESC-Taste beendet werden.

```
0001  0000                    ;* Ports lesen/schreiben *
0002  0000                    $p 8086 $b
0003  0000
0004  0000                    org 100h
0005  0100                    START:
0006  0100  B8 0021            mov ax,2100h        ;Fnr. Port Byte lesen
0007  0103  B6 01              mov dh,01           ;Port = E/A-1 Eingabe
0008  0105  CD 60              int 60h             ;Funktion ausführen
0009  0107  B8 0022            mov ax,2200h        ;Fnr. Port Byte schreiben
0010  010A  B6 00              mov dh,0            ;Port = E/A-1 Ausgabe
0011  010C  CD 60              int 60h             ;Funktion ausführen
0012  010E  B4 06              mov ah,6            ;DOS - Taste lesen
0013  0110  B2 FF              mov dl,0FFh
0014  0112  CD 21              int 21h
0015  0114  3C 1B              cmp al,1Bh          ;ESC - Taste ?
0016  0116  75 E8              jnz START           ;Falls nein, wiederholen
0017  0118  B8 004C            mov ax,4C00h        ;Falls ja, Ende
0018  011B  CD 21              int 21h
0019  011D
0020  011D                    END
```

6.3.3 Programm starten

Das Programm ist geeignet ein Anwenderprogramm im Arbeitsspeicher der IS51, dessen Autostartkennung gelöscht wurde, zu starten. Es startet das Anwenderprogamm mit Hilfe der Funktion 14H des Treiber-Interrupt bei Adresse 8100H (Zeile 7: MOV BX,Startadresse)

```
0001  0000                    ;* Programm in IS51 starten (Adr. 8100) *
0002  0000                    $p 8086 $b
0003  0000
0004  0000                    org 100h
0005  0100                    START:
0006  0100  B8 0014            mov ax,1400h            ;Fnr. Programm starten
0007  0103  BB 0081            mov bx,8100h            ;Startadresse = 8100h
0008  0106  CD 60              int 60h                 ;Funktion ausführen
0009  0108  B8 004C            mov ax,4C00h            ;Beenden
0010  010B  CD 21              int 21h
0011  010D
0012  010D                    END
```

6.3.4 T0-Bit steuert PC-Hupe

Das T0-Bit der IS51 steuert die Systemhupe des PC. Wird das T0-Bit der IS51 gesetzt (T0-Taste gedrückt) wird die Hupe des PC eingeschaltet. Beim Loslassen der T0-Taste (T0-Bit = 0) wird die Hupe des PC wieder ausgeschaltet. Das Programm kann durch die ESC-Taste beendet werden.

```
0001  0000                    ;* T0-Bit steuert PC-Hupe *
0002  0000                    $p 8086 $b
0004  0000                    org 100h
0005  0100                    START:
0006  0100  B8 0021            mov ax,2100h      ;Fnr. Port Byte lesen
0007  0103  B6 03              mov dh,03         ;Port = Port 3
0008  0105  CD 60              int 60h           ;Funktion ausführen
0009  0107  E4 61              in al,61h         ;Hupenport lesen
0010  0109  24 FC              and al,0FCh       ;Hupe aus vorbereiten
0011  010B  F6 C3 10           test bl,10h       ;T0-Bit = 0 ?
0012  010E  74 02              jz HUPE           ;Sprung falls ja
0013  0110  0C 03              or al,03h         ;Hupe ein vorbereiten
0014  0112                    HUPE:
0015  0112  E6 61              out 61h,al        ;Hupe ändern
0016  0114  B4 06              mov ah,6          ;DOS - Taste lesen
0017  0116  B2 FF              mov dl,0FFh
0018  0118  CD 21              int 21h
0019  011A  3C 1B              cmp al,1Bh        ;ESC - Taste ?
0020  011C  75 E2              jnz START         ;Falls nein, wiederholen
0021  011E  B8 004C            mov ax,4C00h      ;Falls ja, Ende
0022  0121  CD 21              int 21h
0023  0123
0024  0123                    END
```

6.4 Programmbeispiel (DLL-Funktionen)

6.4.1 Lauflicht (Windows-Konsolenanwendung)

Das Programm steuert ein Lauflicht für den Ausgabeport der Erweiterungsplatine E/A-1. Es benutzt die Funktionen der „IS51DLL.LIB“ (EA1OUT, EA1 Byte ausgeben). Das Programm läuft auf dem PC und bedient die IS51 über die serielle Schnittstelle. Zur Zeitverzögerung wird eine Windows-Funktion verwendet. Das Programm kann durch eine beliebige Taste beendet werden.

Im Projekt befinden sich die Dateien: IS51DLL.LIB und IS51LL.CPP

Im Verzeichnis des Projekts befindet sich die Datei: IS51DLL.DLL

Das Programm ist eine Konsolenanwendung, der Quelltext und die ausführbare „IS51LL.EXE“ befindet sich auf der CD im Verzeichnis Windows\Lauflicht.

Programm: IS51LL.CPP

```
/************************************************
 * Lauflicht mit Funktionen der IS51DLL.LIB *
 * PC-steuert Funktionen der IS51              *
 ************************************************/
#include <conio.h>              // wegen "kbhit"
#include <windows.h>            // wegen Timer
#include <stdio.h>              // wegen printf
#include "IS51DLL.H"            // Header für Funktionsbibliothek

/***************************************
 * Wartezeit in Schritten zu je 10µs *
 ***************************************/
void Timer (unsigned int iTime)
{ LONGLONG counter;
  LARGE_INTEGER x;
  unsigned int uiTMul;
  QueryPerformanceFrequency(&x);
  uiTMul = x.QuadPart / 95000;
  QueryPerformanceCounter(&x);
  counter = x.QuadPart;
  counter += iTime * uiTMul;
  do {
     QueryPerformanceCounter(&x);
  }while(counter > x.QuadPart);
}

//---------------------------------------------------------------------------
void main(void)
{ WORD wVar;                    // Ausgabevariable
  InitSio(1);                   // COM1: als Schnittstelle zur IS51 einrichten.
  printf("Lauflicht in Betrieb.\n\n");
  printf("Abbruch mit beliebiger Taste!");
  wVar = 1;                     // 1. Ausgabewert
  do {
   EA1OUT(wVar);                // Zu den LED's ausgeben
   wVar <<= 1;                  // links schieben
   if(wVar > 0xFF)              // Schieben zum rotieren wandeln
    wVar = 1;
   Timer(30000);                // 300ms warten
  }while(!kbhit());             // wiederholen bis Taste
  CloseSio();                   // COM restaurieren
}
```

7 IS51 und C

Um Programme für die IS51 mit einem C51-Compiler erstellen zu können, sind einige Vorkehrungen zu treffen. Sollen die im EPROM liegenden Funktionen benutzt werden, muß die Header-Datei "IS51C.H" mit einer INCLUDE Anweisung ins Quellprogramm aufzunehmen. Alle IS51 Programme sind im Speichermodell LARGE zu übersetzen.

Beim Linken der Programm-Module ist die Startup-Objektdateien "STUPIS51C.OBJ" mit einzubinden. Die Startup-Datei trifft alle nötigen Einstellungen um die richtige Speichereinteilung für das Programm und die EPROM-Funktionen zu gewährleisten.

STUPIS51C.OBJ	Startup für C-Programme

Beim Linken ist für den Codespeicher die Adresse 8100h anzugeben (co(8100h)).

Linkeraufruf: L51 Programm-Module, STUPIS51C.OBJ co(8100h) weitere Parameter

Beispiel: L51 TEST.OBJ, STUPIS51C.OBJ co(8100h)

Die Module STUPIS51C.OBJ und TEST.OBJ zu einem lauffähigen IS51 Programm binden.

Die entstandene Zieldatei wird, z.B: mit OHS51, in eine INTEL-HEX Datei gewandelt, welche dann über eine serielle Schnittstelle zur IS51 übertragen wird. Die serielle Schnittstelle muß auf 4800 Baud, ohne Paritätsprüfung, 8 Datenbit und 1 Stopbit eingestellt sein.

Beispiel: OHS51 TEST

mode com1:48,n,8,1

copy TEST.HEX com1:

Typischer Programmaufbau:

```
/* C-Programm für IS51 */

#pragma ROM(LARGE)

#include <reg51.h>
#include <is51c.h>

main()
{

   Hier befindet sich das Programm

 _CLRAUTO();        /* Autostart- Kennung löschen */
 _START_OS();       /* Betriebssystem der IS51 neu starten */
}
```

7.1 Funktionen im Eprom der IS51

Im Eprom der IS51 befinden sich Unterprogramme, die für den Aufruf als C-51 Funktionen geeignet sind. Sollen diese Funktionen benuzt werden, ist die Header-Datei "IS51C.H" in das Programm einzubinden (#include <IS51C.H>). Es können fast alle der Unterprogramme auch als C-Funktion zur verwendet werden. Für alle Funktionen die sich im EPROM befinden werden die Funktionsnamen in Großbuchstaben angegeben. Die Funktion „DIV32“ ist unter C nicht verwendbar, dafür besitzt C eigene Arithmetikfunktionen.

7.1.1 Ausgabefunktionen

Name: **_OPLA1** (Output Erweiterungsplatine 1)

Einen char-Wert (8Bit) zum Ausgabeport (LED´s) der Ein/Ausgabe-Platine E/A-1 ausgeben.

Definition: void _OPLA1(unsigned char)

Beispiel:
```
unsigned char ucVar;        /* Ausgabevariable deklarieren */
ucVar = 0xC9;               /* Ausgabewert festlegen */
_OPLA1(ucVar);              /* Wert ausgeben */
```

Name: **_OPLA1BS** (Out Platine 1 Bit setzen)

Ein oder mehrere Bit am Ausgabeport der Ein/Ausgabe-Platine E/A-1 setzen. Aufrufparameter ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit.

Definition: void _OPLA1BS(unsigned char)

Beispiel:
```
unsigned char ucVar;
ucVar = 0x08;               /* Bitmaske Bit 3 */
_OPLA1BS(ucVar);            /* Bit setzen */
```

Name: **_OPLA1BR** (Out Platine 1 Bit rücksetzen)

Ein oder mehrere Bit am Ausgabeport der Ein/Ausgabe-Platine E/A-1 rücksetzen. Aufrufparameter ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit.

Definition: void _OPLA1BR(unsigned char)

Beispiel:
```
unsigned char ucVar;
ucVar = 0x18;               /* Bitmaske Bit 3 und 4 */
_OPLA1BR(ucVar);            /* Bit rücksetzen */
```

Version 1.20

Name: **_OPLA1BC** (Out Platine 1 Bit Complement)

Ein oder mehrere Bit am Ausgabeport der Ein/Ausgabe-Platine E/A-1 invertieren. Aufrufparameter ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit.

Definition: void _OPLA1BC(unsigned char)

Beispiel:
```
unsigned char ucVar;
ucVar = 0x51;          /* Bitmaske Bit 6, 4 und 1 */
_OPLA1BC(ucVar);       /* Bit setzen */
```

Name: **_SEROA** (Serieller Outport Ausgabe)

Einen unsigned char-Wert (8Bit) zum ersten seriellen Erweiterungsport ausgeben. Daten = P1.0, Load = P1.1, Takt = P1.2.

Definition: void _SEROA(unsigned char)

Beispiel:
```
unsigned char ucVar;   /* Ausgabevariable deklarieren */
ucVar = 0x5A;          /* Ausgabewert festlegen */
_SEROA(ucVar);         /* Wert ausgeben */
```

Name: **_SERAUSG** (Serielle Ausgabeerweiterung)

Serielle Ausgabeerweiterung bis max. 8 Ports bedienen. Beginnend mit dem Inhalt der Rettungszelle EOUTPORT1 (XRAM 8030h) werden, der Reihe nach, die Inhalte der Rettungszellen EOUTPORT1 - 8 zu der angegebenen Anzahl Erweiterungsports ausgegeben. Übergabeparameter ist die Anzahl der anzusprechenden Erweiterungsports (1-8). Daten = P1.0, Load = P1.1, Takt = P1.2.

Definition: void _SERAUSG(unsigned char)

Beispiel:
```
EOUTPORT1 = 0x5A;      /* 1.Ausgabewert festlegen */
EOUTPORT2 = 0xA5;      /* 2.Ausgabewert festlegen */
_SERAUSG(2);           /* 2 Werte ausgeben */
```

Name: **_SERPBS** (Seriell Port Bit setzen)

Ein oder mehrere Bit in einer Ausgabe-Rettungszelle (EOUTPORT1-8) setzen. Aufrufparameter1 ist die Portnummer (1-8) Aufrufparameter 2 ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit. Zur Ausgabe der Inhalte der Rettungszellen kann Funktion SERAUSG() benutzt werden.

Definition: void _SERPBS(unsigned char, unsigned char)

Beispiel:
```
unsigned char ucPortnr, ucBitmaske;
ucPortnr = 1;                       /* EOUTPORT1 für Erweiterungsport 1 */
ucBitmaske = 0x08;                  /* Bitmaske Bit 3 */
_SERPBS(ucPortnr, ucBitmaske);      /* Bit in Rettungszelle setzen */
_SERAUSG(1);                        /* <EOUTPORT1> ausgeben */
```

Name: **_SERPBR** (Seriell Port Bit rücksetzen)

Ein oder mehrere Bit in einer Ausgabe-Rettungszelle (EOUTPORT1-8) rücksetzen. Aufrufparameter1 ist die Portnummer (1-8) Aufrufparameter 2 ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit. Zur Ausgabe der Inhalte der Rettungszellen kann Funktion SERAUSG() benutzt werden.

