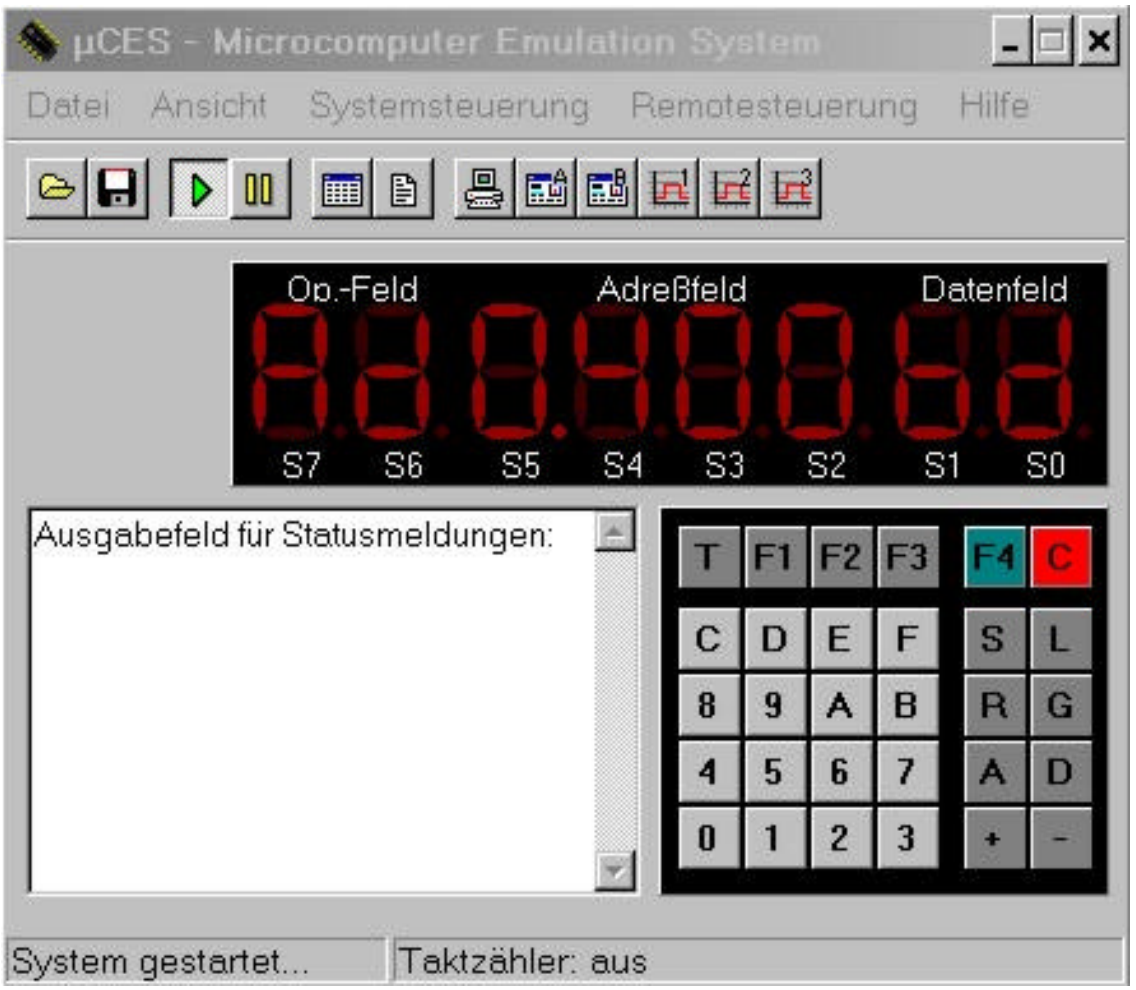
Skript zum Mikrorechner-Praktikum

Kapitel 3:

Aufbau des Praktikumsrechners und der Betriebssoftware

Autor: Helmut Bähring



Inhalt des Kapitels 3:

3. Aufbau des Praktikumsrechners und der Betriebssoftware

Nachdem Sie nun bereits die Bedienung des Mikrorechners beherrschen und die Architektur des Mikroprozessors 6809 und seine "Sprache" kennengelernt haben, werden wir in den folgenden Abschnitten versuchen, Sie so weit wie möglich mit dem Innenleben des Mikrorechners, mit seiner Architektur und seinen Komponenten vertraut zu machen.

3.1 Die Architektur des Praktikumsrechners

3.1.1 Blockschaltbild des Gesamtsystems

In den vorhergehenden Abschnitten mußten wir notgedrungen schon mehrfach einzelne Komponenten des Mikrorechners erwähnen. Das Bild 3.1-1 zeigt Ihnen nun das Blockschaltbild des gesamten Systems.

Darin erkennen Sie ganz unten den Mikroprozessor MC6809 mit seinem Taktgenerator und der Rücksetzlogik *(Reset*). Diese ermöglicht es, den Prozessor hardwaremäßig in einen definierten Grundzustand zu versetzen. Die Verbindung zwischen allen Komponenten wird von dem Bussystem aus Adreß-, Daten- und Steuerbus vorgenommen. Um den Rechner durch externe Komponenten erweitern zu können, ist das Bussystem über Treiberbausteine auf eine 64-polige Steckerleiste gelegt worden, die an der Rückwand des Gehäuses zugänglich ist.

Auf der linken Seite des Bussystems ist der Arbeitsspeicher des µC gezeichnet. Implementiert sind jeweils 8 kbyte RAM- bzw. EPROM-Speicher[1]. Zwei Steckplätze sind für die Erweiterung um zusätzlich 2x8-kbyte vorhanden. Die Aktivierung (*Enable*) aller Systemkomponenten geschieht durch die Adreßauswahl-Logik, die mit den Signalen des Adreßbusses mehr oder weniger große Adreßbereiche selektiert. Diese Adreßbereiche, unter denen die Komponenten vom µP angesprochen werden, sind in hexadezimaler Form an die einzelnen Komponenten geschrieben worden. Die Adreßauswahl ist so ausgelegt, daß durch das Umsetzen von Brücken und den Einsatz höher integrierter Speicherbausteine der gesamte Adreßraum von 64 kbyte auf der Systemplatine bestückt werden kann.

Rechts vom Bussystem sind im Bild 3.1-1 die Komponenten angeordnet, die die Kommunikation mit der Außenwelt, der Peripherie, abwickeln. Das Standard-Ausgabemedium ist die Anzeige, das Standard-Eingabemedium die Tastatur. Die einfachste Schnittstelle wird vom Parallel-Interface geboten. Dieses liefert einen 8 bit breiten Eingabe/Ausgabe-Port mit zwei zusätzlichen Steuerleitungen, der außen über 2-mm-Buchsen zugänglich ist. Dadurch ist es möglich, den Praktikumsrechner mit Schaltungen auf einem Experimentierboard zu koppeln. Außerdem kann auf diesen Port

[1] Speicherbausteine werden ausführlich in Kapitel II.2 beschrieben.

über die 64-polige Steckerleiste zugegriffen werden. Mit Hilfe einer speziellen Schnittstellen-Erweiterungskarte[2], die dem Praktikumsrechner beigefügt ist, können Sie einfache Aufgaben zur Portsteuerung lösen. Eine V24-Schnittstelle erlaubt den Anschluß eines PCs, eines Terminals, eines Druckers oder anderer Peripheriegeräte[3]. Der als *Timer* bezeichnete Baustein dient zur Erzeugung beliebiger Zeitfunktionen oder als Zähler. (Er wird in Kapitel 5 ausführlich beschrieben.)