Definition: void _SERBR(unsigned char, unsigned char)

Beispiel:
```
unsigned char ucPortnr, ucBitmaske;
ucPortnr = 3;                    /* EOUTPORT3 für Erweiterungsport 3 */
ucBitmaske = 0x81;               /* Bitmaske Bit 7 und 1 */
_SERPBR(ucPortnr, ucBitmaske);   /* Bit in Rettungszelle rücksetzen */
_SERAUSG(3);                     /* <EOUTPORT1,2,3> ausgeben */
```

Name: **_SERPBC** (Seriell Port Bit complement)

Ein oder mehrere Bit in einer Ausgabe-Rettungszelle (EOUTPORT1-8) invertieren. Aufrufparameter1 ist die Portnummer (1-8) Aufrufparameter 2 ist die Bitmaske (Kap. 3.9.1(2) S.45) der zu ändernden Bit. Zur Ausgabe der Inhalte der Rettungszellen kann Funktion SERAUSG() benutzt werden.

Definition: void _SERBC(unsigned char, unsigned char)

Beispiel:
```
unsigned char ucPortnr, ucBitmaske;
ucPortnr = 4;                    /* EOUTPORT4 für Erweiterungsport 4 */
ucBitmaske = 0x60;               /* Bitmaske Bit 6 und 5 */
_SERPBC(ucPortnr, ucBitmaske);   /* Bit in Rettungszelle invertieren */
_SERAUSG(4);                     /* <EOUTPORT1,2,3,4> ausgeben */
```

7.1.2 Eingabefunktionen

Name: **_IPLA1** (Input Erweiterungsplatine 1)

Einen char-Wert (8Bit) vom Eingabeport (Schalter) der Ein/Ausgabe-Platine E/A-1 einlesen.

Definition: unsigned char _IPLA1(void)

Beispiel:
```
unsigned char ucVar;    /* Eingabevariable deklarieren */
ucVar = _IPLA1();       /* Wert nach ucVar einlesen */
```

Name: **_IPLA1BIT** (Input Erweiterungsplatine 1 Bit lesen)

Ein Einzelbit vom Eingabeport (Schalter) der Ein/Ausgabe-Platine E/A-1 einlesen. Aufrufparameter ist die Bitnummer (0-7).

Definition: bit _IPLA1BIT(unsigned char)

Beispiel:
```
bit btBit;              /* Bitvariable deklarieren */
btBit = _IPLA1BIT(5);   /* Bit5 nach bVar einlesen */
```

Name: **_SERIA** (Seriell Inport A)

Einen char-Wert (8Bit) vom ersten seriellen Erweiterungsport einlesen. Takt = P1.3, Load = P1.4, Daten = P1.5. Der Wert wird auch in der Rettungszelle EINPORT1 gespeichert.

Definition: unsigned char _SERIA(void)

Beispiel:
```
unsigned char ucVar;        /* Eingabevariable deklarieren */
ucVar = _SERIA();           /* Wert nach ucVar und EINPORT1 einlesen */
```

Name: **_SEREING** (Serielle Eingabeports)

Serielle Eingabeerweiterung bis max. 8 Ports bedienen. Beginnend mit dem Wert am Erweiterungsport1 werden, der Reihe nach, die Werte der seriellen Erweiterungsports in die Rettungszellen EINPORT1 - 8 (XRAM 8038h-803Fh) eingelesen. Übergabeparameter ist die Anzahl der angeschlossenen Erweiterungsports (1-4). Takt = P1.3, Load = P1.4, Daten = P1.5.

Definition: void _SEREING(unsigned char)

Beispiel:
```
_SEREING(2);        /* 2 Ports nach EINPORT1 und 2 einlesen */
```

Name: **_EWIBIT** (Erweiterungsport Input Bit lesen)

Ein Einzelbit aus einer Erweiterungsport-Rettungszelle (EINPORT1-8) lesen. Aufrufparameter1 ist die Portnummer (1-8), Aufrufparameter 2 ist die Nummer (0-7) des gewünschten Bit. Zum Einlesen der Werte in die Rettungszellen kann die Funktion _SEREING() benutzt werden.

Definition: bit _EWIBIT(unsigned char, unsigned char)

Beispiel:
```
bit btBit;                  /* Bitvariable deklarieren */
_SEREING(4);                /* EPORT1,2,3,4 in die Rettungszellen lesen */
btBit = _EWIBIT(4,3);       /* EINPORT4-Bit3 nach btBit einlesen */
```

Name: **_SIOABLOCK** (Datenblock über SIO senden, Anzahl vorgeben)

Einen Block Bytes aus dem Programm/Datenspeicher der IS51über die serielle Schnittstelle senden. Die Speicheradresse des Blocks wird im WORD-Parameter erwartet, die Anzahl der Bytes im Byte-Parameter. Von 8000h-FFFFh sind Programm- und Datenspeicher identisch.

Definition: void _SIOABLOCK(WORD,BYTE)

Beispiel:
```
_SIOABLOCK(0x8200, 10);      /* 10 Bytes ab 8200h über SIO senden */
```

Name: **_SIODBLOCK** (Datenblock über SIO senden, Endzeichen vorgeben)

Einen Block Bytes aus dem Programm/Datenspeicher der IS51über die serielle Schnittstelle senden. Die Speicheradresse des Blocks wird im WORD-Parameter erwartet, die Übertragung endet vor dem $-Zeichen (es wird nicht mit übertragen). Von 8000h-FFFFh sind Programm- und Datenspeicher identisch.

Definition: void _SIODBLOCK(WORD)

Beispiel:
```
_SIODBLOCK(0x8200) ;        /* Bytes ab 8200h über SIO senden, bis $-Zeichen */
```

Name: **_CHRDTO1** (Ein Zeichen von SIO lesen, mit Timeout in 10ms Schritten)

Ein Zeichen von serieller Schnittstelle lesen mit Timeout. Die Timeoutzeit wird in 10ms Schritten als Parameter hexadezimal übergeben (01-FF = 0,01-2,55s., 00 = 2,56s.). Bei Timeout wird 0 zurückgegeben.

Definition: unsigned char _CHRDTO1(unsigned char)

Beispiel: unsigned char bVar;

bVar = _CHRDTO1(50); /* Zeichen lesen mit 500ms Timeout */

Name: **_CHRDTO2** (Ein Zeichen von SIO lesen, mit Timeout in 100ms Schritten)

Ein Zeichen von serieller Schnittstelle lesen mit Timeout. Die Timeoutzeit wird in 100ms Schritten als Parameter hexadezimal übergeben (01-FF = 0,1-25,5s., 00 = 25,6s.). Bei Timeout wird 0 zurückgegeben.

Definition: unsigned char _CHRDTO2(unsigned char)

Beispiel: unsigned char bVar;

bVar = _CHRDTO2(10); /* Zeichen lesen mit 1s. Timeout */

Name: **_BYTERD** (Zwei ASCII-Zeichen von SIO lesen und als Hexbyte zurückgeben)

Zwei ASCII-Zeichen von serieller Schnittstelle lesen und zum Hexbyte gewandelt zurückgeben. Aus den ASCII-Zeichen 32h(2), 43h(C) wird das Hexbyte 2Ch. Kein Timeout, es wird gewartet bis zwei Zeichen empfangen sind.

Definition: unsigned char _BYTERD(void)

Beispiel: unsigned char bVar;

bVar = _BYTERD(); /* 2 ASCII-Zeichen als Hexbyte lesen */

7.1.3 Zeitfunktionen

Name: **_MINH** (Minuten Hex)

Unterbrechung des laufenden Programms für die übergebene Anzahl an Minuten. Aufrufparameter ist die Wartezeit als char Wert (8Bit). Zeiten: 1-255 = 1-255 Min, 0 = 256 Min. Die Zeiten für den Funktionsaufruf werden nicht berücksichtigt. Die Funktion arbeitet mit Programmschleifen, weshalb kein Hardware-Timer verändert wird.

Definition: void _MINH(unsigned char)

Beispiel: _MINH(5); /* 5 Minuten warten */

Version 1.20

Name: **_SECH** (Sekunden Hex)

Unterbrechung des laufenden Programms für die übergebene Anzahl an Sekunden. Aufrufparameter ist die Wartezeit als unsigned char Wert (8Bit). Zeiten: 1-255 = 1-255 sec, 0 = 256 sec. Die Zeiten für den Funktionsaufruf werden nicht berücksichtigt. Die Funktion arbeitet mit Programmschleifen, weshalb kein Hardware-Timer verändert wird.

Definition: void _SECH(unsigned char)

Beispiel: _SECH(20); /* 20 Sekunden warten */

Name: **_ZSECH** (Zehntel Sekunden Hex)

Unterbrechung des laufenden Programms für die übergebene Anzahl an 1/10 Sekunden. Aufrufparameter ist die Wartezeit als unsigned char Wert (8Bit). Zeiten: 1-255 = 0,1-25,5 sec, 0 = 25,6 sec. Die Zeiten für den Funktionsaufruf werden nicht berücksichtigt. Die Funktion arbeitet mit Programmschleifen, weshalb kein Hardware-Timer verändert wird.

Definition: void _ZSECH(unsigned char)

Beispiel: _ZSECH(5); /* 0,5 Sekunden warten */

Name: **_HSECH** (Hundertstel Sekunden Hex)

Unterbrechung des laufenden Programms für die übergebene Anzahl an 1/100 Sekunden. Aufrufparameter ist die Wartezeit als unsigned char Wert (8Bit). Zeiten: 1-255 = 0,01-2,55 sec, 0 = 2,56 sec. Die Zeiten für den Funktionsaufruf werden nicht berücksichtigt. Die Funktion arbeitet mit Programmschleifen, weshalb kein Hardware-Timer verändert wird.

Definition: void _HSECH(unsigned char)

Beispiel: _HSECH(3); /* 0,03 Sekunden warten */

Name: **_MSECH** (Milli Sekunden Hex)

Unterbrechung des laufenden Programms für die übergebene Anzahl an Millisekunden (1/1000s). Aufrufparameter ist die Wartezeit als unsigned char Wert (8Bit). Zeiten: 1-255 = 1-255 msec, 0 = 256 msec. Die Zeiten für den Funktionsaufruf werden nicht berücksichtigt. Die Funktion arbeitet mit Programmschleifen, weshalb kein Hardware-Timer verändert wird.

Definition: void _MSECH(unsigned char)

Beispiel: _MSECH(3); /* 0,003 Sekunden warten */

Name: **_T0PAUSE** (Pause durch T0)

Warten bis das Signal T0 logisch „1“ und anschließend wieder „0“ wird. Das entspricht bei der Erweiterungsplatine E/A-1 dem drücken und wieder loslassen der T0-Taste.

Definition: void _T0PAUSE(void)

Beispiel: _T0PAUSE(); /* warten bis T0 = 1 und wieder 0 */

Name: **_INT0PAUSE** (Pause durch INT0)

Warten bis das Signal INT0 logisch „1“ und anschließend wieder „0“ wird. Das entspricht bei der Erweiterungsplatine E/A-1 dem drücken und wieder loslassen der INT0-Taste.

Definition: void _INT0PAUSE(void)

Beispiel: _INT0PAUSE(); /* warten bis INT0 = 1 und wieder 0 */

7.1.4 Codewandelfunktionen

Name: **_ASCHEX** (ASCII nach Hexadezimal)

Umwandeln des als Parameter übergebenen ASCII-Codes 0-9 oder A-F (30h-39h oder 41h-46h), in ein Hexadezimalbyte (00-0Fh). Ist der übergebene Parameter nicht im Bereich 0-F, wird -1 (FFh) zurückgegeben.

Definition: unsigned char _ASCHEX(unsigned char)

Beispiel:
```
unsigned char ucVar;
ucVar = _ASCHEX(´B´);          /* ASCII-B (42h) in Hexbyte 0B wandeln */
```

Name: **_HEXASC** (Hexadezimal nach ASCII)

Umwandeln des als Parameter übergebenen Hexadezimalbyte (00-FFh), in zwei ASCII-Zeichen. Die zwei ASCII-Codes werden als int-Wert zurückgegeben. High-Tetrade in High, Low-Tetrade in Low.

Definition: unsigned int _HEXASC(unsigned char)

Beispiel:
```
unsigned int uiVar;
uiVar = _HEXASC(0x5B);         /* Hexbyte 5B nach ASCII 3542 wandeln */
```

Name: **_HEXDEZ** (8Bit-Hexadezimal nach 3stellig BCD)

Die übergebene zweistellige HEX-Zahl (00-FF) in eine dreistellige BCD-Zahl (000 – 255) umwandeln. Ergebnis ist ein 16Bit-Wert mit der BCD-Zahl in den unteren 3 Tetraden, die oberste Tetrade ist 0.