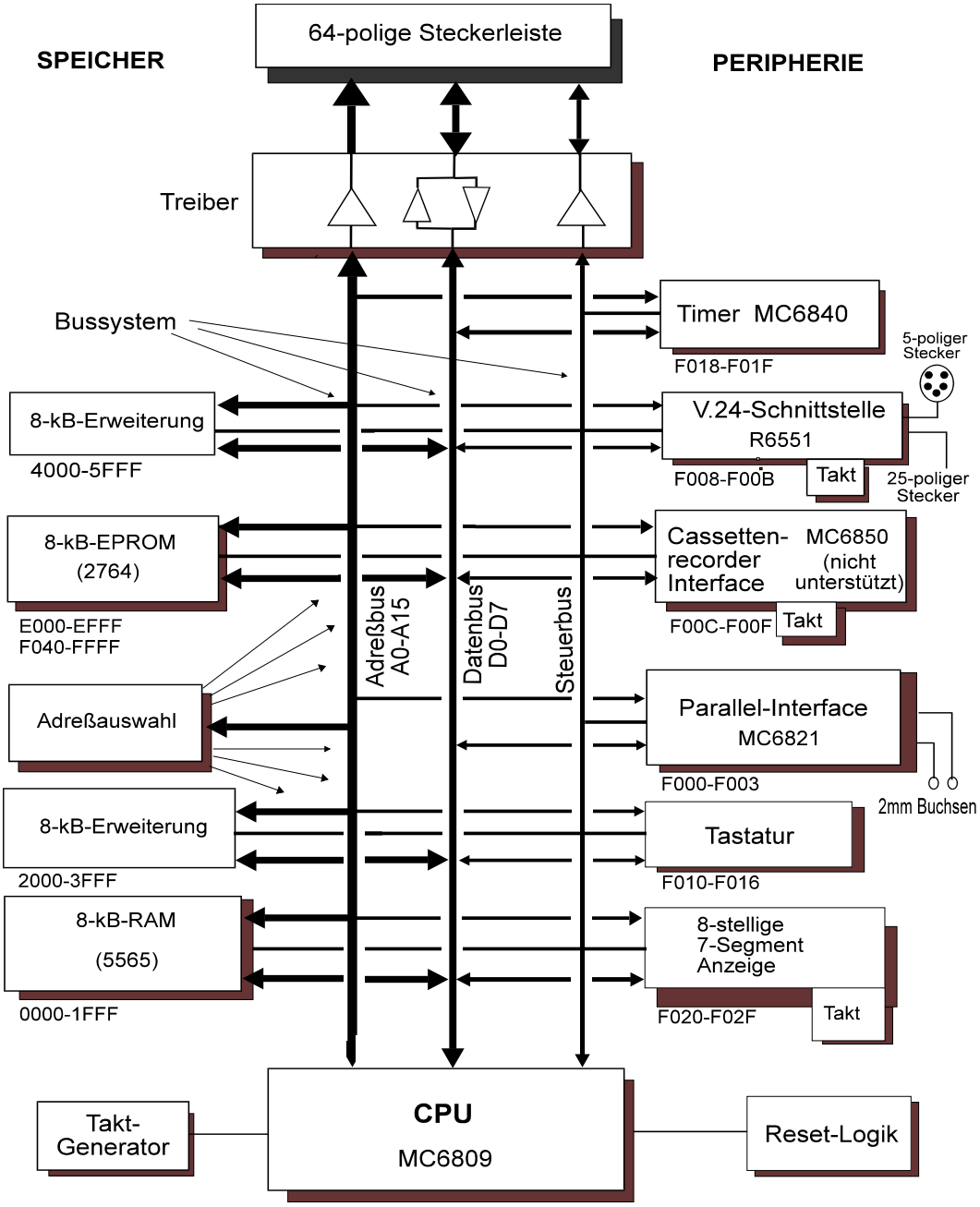


Bild 3.1-1: Blockschaltbild des Praktikumsrechners

[2] Diese Karte wird in Kapitel 4 genau beschrieben.
[3] Wie in Kapitel 2 beschrieben, unterstützt die Betriebssoftware bereits den Anschluß eines PCs oder eines Terminals.

Das Bussystem und die Adreßauswahl-Logik

Bild 3.1-1 kann man entnehmen, daß das Bussystem für den Datenaustausch zwischen den Komponenten des Mikrorechners zuständig ist. Dieses Bussystem besteht aus den Teilbussen Adreßbus, Datenbus und Steuerbus.

Der Adreßbus AB besteht aus 16 Leitungen (A0-A15), auf denen die vom Mikroprozessor erzeugten Adreßsignale parallel ausgegeben werden. Da diese Signale nur in einer Richtung verlaufen, wird der Adreßbus als unidirektional bezeichnet. Die Adressen sorgen über die weiter unten beschriebene Auswahlschaltung (Adreßdecoder) für die eindeutige Selektion einer Komponente, eines darin enthaltenden Registers oder einer Speicherzelle. Über die 16 Leitungen A0-A15 können 2^{16}=65536 verschiedene Adressen ausgegeben werden.

Der Datenbus DB besteht aus den 8 bidirektionalen Leitungen D0-D7. Auf ihm werden die Daten byteweise in beiden Richtungen übertragen, also vom Prozessor zu den Komponenten oder von diesen zum Prozessor.

Der Steuerbus CB (C*ontrol Bus*) enthält die Signale $R / \overline{W}$, E, Q, IRQ, $\overline{\mathrm{NMI}}$ und $\overline{\mathrm{RESET}}$, die Sie bereits bei der Beschreibung des Prozessors im Abschnitt 1.3 kennengelernt haben. Die beiden Taktsignale E und Q kennzeichnen das Bussystem als synchron, da alle Übertragungen in einem starren Zeitraster vorgenommen werden, wodurch also die Länge der einzelnen Übertragungsperioden und die Zeitpunkte der Komponentenauswahl und der Datenübergabe fest vorgegeben sind. Die Empfängerkomponente eines Datums gibt keinerlei Quittungen über den fehlerfreien oder fehlerhaften Empfang.

Ankopplung der Komponenten an den Bus[4]

Alle Komponenten des Mikrorechners hängen physikalisch an den oben beschriebenen Busleitungen und sind somit direkt miteinander verkoppelt. Nun kann man jedoch nicht ohne weiteres die Ausgänge von TTL- oder MOS-Schaltungen parallel miteinander verbinden. Dies würde bei zwei Schaltungen mit entgegengesetztem Ausgangspegel (H- bzw. L-Pegel) zu einem Kurzschluß führen. Deshalb sind für die Ausgänge der Komponenten spezielle elektrische Ankoppelschaltungen nötig. Eine Realisierungsmöglichkeit bietet die sogenannte Tristate-Logik. Der Name deutet schon auf einen dritten Zustand hin. Dieser Zustand ist der *High-Impedance*-Zustand, in dem der Ausgang hochohmig gegen beide Betriebsspannungen +5V und Masse (*GND*) ist. Damit hat der hochohmige Zustand die gleiche Wirkung wie die Durchtrennung der Ausgangsleitung. Bild 3.1-2 zeigt als Beispiel das vereinfachte Schaltbild eines TTL-Inverters (Transistor-Transistor-Logik) in Tristate-Technik.

Der Tristate-Inverter entspricht offensichlich einer NAND-Schaltung mit einem "zweckentfremdeten" Eingang, dem Steuereingang C (*Control*). Liegt dieser Eingang auf H-Potential, so hängt der Zustand des Transistors T1 nur vom Eingang E ab und die Diode D ist gesperrt. Also wirkt die Schaltung wie ein "normaler" Inverter (7404).

[4] Diesen Unterabschnitt sowie die Selbsttestaufgabe S3.1-1 können Sie überspringen, wenn Sie nicht an den elektronischen Hintergründen zum Praktikumsrechner interessiert sind.

Wird der Eingang C auf L-Potential gelegt, so wird über die Diode D die Basis des Ausgangstransistors T2 auf ein niedriges Potential heruntergezogen. Daher sperrt dieser Transistor, und der Ausgang A wird hochohmig gegen +5V. Gleichzeitig sorgt die NAND-Verknüpfung der Eingänge E und C dafür, daß der Transistor T3 und damit auch der Ausgangstransistor T4 sperren, der Ausgang A also auch hochohmig gegen Masse ist. In diesem Zustand hat der Eingang E keinerlei Wirkung.

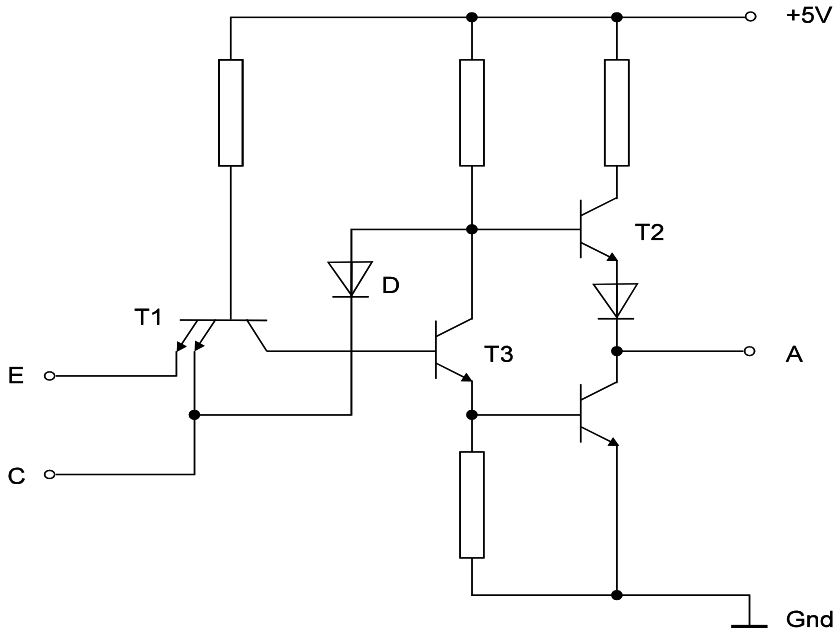
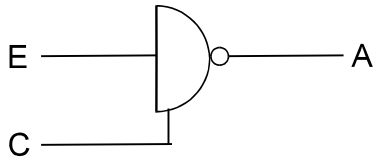


Bild 3.1-2: TTL-Inverter in Tristate-Logik

Selbsttestaufgabe S3.1-1:[5]

Skizzieren Sie eine Busleitung, an der 4 TTL-Tristate-Inverter angeschlossen sind. Benutzen Sie dabei das Schaltsymbol



Entwerfen Sie nun für die vier Steuereingänge C3-C0 eine Auswahlschaltung mit zwei "Adreßeingängen" A1,A0, die dafür sorgt, daß zu jedem Zeitpunkt genau ein Inverter aktiviert ist!

[5] Diese Selbsttestaufgabe können Sie überspringen, wenn Sie an schaltungstechnischen Details nicht interessiert sind.