Definition: unsigned int _HEXDEZ(unsigned char)

Beispiel:
```
unsigned int uiVar;
uiVar = _HEXDEZ(0x7B);         /* Hexbyte-7B nach BCD-0123  wandeln */
```

Name: **_HEX16DEZ** (16Bit-Hexadezimal nach 5stellig BCD)

Die übergebene vierstellige HEX-Zahl (0000-FFFF) in eine 5stellige BCD-Zahl (00000 – 65535) umwandeln. Ergebnis ist ein 24Bit-Wert mit der BCD-Zahl in den unteren 5 Tetraden, die oberen 3 Tetraden sind 0.

Definition: unsigned long HEX16DEZ(WORD)

Beispiel:
```
unsigned long ulVar;
ulVar = _HEXDEZ(0x3039);       /* Hexbyte-3039 nach BCD-00012345  wandeln */
```

Version 1.20

Name: **_DEZHEX** (2stellig BCD in ein Hexbyte wandeln)

Die übergebene zweistellige BCD-Zahl (00-99) in ein Hexadezimalbyte (00 – 63h) umwandeln.

Definition: unsigned char _DEZHEX(unsigned char)

Beispiel:
```
unsigned char bVar;
bVar = _DEZHEX(0x75);          /* BCD-75 nach Hexbyte-4B wandeln */
```

Name: **_DEZ16HEX** (4stellig BCD in ein Hexword wandeln)

Die übergebene vierstellige BCD-Zahl (0000-9999) in ein Hexadezimalword (0000 – 270Fh) umwandeln.

Definition: unsigned int _DEZ16HEX(unsigned int)

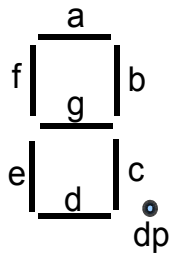
Beispiel:
```
unsigned int wVar;
wVar = _HEXDEZ(0x5678);        /* BCD-5678 nach Hexword-162Eh wandeln */
```

Name: **_BCH1TO7** (Binär Codierte Hextetrade nach 7 Segment-Code)

Umwandeln der als Parameter übergebenen Hexadezimalziffer (00-0Fh), in 7Segment-Code.

Bit0 = Seg. a, Bit1 = b, Bit2 = c, Bit3 = d, Bit4 = e, Bit5 = f, Bit6 = g, Bit7 = Dezimalpunkt

Bit	7	6	5	4	3	2	1	0
Segment	dp	g	f	e	d	c	b	a



Definition: unsigned char _BCH1TO7(unsigned char)

Beispiel:
```
unsigned int ucVar;
ucVar = _BCH1TO7(0x0E);        /* Hexbyte 0E nach 7-Segment 0111 1001 */
```

Name: **_BCH2TO7** (Binär Codierte Hexzahl nach 7 Segment-Code)

Umwandeln der als Parameter übergebenen Hexadezimalzahl (00-FFh), in zwei 7Segment-Codes. Die zwei Rückgabecodes sind in einem int-Wert(16Bit) zusammengefasst. High-Ziffer = High-Byte, Low-Ziffer = Low-Byte.

Definition: unsigned int _BCH2TO7(unsigned char)

Beispiel:
```
unsigned int uiVar;
uiVar = _BCH2TO7(0x1E);        /* Hex 1E nach 7-Segment 0000 0110  0111 1001 */
```

7.1.5 Arithmetikfunktionen

Name: **_DIV16** (16Bit Zahl durch 8Bit Zahl teilen)

Die 16Bit-Zahl des 1. Parameter durch die 8Bit-Zahl im 2. Parameter teilen, Rückgabe ist der ganzzahlige Quotient als 16Bit-Zahl.

Definition: unsigned int _DIV16(unsigned int, unsigned char)

Beispiel: unsigned int wVar;

```
wVar = _DIV16(12845, 23);          /* 12845 / 23 = 558 */
```

Name: **_MUL16** (Zwei 16Bit Zahlen multiplizieren)

Die 16Bit-Zahl des 1. Parameter mit der 16Bit-Zahl im 2. Parameter multiplizieren, Rückgabe ist das Produkt als 24Bit-Zahl.

Definition: long _MUL16(unsigned int, unsigned int)

Beispiel: long lVar;

```
lVar = _MUL16(0x322D, 0x17);       /* 322Dh * 17h = 0004820Bh */
```

Name: **_CMP16** (Zwei 16Bit Zahlen vergleichen)

Die 16Bit-Zahl A des 1. Parameter mit der 16Bit-Zahl B im 2. Parameter vergleichen, Rückgabe ist das Vergleichsergebnis als 8Bit-Zahl. A = B - 0, A > B – 1, A < B – 2.

Definition: unsigned char _CMP16(unsigned int, unsigned int)

Beispiel: unsigned char bVar;

```
bVar = _CMP16(785, 365);    /* Zahl-A (785) > Zahl-B(365) –> bVar = 1 */
```

7.1.6 Beendigungsfunktionen

Name: **_CLRAUTO** (Clear Autostart)

Entfernen der Autostart-Kennung eines IS51 Programms. Sollte beim Beenden eines IS51 Programms aufgerufen werden, um ein erneutes Starten nach dem Beenden zu verhindern.

Definition: void _CLRAUTO(void)

Beispiel:

```
_CLRAUTO();                 /* Autostart-Kennung löschen */
```

Name: **_INT0END** (INT0 Ende)

Ist der INT0-Eingang zur Aufrufzeit "1", wird das Programm beendet. Eine vorhandene Autostart-Kennung wird gelöscht.

Definition: void _INT0END(void)

Beispiel:

```
_INT0END();                 /* Programm bei INT0 = 1 beenden */
```

Name: **_INT1END** (INT1 Ende)

Ist der INT1-Eingang zur Aufrufzeit "1", wird das Programm beendet. Eine vorhandene Autostart-Kennung wird gelöscht.

Definition: void _INT1END(void)

Beispiel: _INT1END(); /* Programm bei INT1 = 1 beenden */

Name: **_T0END** (T0 Ende)

Ist der T0-Eingang zur Aufrufzeit "1", wird das Programm beendet. Eine vorhandene Autostart-Kennung wird gelöscht.

Definition: void _T0END(void)

Beispiel: _T0END(); /* Programm bei T0 = 1 beenden */

Name: **_T1END** (T1 Ende)

Ist der T1-Eingang zur Aufrufzeit "1", wird das Programm beendet. Eine vorhandene Autostart-Kennung wird gelöscht.

Definition: void _T1END(void)

Beispiel: _T1END(); /* Programm bei T1 = 1 beenden */

Name: **_START_OS** (Start Operating System)

Monitorprogramm der IS51 bei Adresse 0000h neu starten. Bei Autostartprogrammen muß vorher die Autostartkennung gelöscht werden, da das Anwendungsprogramm sonst sofort wieder gestartet wird.

Definition: void _START_OS(void)

Beispiel: _START_OS(); /* Monitorprogramm neu starten */

7.1.7 A/D-Wandlerfunktionen

Name: **_ADWI8** (8Bit Messwert von ADWI lesen)

Einen 8Bit-A/D-Wandlerwert vom angeschlossenen ADWI-Wandler lesen. Der Messwert wird als 8Bit-Wert zurückgegeben und in ADWERT (XRAM 807E/7Fh) abgelegt. Die Funktion darf nur aufgerufen werden, wenn der Wandler angeschlossen ist.

Definition: unsigned char _ADWI8(void)

Beispiel:
```
unsigned char bVar;
bVar = _ADWI8();          /* 8Bit A/D-Wert lesen */
```

Name: **_ADWI11** (11Bit Messwert von ADWI lesen)

Einen 11Bit-A/D-Wandlerwert vom angeschlossenen ADWI-Wandler lesen. Der Messwert wird als 16Bit-Wert zurückgegeben und in ADWERT (XRAM 807E/7Fh) abgelegt. Die Funktion darf nur aufgerufen werden, wenn der Wandler angeschlossen ist.

Definition: unsigned int _ADWI11(void)

Beispiel:
```
unsigned int wVar;
wVar = _ADWI11();         /* 11Bit A/D-Wert lesen */
```

Name: **_ADWI14** (14Bit Messwert von ADWI lesen)

Einen 14Bit-A/D-Wandlerwert vom angeschlossenen ADWI-Wandler lesen. Der Messwert wird als 16Bit-Wert zurückgegeben und in ADWERT (XRAM 807E/7Fh) abgelegt. Die Funktion darf nur aufgerufen werden, wenn der Wandler angeschlossen ist.

Definition: unsigned int _ADWI14(void)

Beispiel:
```
unsigned int wVar;
wVar = _ADWI14();         /* 14Bit A/D-Wert lesen */
```

Name: **_AD3162INST** (Interrupt Steuerung AD3162 installieren)

Den Interrupt und die Service Routine für den externen A/D-Wandler AD3162 installieren und freigeben. Als Interrupt wird der externe Interrupt INT0 verwendet. Der Wandler legt anschließend die Messergebnisse interruptgesteuert in ADWERT (XRAM 807E/7Fh) als UNSIGNED INT (10Bit) ab. Der Funktion wird die Priorität (0 = Low, 1 = High) des Interrupt als Parameter übergeben.

Definition: void _AD3162INST(unsigned char)

Beispiel:
```
_AD3162INST(1);           /* Interruptsteuerung AD3162 installieren, hohe Priorität */
```

Name: **_ADREAD** (A/D-Wert lesen)

Einen A/D-Wandler Wert aus ADWERT (XRAM 807E/7Fh) lesen. Falls kein Messwert vorhanden ist, wird -1 (0xFFFFh), sonst der Messwert als UNSIGNED INT zurückgegeben. Nach Auslesen eines Messwertes wird die Kennung ADOK (XRAM 807Dh) gelöscht.

Definition: unsigned int _ADREAD(void)

Beispiel:
```
unsigned int uiVar;
uiVar = _ADREAD();        /* A/D-Wert lesen */
```

7.1.8 Kurzübersicht der C-Funktionen im EPROM

Ausgabefunktionen:

Eingabefunktionen:

Serielle Schnittstelle:

Zeitfunktionen:

Codewandelfunktionen:

Arithmetikfunktionen:

Beendigungsfunktionen:

A/D-Wandlerfunktionen:

7.2 Programmbeispiele - C51

7.2.1 Lauflicht

Das Programm steuert ein Lauflicht für den Ausgabeport der Ein/Ausgabeplatine IS51-E/A1. Zur Zeitverzögerung wird die systemeigene Funktion ZSECH (Zehntel SECunden Hexadezimal) verwendet und mit einer Zeitverzögerung von 2 Zehntelsekunden programmiert. Das Programm kann durch drücken der T0-Taste der E/A-1 beendet werden. Bei Beendigung wird die Autostartkennung des Programms durch die Funktion CLRAUTO entfernt, wodurch es nach Reset nicht wieder gestartet wird. Das Programm als solches bleibt im Speicher bis es überschrieben wird. Es kann auch nach dem Löschen der Autostart-Kennung durch die Funktion 14h "Programm starten" des Treibers (IS51.COM) wieder ausgeführt werden.

Das Programm wird mit einem geeigneten C-Compiler als Autostart-Programm übersetzt und über eine serielle Schnittstelle in den Arbeitsspeicher der IS51 geladen. Durch die Autostart-Kennung wird es nach erfolgreicher Übertragung automatisch gestartet. Zu Beachten ist, daß ein eventuell im Speicher vorhandenes Autostart-Programm vor der Übertragung des neuen Programms gestopt werden muß.

```
/* Lauflicht für IS51, C-Programm */
/* Programmname: LLICHT51.C */

#pragma code ROM(LARGE)

#include <intrins.h>                    /* Wegen _crol_ einbinden */
#include <reg51.h>
#include <is51c.h>                      /* Funktionsdeklarationen für IS51 */

main()
{ unsigned char ucAusg = 1;             /* Variable für Ausgabe, Startwert 1 */
  while(!T0)                            /* Wiederholen bis T0-Taste gedrückt */
  { _OPLA1(ucAusg);                     /* Wert ausgeben */
    _ZSECH(2);                          /* 0,2 Sekunden warten */
    ucAusg = _crol_(ucAusg,1);          /* Ausgabewert links rotieren */
  }
  _CLRAUTO();                           /* Autostart- Kennung entfernen */
  _START_OS();                          /* Monitorprogramm neu starten */
}
```

Übersetzen:		C51 LLICHT51.C	
Linken:		L51 LLICHT51.OBJ, STUPIS51C.OBJ co(8100h)	
Umwandeln:		OHS51 LLICHT51	
Übertragen:		COPY LLICHT51.HEX COM1:	(vorher evtl. MODE COM1:48,n,8,1)
	oder	SEND LLICHT51 1 48 1	(Parameter werden automatisch eingestellt)

7.2.2 Schalter lesen

Die Schalterstellung der Erweiterungsplatine IS51-E/A1 werden mit Hilfe systemeigener Funktionen gelesen und zu den Leuchtdioden wieder ausgegeben. Das Programm wird durch drücken der T0-Taste beendet, wobei auch die Autostart-Kennung entfernt wird.

```
/* Schalter E/A-1 lesen, IS51 C-Programm */

#pragma code ROM(LARGE)

#include <reg51.h>
#include <is51c.h>

main()
{ unsigned char ucVar;
  while(!T0)
  { ucVar = IPLA1();      /* Schalterstellung lesen */
    _OPLA1(ucVar);        /* Zum Ausgabeport schreiben */
  }
  _CLRAUTO();             /* Autostart- Kennung löschen */
  _START_OS();            /* Neustart Monitorprogramm */
}
```

7.2.3 Bit abfragen

Wird die INT0-Taste gedrückt, leuchten nur die grünen LED´s, sonst leuchten nur die gelben LED´s. Das Programm wird durch die T0-Taste beendet.

```
/* INT0-Bit abfragen, IS51 C-Programm */

#pragma code ROM(LARGE)

#include <reg51.h>
#include <is51c.h>

typedef unsigned char BYTE;

main()
{ BYTE bVar;
  while(!T0)
  { if(INT0 == 1)             /* INT0-Bit = 1 ? */
      bVar = 0x24;            /* Falls ja, Ausgabewert für Grün */
    else
      bVar = 0x12;            /* Falls nein, Ausgabewert für Rot */
    _OPLA1(bVar);             /* Wert ausgeben */
  }
  _CLRAUTO();                 /* Autostart- Kennung löschen */
  _START_OS();                /* Monitorprogramm neu starten */
}
```

7.2.4 Interrupt

Der INT0-Interrupt wird freigegeben, auf fallende Flanke programmiert und es wird ihm eine hohe Priorität zugewiesen. Der Interrupt-Vektor wird in Adresse 8003h eingerichtet und zeigt auf die Interrupt-Service-Routine. Das Hauptprogramm steuert ein Lauflicht der Erweiterungsplatine IS51-E/A1, das beim Loslassen der INT0-Taste (fallende Flanke) das Hauptprogramm unterbricht und für eine Sekunde nur die roten LED´s einschaltet. Anschließend wird das Hauptprogramm fortgesetzt. Das Programm wird durch drücken der T0-Taste beendet, wobei auch die Autostart-Kennung gelöscht wird.

HINWEIS: Durch die Pragma-Anweisung NOINTVECTOR muß das automatische Erzeugen eines Interrupt-Vektors abgeschaltet werden, da dieser für eine EPROM-Adresse gebildet würde. In ein EPROM können aber keine Daten geschrieben werden. Interrupt-Vektoren werden mit der IS51 Systemfunktion INTVSET eingerichtet.

```
/* INT0-Interrupt,  IS51  C-Programm  */

#pragma  code  ROM(LARGE)
#pragma  NOINTVECTOR

#include  <reg51.h>
#include  <intrins.h>
#include  <is51c.h>

void  int0isr(void);              /* Funktions- Prototyp  */

main()
{ unsigned  char  ucVar  = 1;
  _INTVSET(0,&int0isr);           /* INT0  Vektor  einrichten  */
  PX0  = 1;                       /* Hohe  Priorität  */
  IT0  = 1;                       /* INT0  Flankengesteuert  */
  IE0  = 0;                       /* Alte  Anforderung  löschen  */
  IE  = 0x81;                     /* Nur  INT0  freigeben  */
  while(!T0)
  { _OPLA1(ucVar);                /* Lauflicht  in  Schleife  */
    _ZSECH(5);
    ucVar  = _crol_(ucVar,1);
  }
  _CLRAUTO();                     /* Autostart- Kennung  löschen  */
  _START_OS();                    /* Betriebssystem  neu  starten  */
}


/* Interrupt- Service- Routine  */

void  int0isr(void)
{ _OPLA1(0xC9);                   /* Rote  LED's  leuchten  */
  _ZSECH(10);                     /* für  1  Sekunde  */
}
```

8 IS51 und µVision

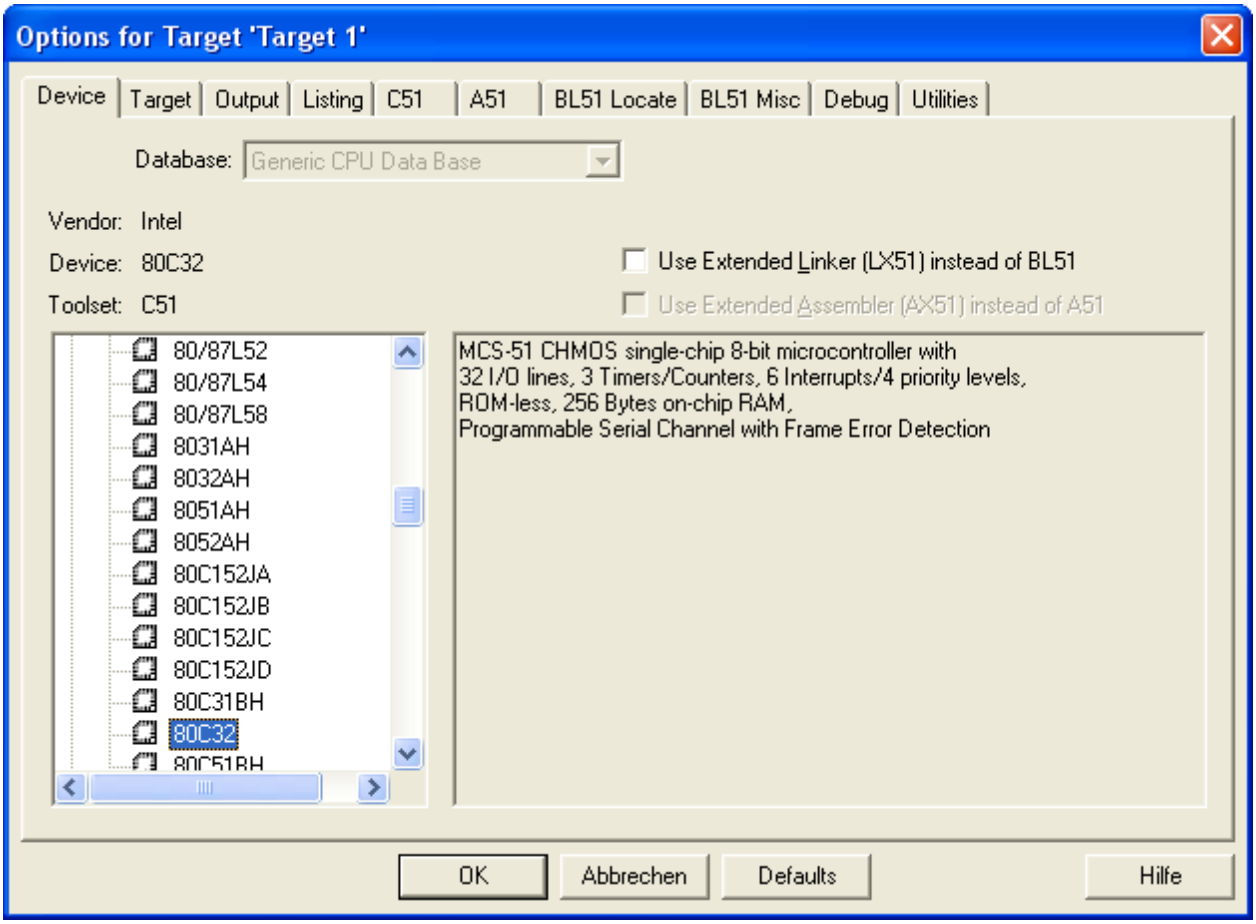
8.1 Allgemein

Die IS51 ist gut geeignet, mit der Keil Entwicklungsumgebung µVision zusammen zu arbeiten. Die Demoversion von µVision kann Programme bis zu einer Größe von 2kByte für Speicheradresse 8000h erzeugen. Der Speicher der IS51 für Downloadprogramme beginnt mit dieser Adresse. Dadurch ist es möglich kleine Assembler- oder C-Programme mit µVision zu erstellen und in der IS51 auszuführen.

Zum erlernen der Programmierumgebung und des Mikrokontrollers steht so ein komfortables Werkzeug zur Verfügung.

8.1.1 Device-Einstellung

In den Target-Optionen von µVision ist der verwendete Mikrokontroller auszuwählen. Im Beispiel wird der Intel-80C32 gewählt.



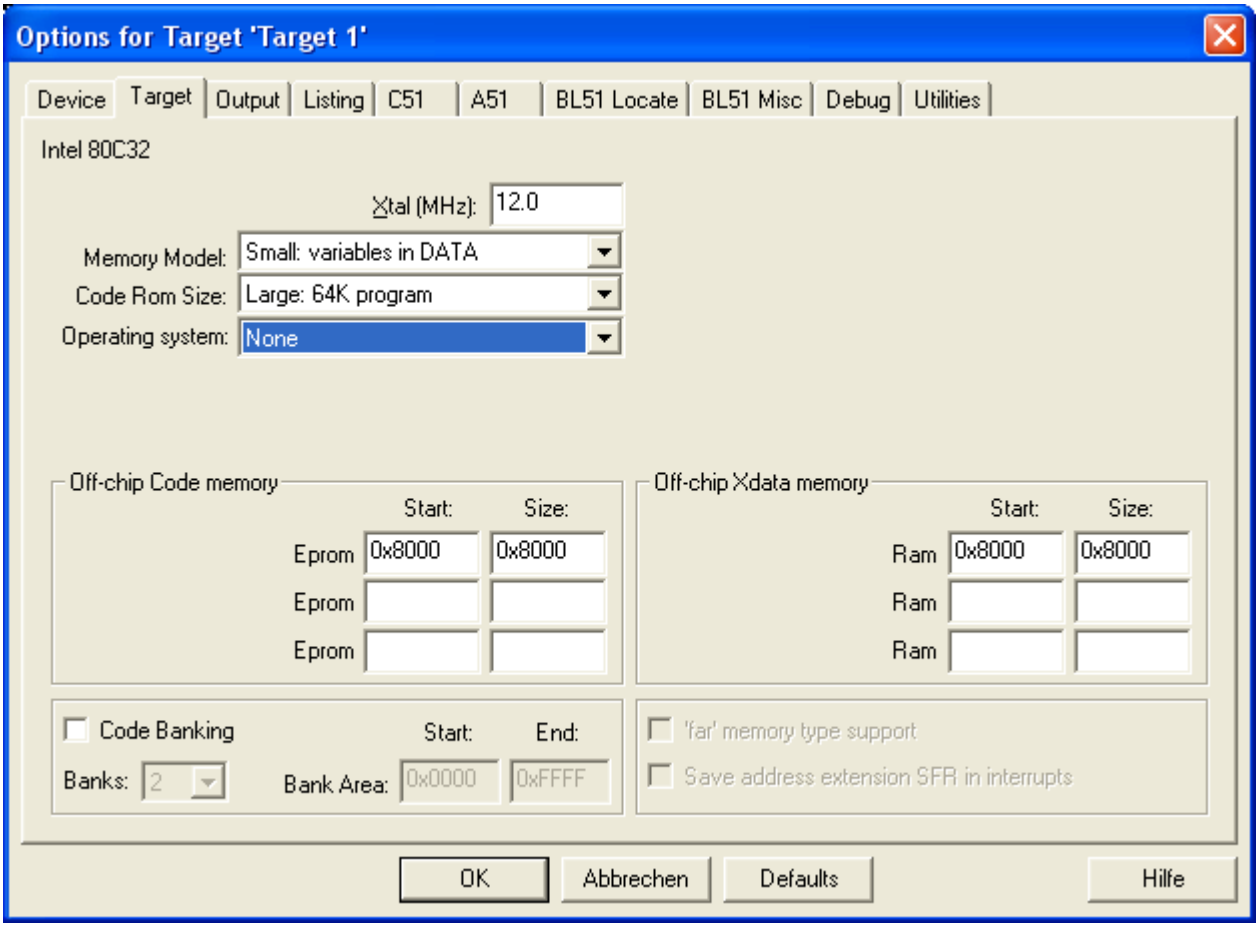
8.1.2 Target-Einstellungen

Als Takt wird die Quarzfrequenz der IS51 mit 12,0MHz eingestellt. Der Programmspeicher (Off-chip Code memory) beginnt bei 0x8000 (8000H) und hat eine maximale Größe von 8000H Bytes. Im Beispiel hat der Programmspeicher eine Größe von 0x8000 (von 8000h bis FFFFH). Der externe Datenspeicher hat die gleiche Größe und Lage. In der IS51 sind beide Speicher überlagert und werden nicht als getrennt angesprochen.

Für die Demoversion von µVision ist die Einstellung 2kByte Programmspeicher und 30kByte Datenspeicher gut geeignet.