Buszuteilung

In der Regel entstehen Konflikte, wenn mehrere aktive Einheiten, insbesondere mehrere Prozessoren, gleichzeitig auf einen Bus zugreifen wollen ("Buskonflikte"). Durch eine besondere Steuerlogik ("Schiedsrichter", *Busarbiter*) muß dann der Zugriff zum Bus geregelt werden[6]. Im vorliegenden Mikrorechner greift jedoch nur der µP MC6809 aktiv auf den Bus zu. Dieser sorgt durch die ausgegebene Adresse für eine konfliktfreie Zuteilung des Bussystems an die einzelnen Komponenten. Prinzipiell wäre es nun möglich, alle 16 Adreßleitungen *jeder* Komponente zuzuführen und dort die Entscheidung über die Zugriffsberechtigung zu fällen. Dies würde jedoch eine unnötige, technisch schwer zu realisierende Vergrößerung der Zahl der Anschlüsse an den Schaltkreisen und eine aufwendige, starre Zuteilungslogik in den Schaltkreisen bedeuten. Deshalb benutzt man in der Regel eine zentrale Zuteilungslogik, die aus den Adreßsignalen verschiedene Adreßbereiche selektiert und diese den einzelnen Komponenten zuweist ("Adreßauswahl-Logik", "Adreßdecoder"). Diese Logik wird bei der Beschreibung eines Systems häufig sehr knapp dargestellt, obwohl sie einen großen Teil der Platinenfläche und der benutzten Schaltkreise benötigt und einen nicht unerheblichen Entwurfs- und Optimierungsaufwand verursacht. Heutzutage werden Adreßdecoder gewöhnlich auch in integrierter Form, z.B. als PLA-Baustein (*Programmable Logical Array*) realisiert. Sehr häufig werden weitere Steuersignale zur Adreßauswahl herangezogen, z.B. das Schreib/Lese-Steuersignal $R/\overline{W}$, um zwischen Speicherzellen oder Registern zu unterscheiden, die nur beschrieben oder gelesen werden können. Der Einfachheit halber werden in der Regel jeder Komponente ein Bereich zugewiesen, dessen Adreßanzahl eine Potenz von 2 ist. Müssen nun in einer Komponente 2^i Register oder Speicherzellen selektiert werden können, so werden in der Regel die unteren i Adreßbits A0,..,A(i-1) dieser Komponente direkt zugeführt. Die restlichen Bits Ai,..,A15 werden im Adreßdecoder ausgewertet. Durch ein (oder wenige) Selektionssignal(e) (*Chip Select* - CS) wird dann die Komponente über das Vorliegen einer für sie gültigen Adresse informiert. Dazu besitzen die hochintegrierten Bausteine besondere "Aktivierungseingänge", die in der Regel mit CE (*Chip Enable*) bezeichnet sind.

Selbsttestaufgabe S3.1-2:[7]

Analysieren Sie im Gesamtschaltplan des Praktikumsrechners im Anhang C den Adreßdecoder, indem Sie für seine Ausgangssignale $\overline{CS0}$ - $\overline{CS13}$ die logischen Funktionen aufstellen und die durch sie selektierten Adreßbereiche angeben. Beginnen Sie der Einfachheit halber mit der Bestimmung des "Hilfssignals" $\overline{CS}$ (Ausgang von IC26). Gehen Sie außerdem davon aus, daß keine zusätzlichen Brücken gesetzt sind (vgl. IC35). Die Datenblätter zu den Schaltkreisen 74138 und 74139 finden Sie in der Datenblattsammlung zum Praktikum.

6 In Abschnitt I.2.6 werden Busarbiter ausführlicher beschrieben.

7 Diese Selbsttestaufgabe können Sie überspringen, wenn Sie nicht an den schaltungstechnischen Details des Praktikumsrechners interessiert sind.

Erweiterungsschnittstelle

Schon Bild 3.1-1 zeigt, daß der Adreßraum des Praktikumsrechners nicht voll belegt ist. Für externe Erweiterungen des Rechners durch zusätzliche Komponenten ist das Bussystem über eine 64-polige Steckerleiste an der Rückwand des Gehäuses zugänglich[8]. Zum Schutz der Schaltkreise, die im Rechner am Bussystem hängen, aber auch zur Erhöhung der Ausgangsleistung, sind alle Bussignale über Treiberschaltungen geführt. Diese sind beim Datenbus naturgemäß bidirektional, bei allen anderen Signalen unidirektional ausgelegt.

Adreß- und Steuerbustreiber sind permanent aktiviert, so daß die entsprechenden Signale jederzeit am Stecker beobachtet werden können. Die Aktivierung der bidirektionalen Datenbus-Treiber geschieht demgegenüber durch ein spezielles, extern zuzuführendes Signal $\overline{G}$ (s. Beschreibung der Steckerbelegung im Anhang B). Zur Erweiterung des Systems muß dieses Signal durch eine externe Schaltung so erzeugt werden, daß dadurch ein Adreßbereich selektiert wird, der sich mit dem intern belegten Adreßraum nicht überschneidet.

Selbsttestaufgabe S3.1-3:[9]
1. Machen Sie sich anhand der Datenblätter im Anhang mit der Funktionsweise der benutzten Treiberbausteine 74245 und 74541 vertraut.
2. Skizzieren Sie den externen Anschluß eines 1-kbyte-RAM-Bausteins an der Erweiterungsschnittstelle und entwerfen Sie dazu einen Adreßdecoder (zur Erzeugung des o.g. $\overline{G}$ Signals), der dieses RAM unter den Adressen $6000-$63FF ansprechen läßt.

Der Festwertspeicher

Damit der Mikrorechner nach dem Einschalten der Netzspannung sofort betriebsbereit ist, wurde sein Monitorprogramm in einem Festwertspeicher abgelegt. Im folgenden werden die physikalischen und technologischen Grundlagen der im Mikrorechner eingesetzten speziellen Festwertspeicher kurz dargestellt.

Dabei handelt es sich um ein EPROM (*Erasable Programmable Read Only Memory*), also um einen Festwertspeicher, der vom Benutzer durch ein spezielles Gerät programmiert und wieder "gelöscht" werden kann. EPROMs eignen sich wegen ihrer relativ hohen Kosten besonders für Anwendungen in kleinen Stückzahlen, für den Laboreinsatz und für die Entwicklungsphase eines Programms[10].

---

[8] Zur Erinnerung: Zusammen mit dem Praktikumsrechner wurde Ihnen eine eigenentwickelte „Schnittstellen-Erweiterungskarte“ zugeschickt, die wir in der Kapitel 4 ausführlich beschreiben werden und die Ihnen erlauben wird, Versuche mit den Schnittstellenbausteinen durchführen.

[9] Diese Selbsttestaufgabe können Sie überspringen, wenn Sie nicht an den schaltungstechnischen Details des Praktikumsrechners interessiert sind.

[10] Ihr Aufbau und ihre Funktion wird in Abschnitt II.2.3 ausführlich beschrieben.

In der folgenden Tabelle 3.1-1 beschreiben wir den im Praktikumsrechner eingesetzten EPROM-Baustein in Stichworten. (Sie sehen den Baustein unter der Plexiglas-abdeckung des Gerätes als 2. Baustein unten links. Im Anhang finden Sie das Datenblatt zum Baustein.)

Tabelle 3.1-1: Kenndaten des EPROM-Bausteins

Typ	: 2764 , von der Firma Intel entwickelt
Kapazität	: 8 kbyte = 8kx8 bit = 64 kbit
Gehäuse	: 28 Anschlußbeine (Pins) und Quarzfenster
Löschen	: je nach Leistung der UV-Lampe einige Minuten.
Programmierung	: byteweise, das Byte muß adressiert werden und dann für mindestens 50ms eine Spannung von 21V an den Programmiereingang V_{PP} des Bausteins gelegt werden (s.u.)
Pegel	: TTL-Pegel an allen Ein- und Ausgängen, Ausgänge in Tristate-Logik
Zugriffszeit	: maximal 250-300 ns, je nach Ausführung
Betriebsspannung	: +5V
Leistungsaufnahme	: 175 mW, falls Baustein nicht selektiert, 750 mW, falls Baustein selektiert.

Zum Abschluß dieses Unterabschnitts gibt Ihnen die folgende Tabelle 3.1-2 die augenblickliche Belegung des Festwertspeichers im Praktikumsrechner an.

Tabelle 3.1-2: Belegung des Festwertspeichers (Adressen in hexadezimaler Form)

Adreßbereich	Inhalt
FF00 - FFFF	Systemkonstanten, i.b. Interruptvektoren
F200 - ...	Hilfsroutinen (werden laufend ergänzt)
F100 - F1FF	Tabelle der Einsprungadressen der Hilfsroutinen nach Abschnitt 1.2.2
F030 - F0FF	frei
F000 - F02F	Adressen der Schnittstellen-Register
EF00	Tabelle der System-Startwerte (Zero-Page)
E800 - EEFF	frei
E000 - E7FF	Monitorroutinen (Tastatur-, Anzeigen- und Schnittstellen-Ansteuerung)

Der Schreib-/Lese-Speicher

Der eben besprochene Festwertspeicher dient zur Aufnahme von Programmen und Daten, die immer, insbesondere auch nach dem Einschalten des Gerätes, zur Verfügung stehen müssen. Daneben besitzt der Praktikumsrechner einen Schreib-/Lese-Speicher, der zur Speicherung von kurzfristig benötigten Daten und Programmen, also z.B. den von Ihnen zu schreibenden Programmen und Ihren Ergebnissen dient. [11] Dieser Schreiblesespeicher ist im Praktikumsrechner 8 kbyte groß. Er kann aber auf 16 kbyte vergrößert werden.[12]

Der RAM-Baustein belegt im Mikrorechner den untersten Adreßbereich $0000-$1FFF. Von diesem wird nur die unterste Seite (*Zero-Page*, $0000-$00FF) für die Systemvariablen reserviert. Alle anderen Speicherzellen stehen Ihnen zur freien Verfügung bereit. (Die drei folgenden 256-byte-Seiten von $0100 - $03FF) sind für zukünftige Erweiterungen des Monitorprogramms vorgesehen. Sie können bis dahin frei darüber verfügen.)

Wir wollen uns hier mit einer kurzen Auflistung der Kenndaten des im Praktikumsrechner eingesetzten RAM-Bausteins begnügen[13], die Sie in Tabelle 3.1-3 finden. (Im Anhang finden Sie auch das Datenblatt zum Baustein.)

Tabelle 3.1-3: Kenndaten des Schreib-/Lesespeicher-Bausteins

Typ:	TC5565Pl-15 der Firma Toshiba, statische CMOS-Zelle
Kapazität:	8 kbytes = 8kx8 bit = 64 kbit
Gehäuse:	28 Anschlußbeine
Zugriffszeit:	maximal 150-200 ns, je nach Ausführung
Pegel:	TTL-Pegel an allen Ein- und Ausgängen, Tristate-Ausgänge
Betriebsspannung:	+5V im "aktiven" Zustand, absenkbar auf +2V im Ruhezustand (*stand-by*)
Leistungsaufnahme:	35-40 mW aktiv, 10 µW stand-by

[11] Dieser Speicher wird im Englischen nicht ganz zutreffend mit *Random Access Memory* (RAM) bezeichnet, also als Speicher mit wahlfreiem Zugriff. Genau genommen ist natürlich auch ein EPROM in der eben beschriebenen Form ein solcher Speicher.

[12] Keine Angst: Für die von uns geforderten Programme reicht diese Speicherkapazität, ja sogar ein Bruchteil davon, vollständig aus.

[13] Eine ausführliche Beschreibung der Funktion und des Aufbaus von RAM-Bausteinen finden Sie in Abschnitt II.2.4.

Praktische Übung P3.1-1:

Der 6809-Simulator bietet Ihnen die Möglichkeit, in den (simulierten) Arbeitsspeicher des Praktikumsrechners hineinzuschauen, während ein Anwenderprogramm (oder das Monitorprogramm) läuft. Wählen Sie dazu unter dem Hauptmenü-Eintrag „Speicher“ den Eintrag „anzeigen“ an. Durch Eingabe einer neuen Startadresse in der ersten Tabellenspalte können Sie nun jeden beliebigen Adreßbereich betrachten und durch den Verschiebebalken verändern. Im RAM-Bereich können Sie jede Speicherzelle manipulieren und so insbesondere auch Maschinenprogramme eingeben. (Vergleiche Kapitel 2.)

1. Betrachten Sie den Adreßbereich $0000-$00FF, in dem das Monitorprogramm seine Systemvariablen verwaltet. Betätigen Sie die Funktionstaste C (*Clear*) und beachten Sie den Aufbau der Systemvariablen durch das Monitorprogramm. Betätigen Sie danach auch andere Funktionstasten.

2. Betrachten Sie den Ablauf im Systemstack (Adressen: .... - $1FFF) bei der Abarbeitung des folgenden Programms:

```
        ORG   $0400
HP:     LDD   #$1020
        CLR   $00
L:      INCA
        PSHS  A
        JSR   SUB1
        PULS  A
        BRA   L
```

```
        ORG   $0500
SUB1:   INCB
        PSHS  B
        JSR   SUB2
        PULS  B
        RTS

        ORG   $0700
SUB3:   LDY   #$0040
        JSR   DLY1MS
        RTS
```

```
        ORG   $0600
SUB2:   INC   $00
        LDA   $00
        PSHS  A
        JSR   SUB3
        PULS  A
        RTS
```

Analysieren Sie das Programm und interpretieren Sie die ersten Bytes des Stacks ! (s. Lösungsvorschlag)

3. Versuchen Sie zunächst im Simulator den Adreßbereich ab $2000 bzw. $4000 zu lesen und zu verändern.

Durch wenige Mausclicks können Sie nun den Arbeitsspeicher des simulierten Praktikumsrechners um 8 oder 16 kbyte vergrößern. Wählen Sie danach im Haupt-Menü den Eintrag „Datei“ und im Submenü „RAM/ROM Erw.“ an. Versuchen Sie nun noch einmal, den erweiterten Speicherbereich ab $2000 oder $4000 zu beschreiben.

3.1.2 Aufbau und Funktion der Anzeige

Bei den meisten Einplatinen-Mikrocomputern wurden in der Vergangenheit als Anzeigekomponente mehrstellige 7-Segmentanzeigen eingesetzt, die im Multiplexbetrieb angesteuert wurden, d.h., daß jede einzelne Stelle jeweils nur für einen kleinen Bruchteil der Zeit selektiert wird und aufleuchtet.[14] Durch eine genügend hohe Frequenz kann dabei vermieden werden, daß das menschliche Auge das Flackern der Anzeige wahrnehmen kann. Das Multiplexen der Anzeige wurde meistens vom Prozessor selbst vorgenommen.

Hauptziel beim Entwurf der Anzeigeeinheit unseres Mikrorechners war es, den Prozessor von dieser Aufgabe zu entlasten und dafür zu sorgen, daß die 8 Stellen der Anzeige für den Prozessor wie gewöhnliche Speicherstellen erscheinen, in die er lediglich die darzustellende Information übertragen muß. Das Multiplexen wird durch eine Zusatzschaltung hardwaremäßig geleistet. Das Bild 3.1-3 zeigt das Blockschaltbild der Anzeigeeinheit.

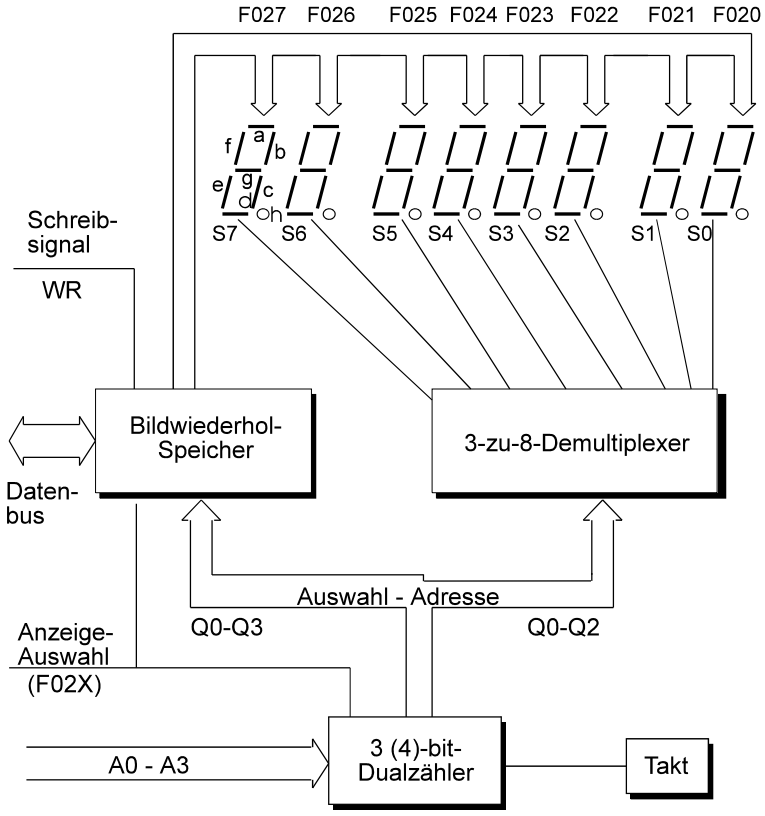


Bild 3.1-3: Blockschaltbild der Anzeigeeinheit

[14] Heute verzichtet man bei den modernen Einplatinen-Mikrocomputern meist auf eine eigene Anzeigeeinheit (und Tastatur). Ersatzweise stattet man diese Computer mit einer V.24-Schnittstelle und der notwenigen Software zur Kommunikation mit einem PC aus.

Das vom Adreßdecoder erzeugte Signal $\overline{CS13}$ (s. Abschnitt 3.1.1) dient zur Aktivierung der gesamten Anzeigeeinheit. Es selektiert die Adressen $F02X (X=0,..,F). Dabei ist zu beobachten, daß jeder Anzeigestelle zwei Adressen zugeordnet sind:

1. Stelle S0 (rechts): $F020 und $F028,
2. Stelle S1: $F021 und $F029,
   ....
8. Stelle S7 (links): $F027 und $F02F

Jede Stelle zeigt abwechselnd die beiden entsprechenden Worte des Bildwiederholspeicher an, der Betrachter sieht natürlich beide Informationen gleichzeitig. Wenn ein Speicherwort gleich null und das andere ungleich null ist, erfolgt die Anzeige in halber Helligkeit. Volle Helligkeit wird erreicht, wenn beide Worte gleich sind (und ungleich Null).

Die Adreßsignale A0-A2 dienen zur Auswahl genau einer Stelle der Anzeige. Wenn eine Adresse aus dem eben erwähnten Adreßbereich auf dem Adreßbus erscheint, werden die Adreßsignale A0-A3 parallel in einen 4-bit-Dualzähler geladen. Die Ausgänge Q0-Q3 dieses Zählers selektieren die entsprechende Speicherstelle in einem (2·8)x8-bit-Bildwiederholspeicher, dessen Eingänge am Datenbus des Systems angeschlossen sind. Durch das Ausgangssignal Q3 wird genau einer der beiden 8x8-bit-Speicherbereiche aktiviert. Durch das Schreibsignal WR wird die Übernahme der Daten in den Speicher erzwungen. Die Ausgänge des Bildwiederholspeichers sind mit den Anoden der Leuchtdioden in den 7-Segmentanzeigen verbunden. Diese Anoden sind im Bild 3.1-3 mit den Buchstaben a,b,c,..,h bezeichnet. Gleichbezeichnete Anoden aller Anzeigestellen sind miteinander verbunden. Dabei besteht folgende Beziehung zwischen den Datenbits D0,..,D7 und den Anoden der Anzeige (vgl. Abschnitt 1.2.1.):

D0 ★ a, D1 ★ b, ..., D7 ★ h.