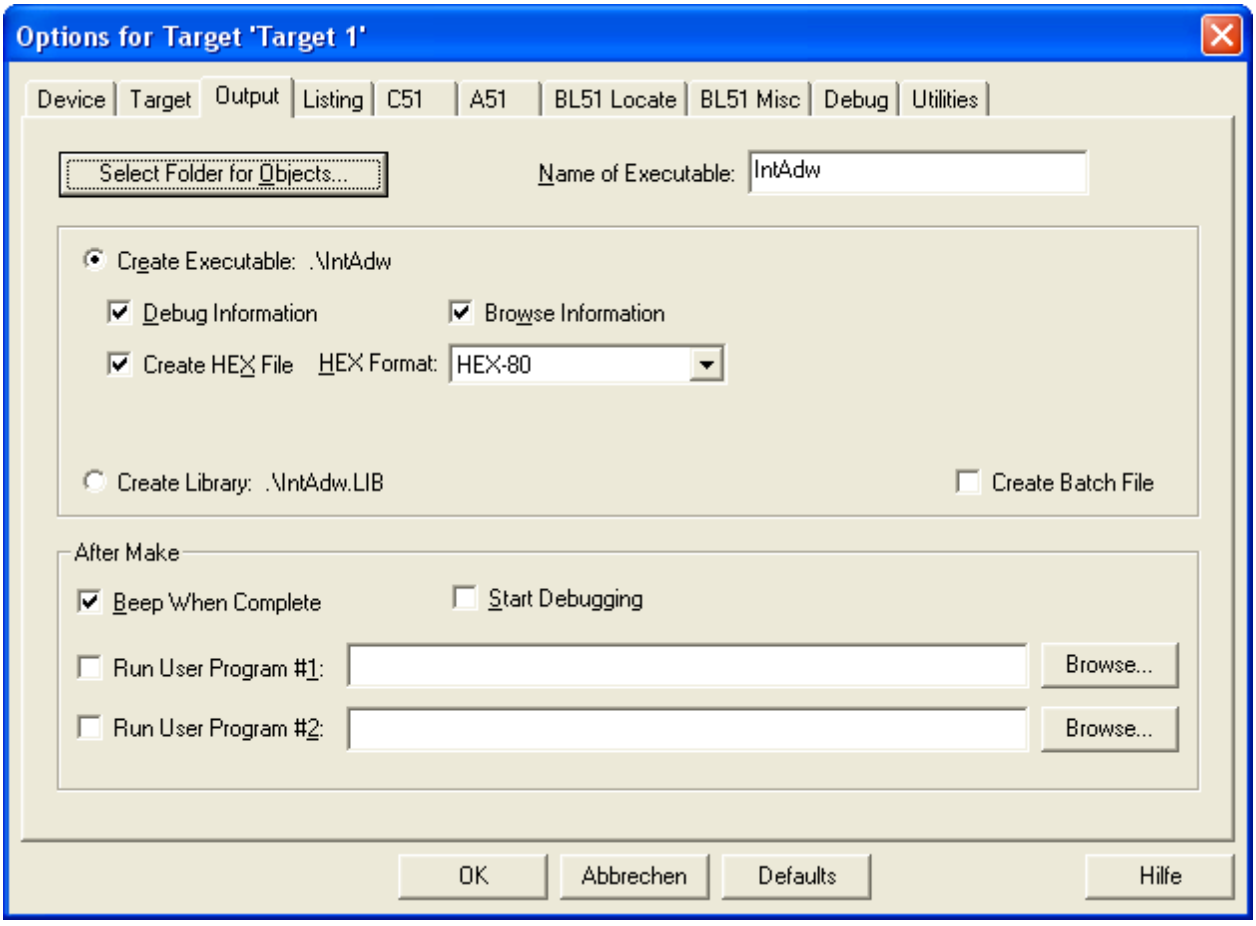
Off-chip Code memory Start = 0x8000, Size = 0x0800

Off-chip Xdata memory Start = 0x8800, Size = 0x7800



8.1.2 Output-Einstellungen

Um eine Intel-Hex-Datei des Programms zu erzeugen, muß Create HEX File aktiviert sein. Die Datei hat den in „Name of Executable“ eingestellten Dateinamen mit dem Dateiextend .HEX



8.2 Das Hilfsprogramm SEND.EXE

Das Hilfsprogramm SEND ist geeignet, Dateien im Intel-Hex-Format, über eine serielle Schnittstelle des PC zur IS51 zu übertragen. Da es sich dabei um ein unter Windows ausführbares Programm handelt, kann es in Entwicklungsumgebungen, wie beispielsweise Keil-µVision, eingebunden werden. Somit ist es möglich, direkt aus der Entwicklungsumgebung Programme zur IS51 zu senden.

Aufruf Allgemein: SEND [Dateiname] [Schnittstelle] [Baudrate] [Ausgabe]

Dateiname: Name der Intel-Hex Datei ohne Extend. den Extend „.HEX“ fügt SEND intern dazu

Schnittstelle: 1 = COM1, 2 = COM2, 3 = COM3.....

Baudrate: 48 = 4800Baud, 24 = 2400Baud..... (IS51Download immer 4800Baud)

Ausgabe: 1 = Send erzeugt Bildschirmausgaben, 0 = ohne Bildschirmausgaben

Beispiel: SEND TEST 1 48 0

Sendet die Datei TEST.HEX mit 4800Baud über die Schnittstelle COM1 des PC, ohne Bildschirmausgabe.

Ohne Entwicklungsumgebung, kann SEND.EXE auch aus einem DOS-Fenster gestartet werden. Bild 7.2 zeigt zwei mögliche Beispiele dafür. Im Bild ist Send einmal mit (… 48 1) und einmal ohne (… 48 0) Bildschirm-ausgabe zu sehen. Dabei wird jeweils die Datei TEST.HEX mit 4800 Baud über COM1: übertragen. Vorausgesetzt ist, dass sich das Programm SEND.EXE im Arbeitsverzeichnis befindet.

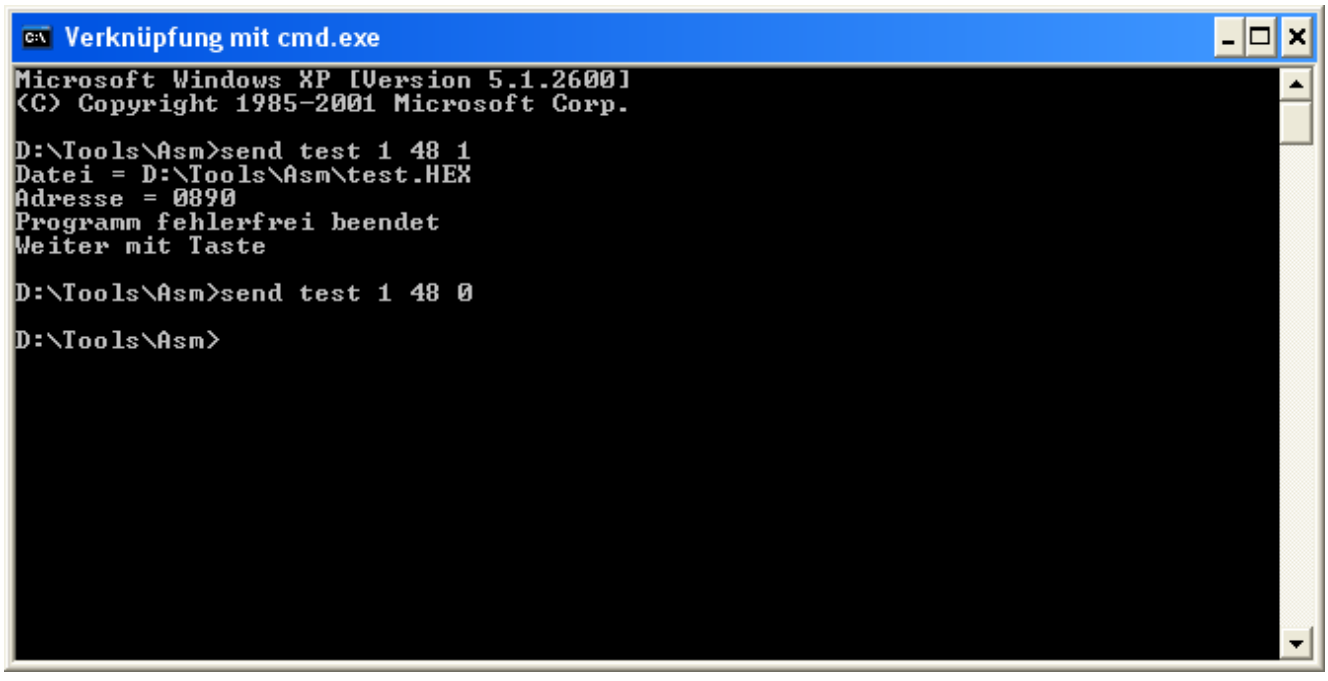


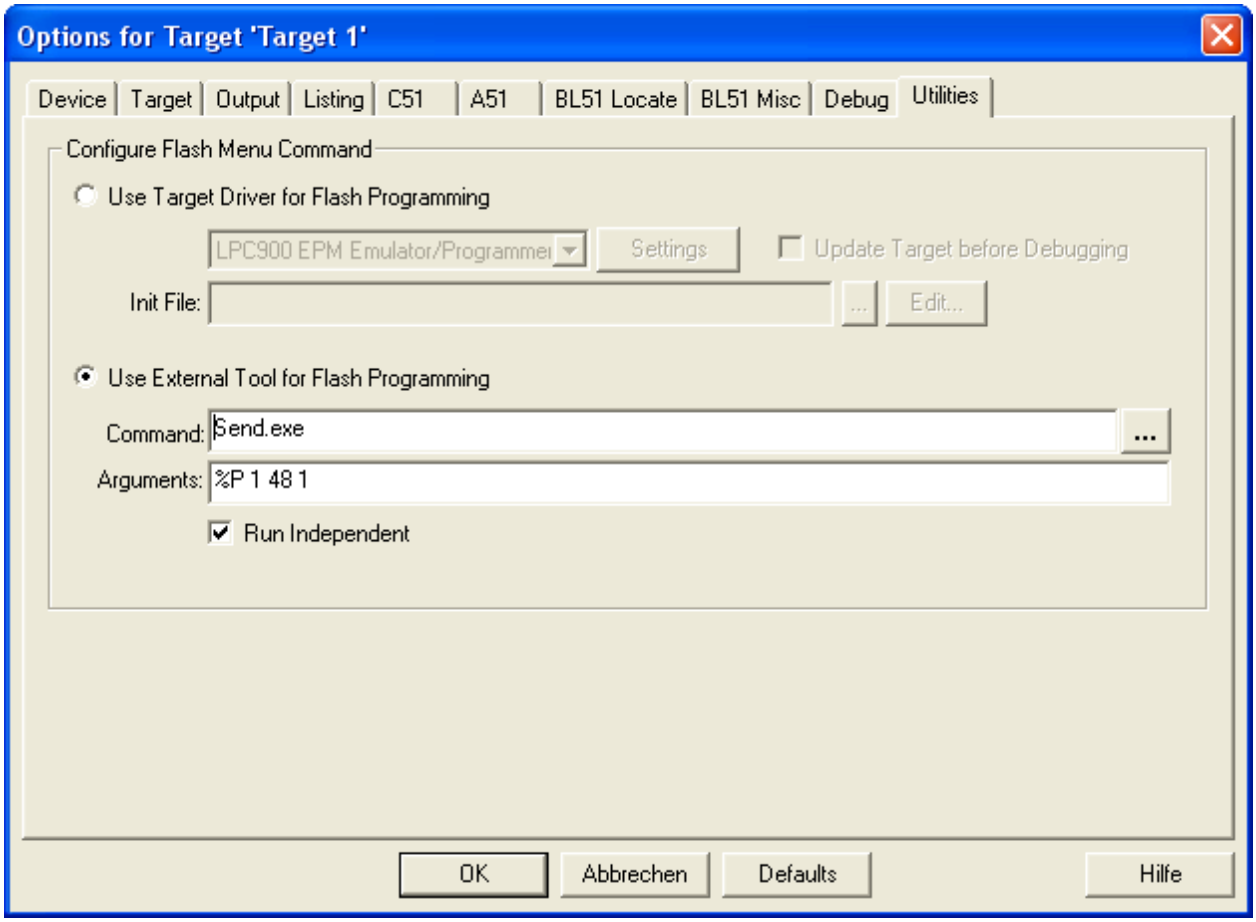
Bild 8.2 Send in einem DOS-Fenster ausgeführt

8.2.1 Utilities-Einstellungen für SEND (µVision)

Die Option „Use External Tool for Flash Programming“ ist zu aktivieren. Als „Command“ ist Send.exe einzutragen. Befindet sich das Hilfsprogramm „SEND.EXE“ nicht im Verzeichnis der Entwicklungsumgebung, ist der Pfad zur Datei voranzustellen. Z.B: D:\Hilfsprogramme\Send.exe

Die Argumente bezeichnen die Aufrufparameter von SEND.

%P	der aktuelle Dateiname in µVision (.HEX wird von SEND eingefügt)
1	serielle Schnittstelle COM1 (COM2 = 2 usw.)
48	4800 Baud
1	Bildschirmausgabe aktiv (0 = inaktiv)



Ist „Run Independent“ aktiviert, erfolgt die Ausgabe von SEND in einem eigenen Fenster, sonst im Nachrichtenfenster von µVision.

Ist diese Einstellung getroffen, kann das aktuell übersetzte Programm durch klicken auf den Button „Load“ oder über Menü – Flash – Download zur IS51 übertragen werden.

8.3 Programme für IS51 erstellen

Damit lauffähige Programme für die IS51 erstellt werden können, müssen dem Compiler/Assembler einige systemspezifische Eigenheiten der IS51 bekannt sein. Dazu zählen die Speicheradressen der Unterprogramme, die Autostartkennung, die Interrupt-Vektoren und die Adressen der Systemvariablen. Diese Informationen erhält der Compiler durch die Headerdateien „IS51xxx.H“ und eine der Startup-Dateien.

In der Bibliothek „IS51LIB.LIB“ befinden sich zudem die Unterprogramme des EPROM, so daß nach einbinden der Bibliothek auch die Unterprogramme zur Verfügung stehen. Dies wird zum debuggen benötigt, da der Debugger sonst die Unterprogramme nicht kennt. In der Demoversion von µVision ist es, aus Platzgründen nicht möglich, die Bibliothek mit einzubinden.

8.3.1 Die Headerdatei IS51x.H

Die Headerdatei definiert zunächst zwei Datentypen, BYTE und WORD. BYTE ist eine vorzeichenlose 8Bit-Variable, WORD ist eine vorzeichenlose 16Bit-Variable.

```
typedef unsigned char BYTE;
typedef unsigned int WORD;
```

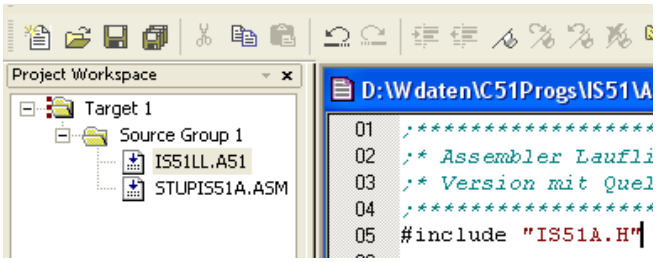
Diese Datentypen können in allen Programmen, welche die Headerdatei einbinden, genutzt werden.

Anschließend werden die Namen, die Aufrufparameter und die Rückgabewerte der Unterprogramme bekannt gemacht. Dies sind die unter 7.1 bereits beschriebenen Funktionen.

Für Assemblerprogramme erfüllt die Headerdatei „IS51A.H“ diese Aufgaben.

8.3.2 Die Startup Dateien

Es stehen, je nach Anforderung, mehrere Startup-Dateien zur Verfügung. Sie sind alle Assembler-Programme und werden in das µVision-Projekt eingebunden. Durch rechtsklick mit der Maus auf „Source Group“ öffnet sich ein Fenster, in dem mit „Add Files to Group ….“ Dateien dem Projekt hinzugefügt werden können. Durch rechtsklick auf eine der Projektdateien, öffnet sich ein Fenster, in dem eine Datei mit „Remove File ….“ wieder aus dem Projekt entfernt wird.

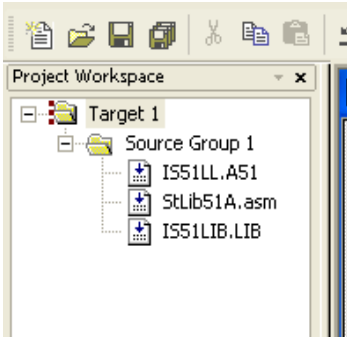


In den Startup-Dateien werden dem Compiler die Speicheradressen der Unterprogramme bekannt gemacht, der Stack und die Interrupt-Vektoren im RAM eingerichtet.

STUPIS51C.ASM	Für Auto/Fremdstart-Programme in Programmiersprache C.
StLib51C.ASM	Zum debuggen von C-Programmen, die Unterprogramme sind Teil des Quellcodes.
STUPIS51A.ASM	Für Auto/Fremdstart-Programme in Assembler.
StLib51A.ASM	Zum debuggen von Assemblerprogrammen, die Unterprogramme sind Teil des Quellcodes.

8.3.3 Die Libraries IS51LIB.LIB und IS51ELIB.LIB

Die Library IS51LIB.LIB



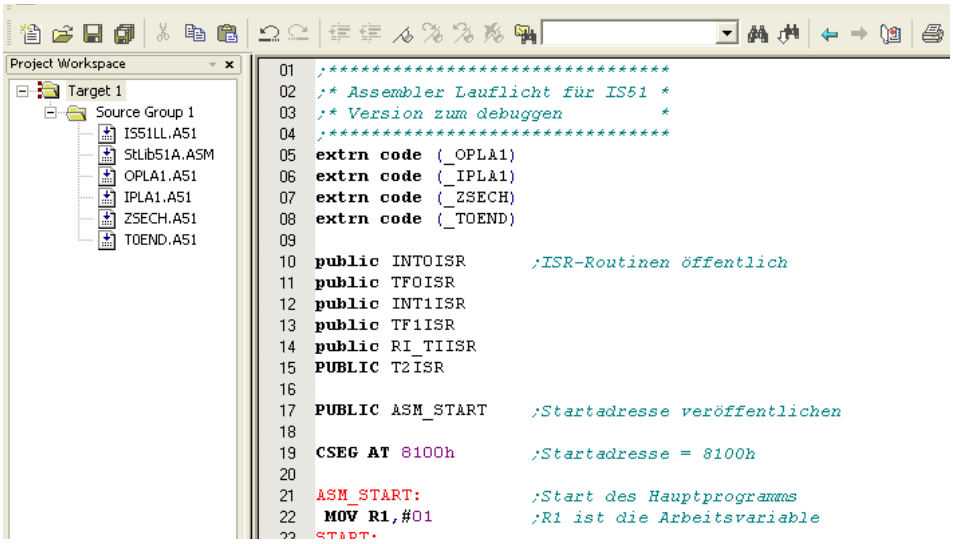
Die Bibliothek IS51LIB.LIB enthält die Unterprogramme aus dem EPROM der IS51. Für die Unterprogramme werden Adressen im RAM, also ab 8100h, generiert. Als Startup-Datei wird“ StLib51A.ASM“ für Assembler- und „StLib51C.ASM“ für C-Programme verwendet. In der Startupdatei kann festgelegt werden, ob auf Adresse 0000h ein Sprung zum Programmanfang eingefügt wird. In der Bibliothek beginnen die Namen der Unterprogramme mit einem Unterstrich , das Unterprogramm ZSECH aus dem EPROM wird dadurch zu _ZSECH. Um den eigenen Programmen die Namen der Unterprogramme bekannt zu machen, ist die Headerdatei „IS51LIB.H“ mit einzubinden. Unbenutzte Interrupts können aus dem Hauptprogramm entfernt und in der Startup-Datei deaktiviert werden.

Beispiel:

```
;**********************************
;* Assembler Lauflicht für IS51     *
;* Version zum debuggen             *
;**********************************
#include "IS51A.H"

public INT0ISR          ;ISR-Routinen
public TF0ISR           ;veröffentlichen
public INT1ISR
public TF1ISR
public RI_TIISR
PUBLIC T2ISR

PUBLIC ASM_START  ;Startadresse veröffentlichen

CSEG AT 8100h           ;Startadresse = 8100h

ASM_START:              ;Start des Hauptprogramms
  MOV A,#01             ;A ist die Arbeitsvariable
START:
  MOV R7,A
  LCALL _OPLA1          ;Ausgabe zu den LED's der E/A-1
  MOV R7,#2             ;Zeitwert für 0,2 Sekunden
  LCALL _ZSECH          ;warten
  RL A                  ;Links schieben
  LCALL _T0END          ;Beenden mit T0-Taste
  SJMP START            ;Endlosschleife

INT0ISR:    RETI  ;Interrupt-Service-Routinen (ISR)
TF0ISR:     RETI  ;*** Nicht benutzt *****
INT1ISR:    RETI
TF1ISR:     RETI
RI_TIISR:   RETI
T2ISR:      RETI

END
```

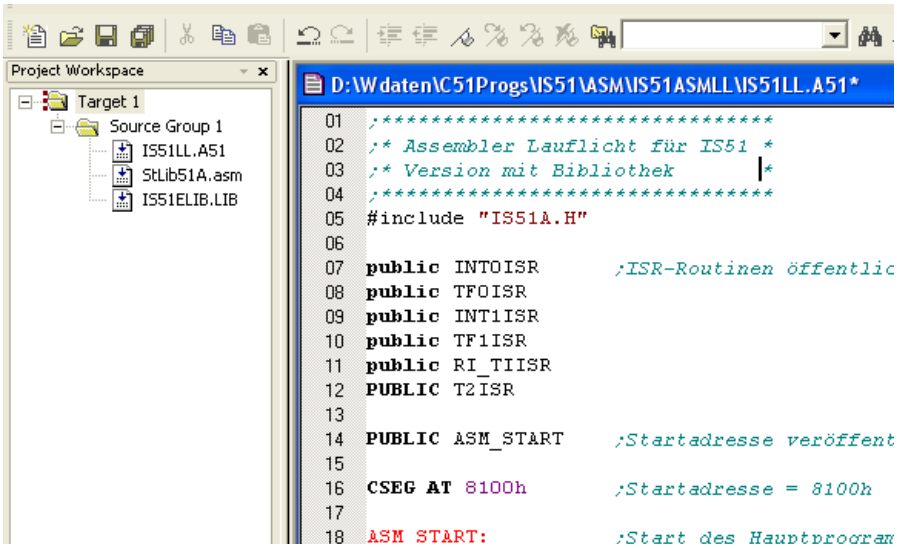
Um kleine Programme schreiben zu können, befinden sich die Unterprogramme der IS51 als Quelltext mit auf der CD. Im Verzeichnis „..\Library\Assembler\Source“ ist der Quellcode der einzelnen Unterprogramme in

Assemblercode zu finden. Die benötigten Routinen sind mit in das Projekt aufzunehmen und ihre Namen durch „EXTRN CODE (_name)“ dem Assembler bekannt zu machen. Auf diese Weise ist es möglich, kleine Programme zu erstellen, welche auch mit der eingeschränkten Demoversion von µVision bearbeitet werden können.



Die Library IS51ELIB.LIB

Die Bibliothek IS51ELIB.LIB enthält die Unterprogramme aus dem EPROM der IS51. Für die Unterprogramme werden die Originaladressen im EPROM generiert. Als Startup-Datei wird“ StLib51A.ASM“ oder „StLib51C.ASM“ verwendet. In der Startupdatei kann festgelegt werden, ob auf Adresse 0000h ein Sprung zum Programmanfang eingefügt wird. In der Bibliothek beginnen die Namen der Unterprogramme mit einem Unterstrich , das Unterprogramm ZSECH aus dem EPROM wird dadurch zu _ZSECH. Um den eigenen Programmen die Namen der Unterprogramme bekannt zu machen, ist die Headerdatei „IS51A.H“ oder „IS51C.H“ mit einzubinden. Unbenutzte Interrupts können aus dem Hauptprogramm entfernt und in der Startup-Datei deaktiviert werden.



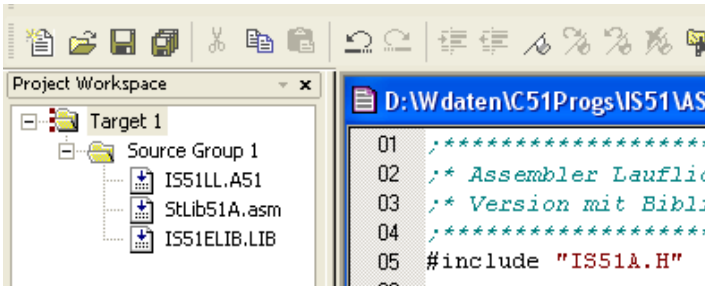
8.3.4 Einbinden der Library

Zum einbinden der Bibliotheken, ist die jeweils richtige Library-Datei (IS51LIB.LIB oder IS51ELIB.LIB) mit in das Projekt aufzunehmen und die Headerdatei (IS51A.H oder IS51C.H) durch eine include-Anweisung mit aufzunehmen.

#include <IS51A.H>

Befinden sich die Headerdatei nicht im aktuellen Verzeichnis, ist der Pfad zu den Datei mit anzugeben.

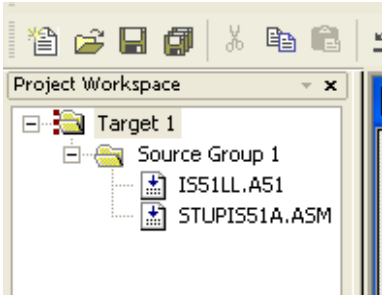
Z.B: #include <D:\MCS51\IS51LIB\IS51A.H>



8.3.5 Beispielprogramme

Lauflicht – in IS51 ausführbar (Assembler)

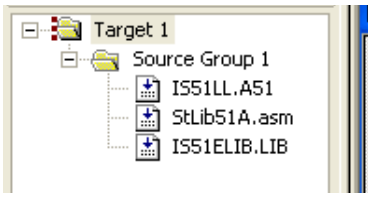
Das folgende Programm erzeugt mit den Leuchtdioden der Ein/Ausgabeplatine E/A1 ein Lauflicht. Es werden die Unterprogramme im EPROM benutzt, dazu wird die Startupdatei STUPIS51A.ASM mit ins Projekt aufgenommen, damit dem Assembler die Adressen der Unterprogramme bekannt sind. Für die Namen der Unterprogramme, ist die Headerdatei „IS51A.H“ zu verwenden. Da die Unterprogramme im EPROM ständig vorliegen, braucht keine Library eingebunden sein. Ohne angebundene Bibliothek ist die erzeugte Zieldatei kleiner, aber der Debugger kann die Unterprogramme nicht finden.