Selbsttestaufgabe S3.1-4:[15]

Machen Sie sich anhand des Gesamtplans des Systems sowie des Datenblattes des verwendeten Bausteins 74289 im Anhang mit dem genauen Aufbau des Bildwiederholspeichers vertraut und beantworten Sie dazu die folgenden Fragen**:**

1. Wie müssen die Eingänge des Speichers beschaltet werden, um die hexadezimale Datenbusinformation $A7 in die Speicherstelle S7 einzuschreiben?
2. Welchen Wert erhält man an den Ausgängen, wenn man danach die 7. Speicherstelle liest?
3. Durch welchen Baustein des Systems wird die in den beiden eben behandelten Punkten festgestellte Diskrepanz aufgehoben?

[15] Diese Selbsttestaufgabe können Sie überspringen, wenn Sie nicht an den schaltungstechnischen Details des Praktikumsrechners interessiert sind.

Wird die Anzeigeeinheit nicht vom Prozessor angesprochen, so übernimmt der 4-bit-Dualzähler das Multiplexen der Anzeigestellen. Dazu ist er mit einem Taktgenerator ausgerüstet, der ihn zyklisch frei umlaufen läßt (Taktfrequenz: 1 kHz). Der aktuelle Zählerstand wird einerseits zur Adressierung der gerade darzustellenden Speicherstelle, andererseits zur Auswahl der zugehörenden Stelle der Anzeige herangezogen. Diese Auswahl geschieht wiederum durch einen 3-auf-8-Demultiplexer, dessen Ausgänge jeweils die gemeinsamen Kathoden je einer 7-Segment-Anzeige steuern (aktiv L-Pegel). Der Demultiplexer ist nicht mit dem höchstwertigen Bit des Dualzählers verbunden. Dadurch wird jede Anzeigestelle zweimal während eines Zählzyklus selektiert.

Praktische Übung P3.1-2:

Schreiben Sie ein Programm, das die als 8-stellige Hexadezimalzahl aufgefaßten Speicherstellen $F020-$F027 der Anzeige vom Wert 0 beginnend binär hochzählt. Der Zähltakt betrage ca. 1 kHz. Beachten Sie dabei, daß Sie diese Speicherstellen zwar beschreiben, nicht aber lesen können.

3.1.3 Aufbau und Funktion der Tastatur

Im Abschnitt 1.2.1 wurde bereits die Tastatur mit ihren Daten- und Funktionstasten ausführlich beschrieben. In diesem Abschnitt folgt nun die Beschreibung ihrer Ansteuerschaltung[16].

Dazu zeigt Bild 3.1-4 zunächst das Blockschaltbild. Die Tasten sind in Form einer zweidimensionalen Matrix angeordnet. Die Spalten dieser Matrix werden durch die Ausgänge eines 3-auf-8-Demultiplexers angesprochen. Die Auswahl einer bestimmten Spalte geschieht durch die Adreßsignale A0-A2. Der jeweils ausgewählte Ausgang nimmt den L-Pegel, d.h. logischen 0-Zustand an. Die Zeilen der Matrix werden über invertierende Tristate-Treiber auf die unteren vier Bits D0-D3 des Datenbusses gelegt. Sobald in einer angesprochenen, aktivierten Spalte eine Taste gedrückt wird, wird die zugehörende Zeile "kurzgeschlossen". Die drei anderen Zeileneingänge tragen (in der Regel, d.h. wenn in dieser Spalte nicht eine weitere Taste gedrückt ist,) den logischen Wert 1. Aus den eingelesenen Datenbits D0-D3 und der Adreßinformation A0-A2 kann der Prozessor die ausgewählte Taste eindeutig identifizieren. Die gesamte Ansteuerschaltung der Tastatur wird durch das Signal $\overline{CS12}$ des Adreßdecoders aktiviert (s. Abschnitt 2.2), das dazu auf die Steuereingänge des Demultiplexers und der Treiber gegeben wird. Zur Aktivierung des Demultiplexers (*enable*, aktiv L-Pegel) wird zusätzlich das Adreßsignal A3 benutzt.

[16] Diese Beschreibung können Sie wieder überspringen, wenn Sie an schaltungstechnischen Details nicht interessiert sind. Bitte lesen Sie in diesem Fall weiter ab der Beschreibung von Tabelle 3.1-5.

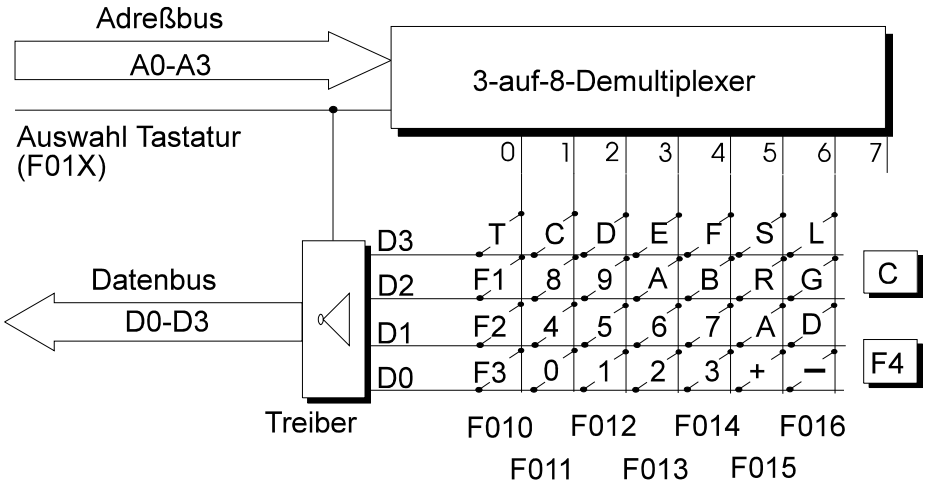


Bild 3.1-4: Die Ansteuerschaltung der Tastatur

Da durch das Auswahlsignal der Tastatur (Signal $\overline{CS12}$) die Speicheradressen $F01X angesprochen werden (X=0,..,F), ergeben sich die in der Tabelle 3.1-4 angegebenen Adreß- und Datenbuswerte für die einzelnen Tasten. (Diese Werte sind alle hexadezimal aufzufassen. X bedeutet wiederum „beliebig“ (*don't care*).)

Tabelle 3.1-4: Adreß- und Datenbusinformation zur Dekodierung der Tasten)

Datentasten			**Funktionstasten**		
Taste	Adresse	Datenbus	Taste	Adresse	Datenbus
0	F011	X1	+	F015	X1
1	F012	X1	-	F016	X1
2	F013	X1	A	F015	X2
3	F014	X1	D	F016	X2
4	F011	X2	R	F015	X4
5	F012	X2	G	F016	X4
6	F013	X2	S	F015	X8
7	F014	X2	L	F016	X8
8	F011	X4	T	F010	X8
9	F012	X4	F1	F010	X4
A	F013	X4	F2	F010	X2
B	F014	X4	F3	F010	X1
C	F011	X8			
D	F012	X8			
E	F013	X8			
F	F014	X8			

Um Ihnen das Arbeiten mit der Tastatur zu erleichtern, sind in der Tabelle 3.1-5 die Tasteninformationen als Matrix dargestellt. (Adressen und Datenbuswerte sind hexadezimal aufzufassen. Das Symbol X bezeichnet wieder einen beliebigen Wert aus 0,..,F.)

Tabelle 3.1-5: Matrix zur Dekodierung der Tasten

	Adresse							**Datenbus**
	F010	F011	F012	F013	F014	F015	F016	
Tasten-betätigung	F0	C	D	E	F	S	L	X8
	F1	8	9	A	B	R	G	X4
	F2	4	5	6	7	A	D	X2
	F3	0	1	2	3	+	-	X1

Selbsttestaufgabe S3.1-5:

Begründen Sie anhand des Gesamtplans des Praktikumsrechners im Anhang B, warum die Rücksetztaste C sowie die Break-Taste F4 nicht in der oben dargestellten Ansteuerschaltung der Tastatur beschrieben wurden !

Selbsttestaufgabe S3.1-6:[17]

Machen Sie sich anhand des Gesamtplans des Praktikumsrechners und des Datenblattes des Demultiplexers 74138 im Anhang A mit der Funktion dieses Bausteins vertraut und beantworten Sie dazu die folgende Frage**:**

1. Welcher Ausgang (Y0,..,Y7) ist aktiviert, wenn auf dem Adreßbus die Adresse $F017 ausgegeben wird, und welche Wirkung hat das auf die Tastatur ?
2. Welche Adresse muß der Prozessor ausgeben, um die 3. Spalte (der Tastatur nach Bild 3.1-4) zu selektieren, und welcher Hexadezimalwert wird auf dem Datenbus eingelesen, wenn Sie in dieser Spalte gleichzeitig die Tasten ‘5’ und ‘9’ drücken ?

Praktische Übung P3.1-3:

Schreiben Sie ein Programm, das zuerst die Anzeige löscht und dann zyklisch alle Spalten der Tastaturmatrix selektiert und die Zeilen der Matrix den Segmenten a bis d der rechten Anzeigestelle (S0) zuordnet. Dabei soll gelten: Wenn in einer Zeile mindestens eine Taste gedrückt ist, wird das zugeordnete Segment eingeschaltet.

[17] Diese Selbsttestaufgabe können Sie wieder überspringen, wenn Sie an schaltungstechnischen Details nicht interessiert sind.