Wurde das Programm fehlerfrei übersetzt, kann die erzeugte Intel-Hex Datei mit SEND zur IS51 übertragen und dort ausgeführt werden.

```
;*******************************
;* Assembler Lauflicht für IS51 *
;* Version für Download in IS51 *
;*******************************
#include "IS51A.H"

public INT0ISR          ;ISR-Routinen öffentlich
public TF0ISR
public INT1ISR
public TF1ISR
public RI_TIISR
public T2ISR

PUBLIC ASM_START  ;Startadresse veröffentlichen
CSEG AT 8100h           ;Startadresse = 8100h

ASM_START:              ;Start des Hauptprogramms
  MOV A,#01             ;A ist die Arbeitsvariable
START:
  MOV R7,A
  LCALL _OPLA1          ;Ausgabe zu den LED's der E/A-1
  MOV R7,#2             ;Zeitwert für 0,2 Sekunden
  LCALL _ZSECH          ;warten
  RL A                  ;Links schieben
  LCALL _T0END          ;Beenden mit T0-Taste
  SJMP START            ;Endlosschleife

INT0ISR:    RETI  ;Interrupt-Service-Routinen (ISR)
TF0ISR:     RETI  ;*** Nicht benutzt *****
INT1ISR:    RETI
TF1ISR:     RETI
RI_TIISR:   RETI
T2ISR:      RETI

END
```

Lauflicht – zum debuggen mit Originaladressen (Assembler)



Das folgende Programm ist das Lauflicht für die Platine E/A1. Es benutzt die Unterprogramme der Bibliothek „IS51ELIB.LIB“ mit den originalen EPROM-Adressen. Somit kann es nicht zur IS51 übertragen werden, es ist ausschließlich zum untersuchen mit dem Debugger erstellt. Um für den Debugger den Sprung zum Programmanfang auf Adresse 0000h zu legen, sind in der Startupdatei „StLib51A.ASM“ die Kommentar-zeichen „;“ vor der Zeile (LJMP ASM_START) entfernt. Das Projekt beinhaltet als Hauptprogramm die Datei „IS51LL.A51“, als Startupdatei „StLib51A.ASM“ und als Bibliothek „IS51ELIB.LIB“. Im Hauptprogramm wird die Headerdatei „IS51A.H“ durch #include eingebunden.

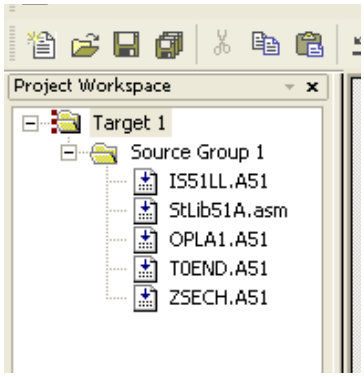
Startup - StLib51A.ASM:

```
CSEG  AT      0       ;Speicheradresse 0000h
LJMP  ASM_START       ;Sprung zum Programmstart
```

Hauptprogramm:

```
;*******************************
;* Assembler Lauflicht für IS51 *
;* Version mit Originaladressen *
;*******************************
#include "IS51A.H"

public INT0ISR          ;ISR-Routinen öffentlich
public TF0ISR
public INT1ISR
public TF1ISR
public RI_TIISR
public T2ISR

PUBLIC ASM_START  ;Startadresse veröffentlichen

CSEG AT 8100h           ;Startadresse = 8100h

ASM_START:              ;Start des Hauptprogramms
 MOV A,#01              ;A ist die Arbeitsvariable
START:
 MOV R7,A
 LCALL _OPLA1           ;Ausgabe zu den LED's der E/A-1
 MOV R7,#2              ;Zeitwert für 0,2 Sekunden
 LCALL _ZSECH           ;warten
 RL A                   ;Links schieben
 LCALL _T0END           ;Beenden mit T0-Taste
 SJMP START             ;Endlosschleife

INT0ISR:    RETI  ;Interrupt-Service-Routinen (ISR)
TF0ISR:     RETI  ;*** Nicht benutzt *****
INT1ISR:    RETI
TF1ISR:     RETI
RI_TIISR:   RETI
T2ISR:      RETI

END
```

Lauflicht – zum debuggen und Download (Assembler)



Das folgende Programm ist das Lauflicht für die Platine E/A1. Es benutzt nur die benötigten Unterprogramme und nur diese werden im Quelltext mit eingebunden. Somit kann es zur IS51 übertragen und im Debugger ausgeführt werden. Die Unterprogramme befinden sich nicht auf den Originaladressen. Das Projekt beinhaltet als Hauptprogramm die Datei „IS51LL.A51“, als Startupdatei „StLib51A.ASM“. Es ist keine Bibliothek und keine dazugehörige Headerdatei eingebunden.

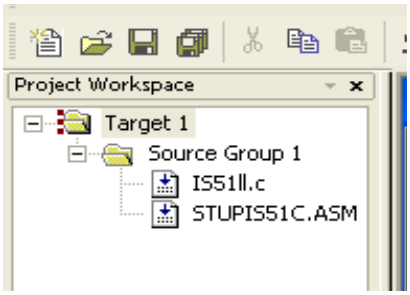
Um für den Debugger den Sprung zum Programmanfang auf Adresse 0000h zu legen, wird in der Startupdatei „StLib51A.ASM“ das Kommentar-zeichen „;“ vor der entsprechenden Zeile entfernt (LJMP ASM_START), zum Download ist die Zeile durch ein Semikolon auskommentiert.

```
Startup - StLib51A.ASM:          CSEG  AT     0
(Debug)                          LJMP  ASM_START   ;Sprung zum Programmstart

Startup - StLib51A.ASM:          CSEG  AT     0
(Download)               ;       LJMP  ASM_START   ;Sprung zum Programmstart
```

```
;*******************************
;* Assembler Lauflicht für IS51 *
;* Version mit Quellcodeunterp. *
;*******************************
EXTRN CODE (_OPLA1)
EXTRN CODE (_T0END)
EXTRN CODE (_ZSECH)

public INT0ISR          ;ISR-Routinen öffentlich
public TF0ISR
public INT1ISR
public TF1ISR
public RI_TIISR
public T2ISR

PUBLIC ASM_START  ;Startadresse veröffentlichen

CSEG AT 8100h           ;Startadresse = 8100h

ASM_START:              ;Start des Hauptprogramms
 MOV A,#01              ;A ist die Arbeitsvariable
START:
 MOV R7,A
 LCALL _OPLA1           ;Ausgabe zu den LED's der E/A-1
 MOV R7,#2              ;Zeitwert für 0,2 Sekunden
 LCALL _ZSECH           ;warten
 RL A                   ;Links schieben
 LCALL _T0END           ;Beenden mit T0-Taste
 SJMP START             ;Endlosschleife

INT0ISR:    RETI  ;Interrupt-Service-Routinen (ISR)
TF0ISR:     RETI  ;*** Nicht benutzt *****
INT1ISR:    RETI
TF1ISR:     RETI
RI_TIISR:   RETI
T2ISR:      RETI

END
```

Lauflicht – in IS51 ausführbar (C-Programm))

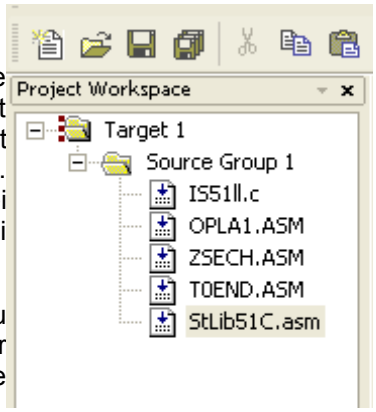
Das folgende Programm erzeugt mit den Leuchtdioden der Ein/Ausgabeplatine E/A1 ein Lauflicht. Es werden die Unterprogramme im EPROM benutzt, dazu wird die Startupdatei „STUPIS51C.ASM“ mit ins Projekt aufgenommen, damit dem Assembler die Adressen der Unterprogramme bekannt sind. Für die Namen der Unterprogramme, ist die Headerdatei „IS51C.H“ zu verwenden. Da die Unterprogramme im EPROM ständig vorliegen, braucht keine Library eingebunden sein. Ohne angebundene Bibliothek ist die erzeugte Zieldatei kleiner, aber der Debugger kann die Unterprogramme nicht finden.



Wurde das Programm fehlerfrei übersetzt, kann die erzeugte Intel-Hex Datei mit SEND zur IS51 übertragen und dort ausgeführt werden.

```
/*********************************
 * Lauflicht für IS51 C-Programm *
 * Version zum ausführen in IS51 *
 *********************************/
#include <REG52.H>
#include <intrins.h>    // wegen Rotation „_crol_“
#include "IS51C.H"

main()
{ BYTE bZlr = 0x01;           // Ausgabevariable
  for(;;)                     // Endlosschleife
  { bZlr = _crol_(bZlr,1);    // Links rotieren
    _OPLA1(bZlr);             // Ausgeben
    _ZSECH(3);                // warten
    _T0END();                 // Mit T0-Taste beenden

  }
}

/*****************************************
* Interrupt-Service-Routinen              */

void INT0ISR (void) {}        // INT0 Interrupt
void TF0ISR (void) {}         // TF0 Interrupt
void INT1ISR (void) {}        // INT1 Interrupt
void TF1ISR (void) {}         // TF1 Interrupt
void RI_TIISR(void) {}        // RI/TI Interrupt (UART)
void T2ISR (void) {}          // T2 Interrupt
```

Lauflicht – zum debuggen und Download (C-Programm)



Das folgende Programm ist das Lauflicht für die Platine E/A1. Es benutzt nur die benötigten Unterprogramme und nur diese werden im Quelltext mit eingebunden. Somit kann es zur IS51 übertragen und im Debugger ausgeführt werden. Die Unterprogramme befinden sich nicht auf den Originaladressen. Das Projekt beinhaltet als Hauptprogramm die Datei „IS51LL.C“, als Startupdatei „StLib51C.ASM“. Es ist keine Bibliothek und keine dazugehörige Headerdatei eingebunden.

Um für den Debugger den Sprung zum Programmanfang auf Adresse 0000h zu legen, wird in der Startupdatei „StLib51A.ASM“ das Kommentar-zeichen „;“ vor der entsprechenden Zeile entfernt (LJMP ASM_START), zum Download ist die Zeile mit dem Semikolon auskommentiert.

```
Startup - StLib51A.ASM:          CSEG  AT      0
(Debug)                          LJMP  ASM_START   ;Sprung zum Programmstart

Startup - StLib51A.ASM:          CSEG  AT      0
(Download)              ;        LJMP  ASM_START   ;Sprung zum Programmstart
```

```
/************************************
 * Lauflicht für IS51 C-Programm     *
 * Version zum ausführen und debuggen *
 ************************************/
#include <REG52.H>
#include <intrins.h>    // wegen _crol_

typedef unsigned char BYTE;

extern void OPLA1(BYTE);
extern void ZSECH(BYTE);
extern void _T0END(void);

main()
{ BYTE bZlr = 0x01;           // Ausgabevariable
  for(;;)                     // Endlosschleife
  { bZlr = _crol_(bZlr,1);    // Links rotieren
    OPLA1(bZlr);              // Ausgeben
    ZSECH(3);                 // warten
    _T0END();                 // Mit T0-Taste beenden
  }
}

/****************************************
* Interrupt-Service-Routinen             */
void INT0ISR (void) {}        // INT0 Interrupt
void TF0ISR (void) {}         // TF0 Interrupt
void INT1ISR (void) {}        // INT1 Interrupt
void TF1ISR (void) {}         // TF1 Interrupt
void RI_TIISR(void) {}        // RI/TI Interrupt (UART)
void T2ISR (void) {}          // T2 Interrupt
```

A Anhang

A.1 Befehlsliste 8051

A.1.1 Allgemeine Transferbefehle

	A	Rr	Dadr	#ko	@Ri
MOV A,	Nein	Ja	Ja	Ja	Ja
MOV Rr,	Ja	Nein	Ja	Ja	Nein
MOV dadr,	Ja	Ja	Ja	Ja	Nein
MOV @Ri,	Ja	Nein	Ja	Ja	Nein
MOV DPTR,	Nein	Nein	Nein	Ja*	Nein

* = 16Bit Konstante

A.1.2 Spezielle Transferbefehle

Ext. Programmspeicher:	MOVC A,@A+PC	(A <- <<A+PC>>)
	MOVC A,@A+DPTR	(A <- <<A+DPTR>>)
Externer Datenspeicher:	MOVX A,@Ri	(A <- <<Ri>>)
	MOVX A,@DPTR	(A <- <<DPTR>>)
	MOVX @Ri,A	(<Ri> <- <A>)
	MOVX @DPTR,A	(<DPTR> <- <A>)
Stack:	PUSH dadr	(Stack <- <dadr>)
	POP dadr	(dadr <- <Stack>)
Tauschbefehle:	XCH A,Rr	(<A> <-> <Rr>)
	XCH A,dadr	(<A> <-> <dadr>)
	XCH A,@Ri	(<A> <-> <<Ri>>)
	XCHD A,@Ri	(<$A2^0$-2^3> <-> <<Ri 2^0-2^3>>)
	SWAP A	(<$A2^0$-2^3> <-> <$A2^4$-2^7>)

A.1.2 Logik-Befehle

	A	Rr	dadr	#ko.	@Ri	Flags
ANL A,	-	Ja	Ja	Ja	Ja	P
ANL dadr,	Ja	-	-	Ja	-	-
ORL A,	-	Ja	Ja	Ja	Ja	P
ORL dadr,	Ja	-	-	Ja	-	-
XRL A,	-	Ja	Ja	Ja	Ja	P
XRL dadr,	Ja	-	-	Ja	-	-

ANL Ziel,Quelle (<Ziel> UND <Quelle> -> Ziel)

ORL Ziel,Quelle (<Ziel> ODER <Quelle> -> Ziel)

XRL Ziel,Quelle (<Ziel> EX-OR <Quelle> -> Ziel)

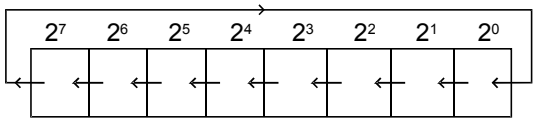
A.1.2.1 Spezielle Logik-Befehle

CLR A 00 -> A Flags: P

CPL A <A> NICHT -> A Flags: -

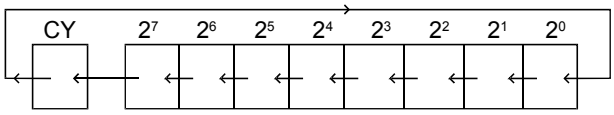
A.1.3 Rotier-Befehle

RL A



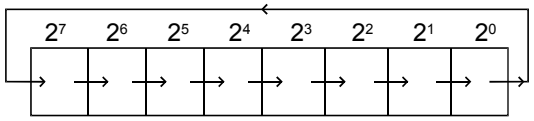
AKKU Flags: -

RLC A



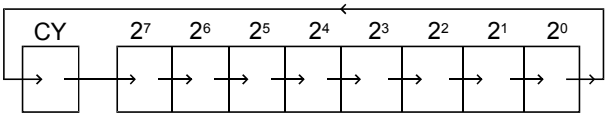
AKKU Flags: CY, P

RR A



AKKU Flags: -

RRC A



AKKU Flags: CY, P

A.1.4 Arithmetik-Befehle

	Rr	dadr	#ko.	@Ri	Funktion	Flags
ADD A,	Ja	Ja	Ja	Ja	<A> + X -> A	OV,P
ADDC A,	Ja	Ja	Ja	Ja	<A> + X + CY -> A	OV,P
SUBB A,	Ja	Ja	Ja	Ja	<A> - X - CY -> A	OV,P

	A	Rr	dadr	@Ri	DPTR	Funktion	Flags
INC	Ja	Ja	Ja	Ja	Ja	<X> + 1 -> X	INC A: P
DEC	Ja	Ja	Ja	Ja	-	<X> - 1 -> X	DEC A: P

MUL AB <A> x <B> -> A (LSB) und B (MSB) Flags: Cy,OV,P

DIV AB <A> / <B> -> A (Rest in B) Flags: Cy,OV,P

DA A Dezimalkorrektur des <A> nach Addition Flags: Cy,AC,P

A.1.5 Bitverarbeitungs-Befehle

CLR C 0 -> Cy Flags: P

CLR badr 0 -> Bit (falls Akkubit) Flags:P

SETB C 1 -> Cy Flags: Cy

SETB badr 1 -> Bit Flags: -

CPL C <Cy> NICHT -> Cy Flags: Cy

CPL badr <Bit> NICHT -> Bit Flags: P

ANL C,badr <Cy> UND <badr> -> Cy Flags: Cy

ANL C,/badr <CY> UND <badr (inv.)> -> Cy Flags: Cy

ORL C,badr <Cy> ODER <badr> -> Cy Flags: Cy

ORL C,/badr <Cy> ODER <badr (inv.)> -> Cy Flags: Cy

MOV C,badr <badr> -> Cy Flags: Cy

MOV badr,C <Cy> -> badr Flags: -

A.1.6 Unterprogramm-Befehle

ACALL adr11	<PC+2> -> Stack, <PC 2^{11}-2^{15}> + adr11 -> PC	Flags: -
LCALL adr16	<PC+3> -> Stack, adr16 -> PC	Flags: -
RET	<Stack> -> PC	Flags: -
RETI	<Stack> -> PC, Interrupt freigeben	Flags: -

A.1.7 Sprungbefehle

AJMP adr11	<PC 2^{11}-2^{15}> + adr11 -> PC	Flags: -
LJMP adr16	adr16 -> PC	Flags: -
SJMP rel	<PC+2> ± rel	Flags: -
JMP @A+DPTR	<A> + <DPTR> -> PC	Flags: -
JZ rel	Falls <A> = 00: <PC+2> ± rel -> PC Falls <A> <> 00: <PC+2> -> PC	Flags: -
JNZ rel	Falls <A> <> 00: <PC+2> ± rel -> PC Falls <A> = 00: <PC+2> -> PC	Flags: -
JC rel	Falls <Cy> = 1: <PC+2> ± rel -> PC Falls <CY> = 0: <PC+2> -> PC	Flags: -
JNC rel	Falls <Cy> = 0: <PC+2> ± rel -> PC Falls <Cy> = 1: <PC+2> -> PC	Flags: -
JB badr,rel	Falls <badr> = 1: <PC+2> ± rel -> PC Falls <badr> = 0: <PC+2> -> PC	Flags: -
JNB badr,rel	Falls <badr> = 0: <PC+2> ± rel -> PC Falls <badr> = 1: <PC+2> -> PC	Flags: -
JBC badr,rel	Wie JB aber das Bit wird gelöscht	

	dadr,	#ko.,	Adr.	Flags
CJNE A,	Ja	Ja	rel	CY
CJNE Rr,	-	Ja	rel	CY
CJNE @Ri,	-	Ja	rel	CY

Falls ungleich: <PC+2> ± rel -> PC

Falls gleich: <PC+2> -> PC

Mögliche Formen:

CJNE A,dadr,rel CJNE Rr,#ko.,rel CJNE A,#ko.,rel CJNE @Ri,#ko.,rel

DJNZ Rr,rel	Falls <Rr> - 1 <> 00:	<PC+2> ± rel -> PC	
	Falls <Rr> - 1 = 00:	<PC+2> -> PC	Flags: -
DJNZ dadr,rel	Falls <dadr> - 1 <> 00:	<PC+2> ± rel -> PC	
	Falls <dadr> - 1 = 00:	<PC+2> -> PC	Flags: P, falls dadr = A

A.1.8 Sonder-Befehl

NOP	Leerbefehl (No OPeration)

A.1.9 Verwendete Kurzzeichen:

Rr	Register (R0..R7)
dadr	direkt adressierbare Speicherzelle
@Ri	indirekt adressierbare Speicherzelle (durch R0 oder R1)
#ko.	Konstante (8-Bit)
badr	Bitadresse
adr11	11-Bit Adresse (Bereich 2K)
adr16	16-Bit Adresse (Bereich 64K)
rel	relative 8-Bit Adresse (Bereich +127 bis -128)

A.2 Befehlsbyte und Zyklen

Transferbefehle

Befehl	Byte	Zyklen	Befehl	Byte	Zyklen
MOV A,Rr	1	1	MOV A,dadr	2	1
MOV A,#ko	2	1	MOV A,@Ri	1	1
MOV Rr,A	1	1	MOV Rr,dadr	2	2
MOV Rr,#ko	2	1	MOV dadr,A	2	1
MOV dadr,Rr	2	2	MOV dadr,dadr	3	2
MOV dadr,#ko	2	1	MOV dadr,@Ri	2	2
MOV @Ri,A	1	1	MOV @Ri,dadr	2	2
MOV @Ri,#ko	2	1	MOV DPTR,#ko	3	2
MOVC A,@A+PC	1	2	MOVC A,@A+DPTR	1	2
MOVX A,@Ri	1	2	MOVX A,@DPTR	1	2
MOVX @Ri,A	1	2	MOVX @DPTR,A	1	2
PUSH dadr	2	2	POP dadr	2	2
XCH A,Rr	1	1	XCH A,dadr	2	1

Befehl	Byte	Zyklen	Befehl	Byte	Zyklen
XCH A,@Ri	1	1	XCHD A,@Ri	1	1
SWAP A	1	1			

Logik-Befehle

ANL A,Rr	1	1	ANL A,dadr	2	1
ANL A,#ko	2	1	ANL A,@Ri	1	1
ANL dadr,A	2	1	ANL dadr,#ko	3	2
ORL A,Rr	1	1	ORL A,dadr	2	1
ORL A,#ko	2	1	ORL A,@Ri	1	1
ORL dadr,A	2	1	ORL dadr,#ko	3	2
XRL A,Rr	1	1	XRL A,dadr	2	1
XRL A,#ko	2	1	XRL A,@Ri	1	1
XRL dadr,A	2	1	XRL dadr,#ko	2	1
CLR A	1	1	CPL A	1	1

Arithmetik-Befehle

RL A	1	1	RR A	1	1
RLC A	1	1	RRC A	1	1
ADD A,Rr	1	1	ADD A,dadr	2	1
ADD A,#ko	2	1	ADD A,@Ri	1	1
ADDC A,Rr	1	1	ADDC A,dadr	2	1
ADDC A,#ko	2	1	ADDC A,@Ri	1	1
SUBB A,Rr	1	1	SUBB A,dadr	2	1
SUBB A,#ko	2	1	SUBB A,@Ri	1	1
INC A	1	1	INC Rr	1	1
INC dadr	2	1	INC @Ri	1	1
INC DPTR	1	2	DEC A	1	1
DEC Rr	1	1	DEC dadr	2	1
DEC @Ri	1	1	DA A	1	1
MUL AB	1	4	DIV AB	1	4

Bit-Befehle

CLR C	1	1	CLR badr	2	1
SETB C	1	1	SETB badr	2	1
CPL C	1	1	CPL badr	2	1

Befehl	Byte	Zyklen	Befehl	Byte	Zyklen
ANL C,badr	2	2	ANL C,/badr	2	2
ORL C,badr	2	2	ORL C,/badr	2	2
MOV C,badr	2	1	MOV badr,C	2	2

Sprung- und Unterprogramm Befehle

ACALL adr11	2	2	LCALL adr16	3	2
SJMP rel	2	2	JMP @A+DPTR	1	2
RET	1	2	RETI	1	2
JZ rel	2	2	JNZ rel	2	2
JC rel	2	2	JNC rel	2	2
JB rel	3	2	JNB rel	3	2
JBC rel	3	2			
CJNE A,dadr,rel	3	2	CJNE A,#ko,rel	3	2
CJNE Rr,#ko,rel	3	2	CJNE @Ri,#k,rel	3	2
DJNZ Rr,rel	2	2	DJNZ dadr,rel	3	2

A.3 HEX-DEZ Umrechnungstabelle

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
80	128	90	144	A0	160	B0	176	C0	192	D0	208	E0	224	F0	240
81	129	91	145	A1	161	B1	177	C1	193	D1	209	E1	225	F1	241
82	130	92	146	A2	162	B2	178	C2	194	D2	210	E2	226	F2	242
83	131	93	147	A3	163	B3	179	C3	195	D3	211	E3	227	F3	243
84	132	94	148	A4	164	B4	180	C4	196	D4	212	E4	228	F4	244
85	133	95	149	A5	165	B5	181	C5	197	D5	213	E5	229	F5	245
86	134	96	150	A6	166	B6	182	C6	198	D6	214	E6	230	F6	246
87	135	97	151	A7	167	B7	183	C7	199	D7	215	E7	231	F7	247
88	136	98	152	A8	168	B8	184	C8	200	D8	216	E8	232	F8	248
89	137	99	153	A9	169	B9	185	C9	201	D9	217	E9	233	F9	249
8A	138	9A	154	AA	170	BA	186	CA	202	DA	218	EA	234	FA	250
8B	139	9B	155	AB	171	BB	187	CB	203	DB	219	EB	235	FB	251
8C	140	9C	156	AC	172	BC	188	CC	204	DC	220	EC	236	FC	252
8D	141	9D	157	AD	173	BD	189	CD	205	DD	221	ED	237	FD	253
8E	142	9E	158	AE	174	BE	190	CE	206	DE	222	EE	238	FE	254
8F	143	9F	159	AF	175	BF	191	CF	207	DF	223	EF	239	FF	255

Zum Umrechnen der Hexcodes 00-7Fh kann die ASCII-Tabelle benutzt werden.

A.4 ASCII-Tabelle

HEX	DEZ	ASCII	HEX	DEZ	ASCII	HEX	DEZ	ASCII	HEX	DEZ	ASCII
00	0	NUL	20	32	SP	40	64	@	60	96	`
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	´	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

SOH =
STX = Start of Text
ETX = End of Text
EOT =
ENQ =
ACK = Acknowledge
BEL = Bell

BS = Backspace
HT = Horizontal Tab
LF = Line Feet
VT = Vertical Tab
FF = Form Feet
CR = Cariage Return
SO =

SI =
DLE =
DC1-4 = Device Control 1-4
NAK = Neg. ACK
ESC = Escape
SP = Space
DEL = Delete