3.2 Das Monitorprogramm

3.2.1 Allgemeines

Ein Rechensystem, dessen Arbeitsspeicher ausschließlich aus einem flüchtigen Schreib-/Lese-Speicher besteht, dessen Inhalt demnach beim Abschalten der Versorgungsspannung verloren geht, ist nur bedingt funktionsfähig: Da der Inhalt des Arbeitsspeichers beim Wiedereinschalten nicht definiert ist, ist der Prozessor nicht in der Lage, sinnvolle Operationen auszuführen. Es sind nicht einmal Grundfunktionen wie das Lesen der Tastatur möglich.

In der Frühzeit der Computer hat man sich damit beholfen, den Prozessor zunächst einmal - durch Sperren des Taktsignals - anzuhalten, um dann ein Programm unter Umgehung des Prozessors direkt in den Speicher zu schreiben.

Auch der in den Anfangsjahren unseres Praktikums eingesetzte Praktikumsrechner wurde nach diesem Prinzip betrieben: Zuerst wurde ein kurzes Hilfsprogramm eingegeben, indem man mittels Tastatur zunächst die Adresse auswählte und dann den zugehörigen Befehl in binärer Form in den Speicher schrieb. Dieses Programm führte dann automatisch die Inkrementierung der Adresse durch, und man konnte sich für die Eingabe des eigentlichen Programms auf die Eingabe des Befehls - natürlich immer noch in binärer Form - beschränken.

Stürzte das zu testende Programm dann ab, was, wie Sie mittlerweile wahrscheinlich wissen, nicht die Ausnahme, sondern die Regel ist, so war im allgemeinen auch das mühsam eingetippte Hilfsprogramm zerstört, und man mußte wieder völlig von vorn anfangen !

Wie Sie aus dieser kurzen Darstellung schon ersehen können, ist diese Methode der Programmierung nicht nur sehr fehleranfällig, sondern auch äußerst mühselig und zeitaufwendig. Daher ist die Forderung nach einem Programmsystem verständlich, das bestimmte Grundfunktionen des Rechners schon unmittelbar nach der Inbetriebnahme bereitstellt.

Bei diesem System muß es sich nicht unbedingt um ein Monitorprogramm oder ein komplettes Betriebssystem handeln. Insbesondere bei größeren Rechnersystemen, die auf jeden Fall mit externen Speichermedien wie Platten- oder Diskettenlaufwerken oder Bandmaschinen verbunden sind, existiert oft nur ein sogenannter Urlader (*Bootstrap Loader*), der den Rechner in den Stand setzt, ein Betriebssystem "von außen" in den Arbeitsspeicher zu laden und zu starten.

Bei einem kleineren System, wie dem von Ihnen benutzten Praktikumsrechner, der auch ohne angeschlossene Peripheriegeräte betriebsfähig sein soll, ist es dagegen günstiger, zumindest ein einfaches Betriebssystem in einem Festwertspeicher mitzuliefern, das dem Benutzer sofort nach dem Einschalten zur Verfügung steht und bei der Erstellung von Programmen, bei der Fehlerbeseitigung und der Benutzung der verschiedenen Komponenten des Rechners Hilfestellung gibt.

Anforderungen an ein Monitorprogramm

Wie oben bereits erwähnt, muß ein Rechner nach dem Einschalten in einen definierten Grundzustand überführt werden. Dazu gehören insbesondere das Setzen des Stackpointers und die Initialisierung der verschiedenen Rechnerkomponenten, insbesondere der Schnittstellen. Diese Aufgaben müssen von einem "eingebauten" Monitorprogramm übernommen werden.

Die Hauptaufgabe eines Monitors besteht aber in der Unterstützung des Programmierers bei der Eingabe und beim Austesten von Programmen. Der angebotene Komfort ist dabei vorwiegend von dem für den Monitor zur Verfügung stehenden Speicherplatz abhängig.

Da die Entwicklungssysteme auf der Basis von Einplatinencomputern noch vor einigen Jahren aufgrund der hohen Preise für Halbleiterspeicher mit sehr kleinen Monitorprogrammen auskommen mußten, war die Zahl der verfügbaren Funktionen auf ein absolut notwendiges Minimum beschränkt, das die Programmierung des Prozessors in Maschinensprache mit Hilfe einer Hexadezimaltastatur erlaubte.

Mit dem Preisverfall der Halbleiterbauelemente wurden jedoch immer umfangreichere und damit komfortablere Programme möglich, so daß ein Monitor mittlerweile durchaus über einen einfachen Assembler/Disassembler und eine ganze Reihe weiterer komplexer Funktionen verfügen kann.

Der im Praktikumsrechner benutzte Monitor ist in einem EPROM vom Typ 2764 untergebracht. Dieser Baustein wurde schon im Abschnitt 3.1 kurz vorgestellt. Er besitzt eine Organisation von 8k·8 bit, also eine Kapazität von 8 kbyte. Davon sind zur Zeit knapp 4 kbyte vom Monitor belegt, die restlichen 4 kbyte sind für die schon in Kapitel 1 angesprochenen Hilfsroutinen reserviert.[1]

Für ein reines Entwicklungssystem, auf dem hauptsächlich Programme für andere Rechner getestet und zum Laufen gebracht werden sollen, sind diese Routinen relativ uninteressant. Sollen die fertigen Programme aber auf demselben System ausgeführt werden, auf dem sie entwickelt wurden, ergibt sich durch eine solche Bibliothek eine bedeutende Einsparung an Speicherplatz, vor allem aber eine große Zeitersparnis bei der Programmierung. Bei geschickter Anwendung der Bibliotheksprogramme kann ein Anwenderprogramm dann zu einem großen Teil aus Unterprogrammsprüngen zu den Hilfsroutinen bestehen.

Selbsttestaufgabe S3.2-1:
Überlegen Sie sich, warum die Routinen der Bibliothek nicht in einem Programm benutzt werden können, das auf einem anderen Rechner ausgeführt werden soll.

[1] Zu Ihrer Erinnerung: Es ist noch genug Platz für weitere nützliche Routinen vorhanden. Falls Sie also eigene Ideen oder gar fertige Routinen haben, die den Teilnehmern die Arbeit mit dem µR vereinfachen können, teilen Sie uns das bitte mit. Wir werden Ihre (geeigneten) Anregungen gerne in der nächsten Version des Monitors berücksichtigen.

3.2.2 Aufbau und Flußdiagramm des Monitors

Der Aufbau und die Funktionsweise eines Monitors sollen hier beispielhaft anhand des im Praktikumsrechner benutzten Monitors erklärt werden. Allerdings werden wir bei einigen Funktionen, die unserer Meinung nach schlecht gelöst wurden, auf andere bzw. bessere Methoden zurückgreifen.

Zum besseren Verständnis sehen Sie in Bild 3.2-1 die Adreßraumbelegung des Rechners.

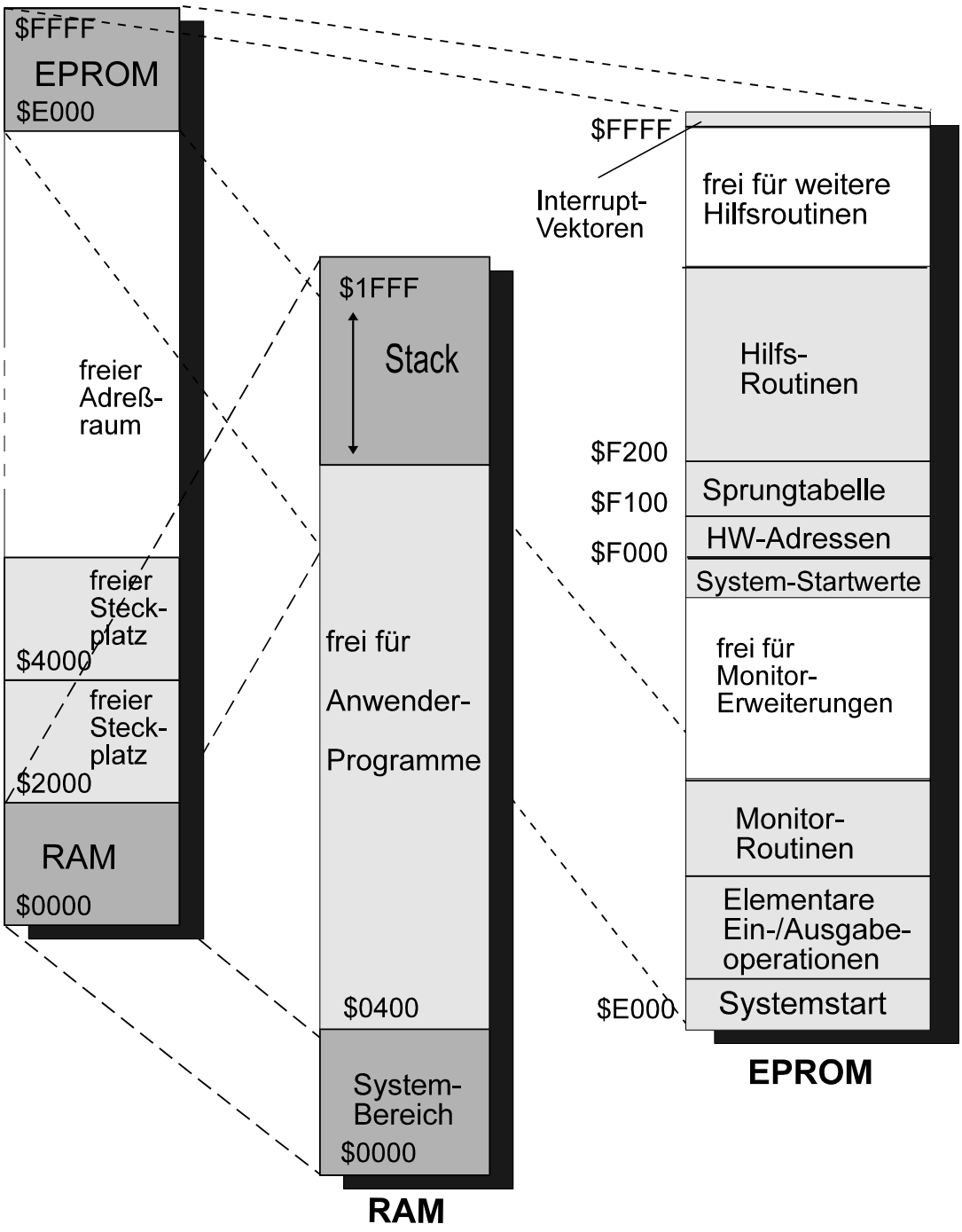


Bild 3.2-1: Die Adreßraumbelegung des Praktikumsrechners

Der gesamte adressierbare Bereich beträgt 64 kbyte. Davon werden 8 kbyte ($0000 - $1FFF) vom RAM belegt, weitere 8 kbyte vom EPROM ($E000 - $FFFF). Außerdem sind noch zwei Steckplätze für Speicherbausteine vorhanden, die aber in der Ihnen zugeschickten Grundausstattung des Praktikumsrechners nicht benutzt werden.

Das RAM enthält neben den Anwenderprogrammen die Systemvariablen, die in den ersten vier 256-byte-Seiten untergebracht sind, und den (System-)Stack. Da seine Größe statisch nicht feststeht, wird der Stackpointer zunächst auf das Ende des RAM-Bereichs gesetzt. Der Stack wächst dann rückwärts auf die Anwenderprogramme zu. Er benötigt in der Regel weniger als 200 byte[2], kann aber erheblich größer werden, falls ein Programm sehr stark verschachtelt ist oder die Parameter für Unterprogramme auf dem Stack übergeben werden.

Insbesondere Programmierfehler (z.B. das Vergessen, auf dem Stapel abgelegte Werte wieder herunterzunehmen) können dazu führen, daß der Stack in den Programmbereich "hineinwächst" und damit das Anwenderprogramm zerstört.

Das Flußdiagramm in Bild 3.2-2 zeigt in stark vereinfachter Form den allgemeinen Ablauf des Monitorprogramms.

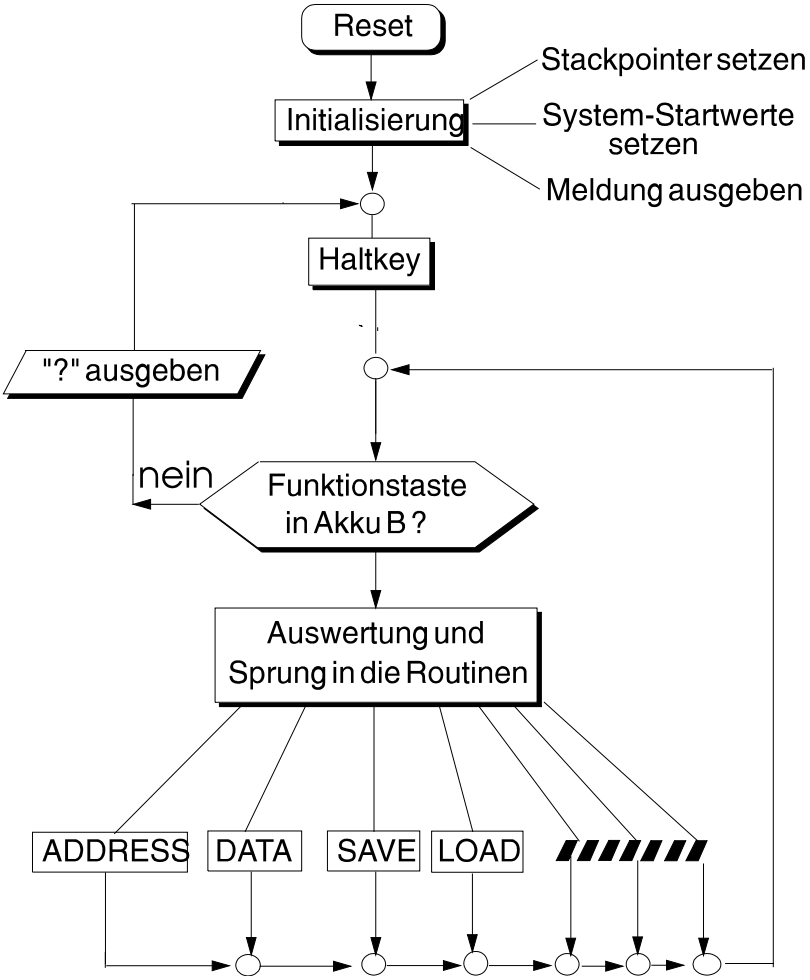


Bild 3.2-2: Das Flußdiagramm des Monitors

[2] Bei den Programmen, die wir von Ihnen verlangen, wird er in der Regel sehr viel weniger Platz beanspruchen.

Nach dem Rücksetzen, ausgelöst durch die Taste C bzw. automatisch beim Einschalten des Rechners (*Power on Reset*), springt der Rechner in die Initialisierungsroutine. Diese Routine kann sehr vielfältige Aufgaben haben. Dazu gehören z.B. Selbsttests, Feststellung des RAM-Bereichs und der angeschlossenen Geräte oder Initialisierung der Schnittstellen.

Zwei Funktionen müssen aber auf jeden Fall ausgeführt werden: Der Stackpointer S muß gesetzt werden und die vom Monitor benötigten Variablen müssen ihre Startwerte erhalten. Dazu wird ein Bereich des ROM in den Systembereich des RAM kopiert, wo die Werte dann vom Monitor benutzt und bei Bedarf verändert werden können.

Selbsttestaufgabe S3.2-2:
Der 6809-Prozessor sperrt alle Unterbrechungen, bis ein Wert in den Stackpointer S geschrieben wurde. Warum ?

Zum Abschluß der Initialisierung wird eine Meldung ausgegeben, die anzeigt, daß der Rechner jetzt betriebsbereit ist, und eventuell noch einige Informationen über den Zustand des Systems gibt. Die Meldung Ihres Rechners wurde schon in Kapitel 1, Abschnitt 1.2.1, erklärt.

Der **Kommandointerpreter** ist das Kernstück des Monitors. Hier werden die Kommandos von der Tastatur eingelesen, ausgewertet und die entsprechenden Monitorroutinen aktiviert.

Das Assemblerlisting der Interpreterschleife in Bild 3.2-3 entspricht nicht der augenblicklichen Implementierung im Praktikumsrechner. Nach ihrem Muster soll die Interpreterschleife aber in einer der nächsten Versionen in den Monitor eingebaut werden. Durch die spezielle Codierung der Tasten (siehe Tabelle 1.2-2 in Kapitel 1) wird sowohl die Auswertungsroutine als auch die Sprungtabelle sehr kurz.

Die Interpreterschleife springt nach der Ausführung einer Funktion nicht automatisch wieder in die Tastaturabfrage (*HALTKEY*), und zwar deshalb, weil in den Monitorroutinen in der Regel nach weiteren Eingaben gefragt wird. Beim Erkennen einer für die laufende Routine ungültigen Eingabe wird die Routine beendet. Nur wenn der Kommandointerpreter im Akku B keinen gültigen Befehl findet (*TESTB*), wird die Tastatur erneut abgefragt.

Der Pseudobefehl FDB im Listing bedeutet "*Form Double Byte*". Es handelt sich also nicht um einen ausführbaren Befehl, sondern um die Anweisung an den Assembler, an dieser Stelle einen 16-bit-Wert einzutragen, in diesem Fall die Startadressen der Monitorroutinen.

Falls die Sprungtabelle sich im RAM-Bereich des Rechners befindet, ist der Austausch der Routinen durch den Benutzer möglich. Für die Tasten F1 - F3 gilt dies auch im vorliegenden Monitor (siehe auch Kapitel 1). Ein konkretes Beispiel dazu finden Sie in Unterabschnitt 3.2.5.

```
; Interpreterschleife
;
FALSCH:     JSR   CLRDISP     ; Anzeige löschen
            LDA   #$D3        ; 7-Segmentcode für '?'
            LDX   #0          ;  in der letzten Stelle
            JSR   SHOWA       ; anzeigen
            JSR   HALTKEY     ; auf einen Befehl warten
SCHLEIFE:   TSTB  ; Code in Akku B testen
            BPL   FALSCH      ; keine Funktionstaste, zurück
            ANDB  #$7F        ; Bit 7 löschen
            LSLB              ; 2 Bytes pro Eintrag
            LDX   #SPRUNGTB   ; Basisadresse der Tabelle
            JSR   [B,X]       ; indirekter Sprung zur Routine
            BRA   SCHLEIFE    ; Endlos-Schleife
;
; In jeder Funktionsroutine: Fehlerbehandlung bei unzulässiger Funktionstaste
;
FEHLER:     CLRB              ; Tastencode löschen
            RTS               ; zur Interpreterschleife
;
; Sprungtabelle
;
SPRUNGTB:   FDB   #FEHLER     ; +     den Tasten '+' und '-' sind
            FDB   #FEHLER     ; -     keine Routinen zugeordnet
            FDB   #ADDRESS    ; A   |
            FDB   #DATA       ; D   |
            FDB   #REGIST     ; R   |  ansonsten werden die
            FDB   #GO         ; G   |
            FDB   #SAVE       ; S   |  jeweiligen Startadressen
            FDB   #LOAD       ; L   |
            FDB   #TRACE      ; T   |  der Routinen in die
            FDB   #INSERT     ; F1  |
            FDB   #DELETE     ; F2  |  Tabelle eingetragen
            FDB   #DUMP       ; F3  |
```

Bild 3.2-3: Listing des Kommandointerpreters

Praktische Übung P3.2-1:

Schreiben Sie eine Interpreterschleife, die Kommandos aus vier beliebigen Hexadezimalziffern (zwei Bytes) verwendet. Benutzen Sie zur Eingabe des Befehls die Routine SHOWADR.

3.2.3 Aufbau der Hilfsroutinen

Beim Betrieb eines Rechners werden einige Standardfunktionen immer wieder benötigt. Um sie nicht jedesmal neu programmieren zu müssen und damit Arbeitszeit und Speicherplatz zu verschwenden, wird von vornherein eine Reihe dieser Funktionen als komplette Unterprogramme in den Monitor eingebaut.

Einen Teil der Hilfsroutinen haben Sie bereits in Kapitel 1, Abschnitt 1.2.2, kennengelernt. Dazu gehören die Ein-/Ausgaberoutinen und die Umwandlungsroutinen, z.B. vom Hexadezimal- in den Siebensegment-Code.

Daneben werden von einigen Monitorroutinen noch Unterprogramme zum Editieren der Parameter und zur Adreß- bzw. Dateneingabe gebraucht.

Als einfaches Beispiel ist in Bild 3.2-4 die Routine *CLRDISP* abgebildet, die die eingebaute 7-Segment-Anzeige löscht:

```
;CLeaRDISPlay:      Siebensegmentanzeige löschen
;
CLRDISP:    PSHS  X,CC          ; CPU Register retten
            LDX    #DISPL       ; Display, rechte Stelle (S0)
CL1:        CLR    ,X+          ; löschen
            CMPX  #DISPL+16     ; bis DISPL+7 (S7)
            BNE    CL1          ;
            PULS   X,CC         ; Register restaurieren
            RTS                 ; Ende CLRDISP
;
DISPL       EQU   $F020         ; Startadresse der Anzeige, s. Abschnitt 3.1
```

Bild 3.2-4: Listing der Routine *CLRDISP*

Alle von der Routine benutzten Register werden zu Beginn auf den Stapel gerettet und vor der Beendigung wieder heruntergenommen, so daß der alte Zustand der CPU wiederhergestellt wird. Die Funktion der Routine ist einfach: Ein Register zeigt auf die erste Stelle der Anzeige. Dann wird in einer Schleife eine 0 in die entsprechende Adresse geschrieben und das Register inkrementiert, bis alle acht Stellen gelöscht sind. Durch den Pseudo-Opcode *EQU* wird der Variablen DISPL die Startadresse der Anzeige zugewiesen.[3]

Obwohl die Hilfsroutinen nur Standardfunktionen realisieren, müssen sie nicht unbedingt trivial sein. Das läßt sich recht eindrucksvoll an der Routine *SHOWADR* nachweisen. Sie liest eine Adresse (zwei Bytes) zyklisch von der Tastatur in das Y-Register und stellt diese im Adreßfeld der Anzeige dar, bis eine Funktionstaste gedrückt wird. Deren Tastencode steht dann im Akku B zur Verfügung.

SHOWADR bedient sich mehrfach der Hilfsroutinen *HALTKEY* und *SHOWD7SG*, die ihrerseits weitere Routinen aufrufen. Bild 3.2-5 dient zur Veranschaulichung des Ablaufs: Bei der Ausführung von *SHOWADR* werden insgesamt acht weitere Routinen aufgerufen, die maximale Schachtelungstiefe ist 5.

[3] Assemblieren Sie zur Übung die Routine CLRDISP und testen Sie sie aus !

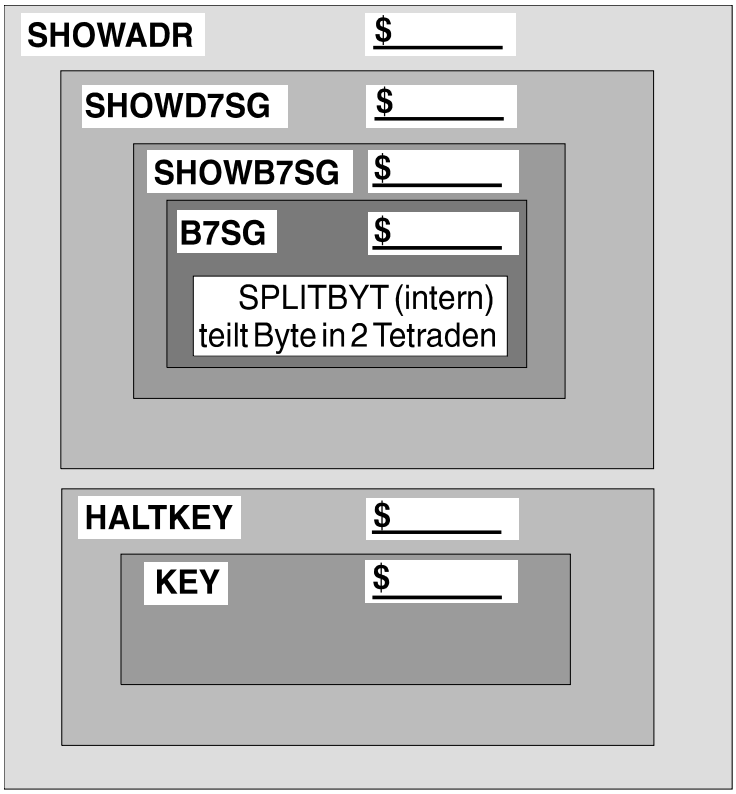


Bild 3.2-5: Beispiel zur Unterprogramm-Schachtelung

Aus dieser Sicht wird auch die Bedeutung der Zusicherung aus Kapitel 1, Abschnitt 1.2.2, klar, daß nur die Inhalte der explizit angegebenen Register verändert werden und alle anderen Daten unverändert bleiben. Anderenfalls wären die Auswirkungen eines solchen Funktionsaufrufs für den Programmierer kaum noch nachvollziehbar. Die einzige Möglichkeit, sich vor unliebsamen Überraschungen zu schützen, bestünde vor jedem Aufruf einer Hilfsroutine in der Rettung aller weiterhin benötigter Register auf den Stack und deren Restaurierung nach der Rückkehr.

Praktische Übung P3.2-2:

1. Lesen Sie die Routine *SHOWADR* aus dem EPROM und tragen Sie in das Diagramm von Bild 3.2-5 die absoluten Adressen ein, an denen die Hilfsroutinen aufgerufen werden. Achtung**:** Die Hilfsroutinen werden zum Teil mit relativen Sprüngen (LBSR) aufgerufen.
2. Analysieren Sie die Routine *INDATA*. Disassemblieren Sie das Programm, kommentieren Sie die Befehle und zeichnen Sie das Flußdiagramm.

3.2.4 Aufbau der integrierten Monitorroutinen

Abgesehen von der Hexadezimaltastatur gibt es auf dem Tastenfeld des Rechners vier Tasten, denen keine Routine zugeordnet ist: Die Funktionstaste C führt einen Systemstart durch und die Taste F4 gibt eine Interruptanforderung an den Prozessor. (Auf die Interruptbehandlung werden wir im Unterabschnitt 3.2.5 noch näher eingehen). Die Tasten '+' und '-' haben verschiedene Bedeutungen, auf die wir bei der Behandlung der einzelnen Monitorroutinen zurückkommen. Die übrigen Tasten rufen die verschiedenen Funktionen des Monitors auf, die bereits in Kapitel 1, Abschnitt 1.2.1, erklärt wurden und die sich grob in vier Klassen einteilen lassen:

A. Bearbeitung des Speichers:
   - Tasten A, D: Adressen und Daten editieren
   - Tasten F1, F2: Einfügen und Entfernen von Daten und Befehlen
   - Taste F3: Durchsehen eines Speicherbereichs

B. Bearbeitung der Prozessor-Register: Taste R

C. Laden und Sichern von Anwenderprogrammen: Tasten L, S

D. Ausführung der Anwenderprogramme:
   - Taste G: Starten eines Programms
   - Taste T: Einzelschrittausführung
   - Befehl SWI1[*)] : Unterbrechung bzw. kontrollierte Beendigung
   - Taste F4: Programmabbruch von außen

A. Speicherbearbeitung

Die Speicherbearbeitung wirft keine größeren Probleme auf. Es stehen geeignete Hilfsroutinen zur Verfügung, die den Großteil der auszuführenden Teilaufgaben übernehmen, so daß nur noch das Gerüst zusammengesetzt werden muß.

Die Tasten '+' und '-' haben bei diesen Routinen die Aufgabe, eingegebene Daten zu übernehmen und den Adressenzeiger zu inkrementieren bzw. dekrementieren.
Anhand der Routine *ADDRESS* demonstrieren wir Ihnen hier einmal die Implementierung einer Routine von der Problemstellung bis zum fertigen Programm mittels der Ihnen bekannten Hilfsroutinen.

Zunächst die Anforderungen:

1. Löschen der Anzeige und Ausgabe der Kennung "Ad" im Operationsfeld,
2. Ausgabe der zu editierenden Adresse und des zugehörigen Datums,
3. Eingabe einer neuen Adresse bzw. inkrementieren / dekrementieren der Adresse mit den Tasten '+' und '-',
4. Rückkehr aus der Routine, wenn eine andere Taste gedrückt wurde als eine Ziffer oder die Tasten '+' bzw. '-'.

[*)] In den Unterlagen und der Entwicklungssoftware zum Praktikum z.T. auch nur SWI genannt.

Aus den Anforderungen wird das Flußdiagramm nach Bild 3.2-6 entwickelt:

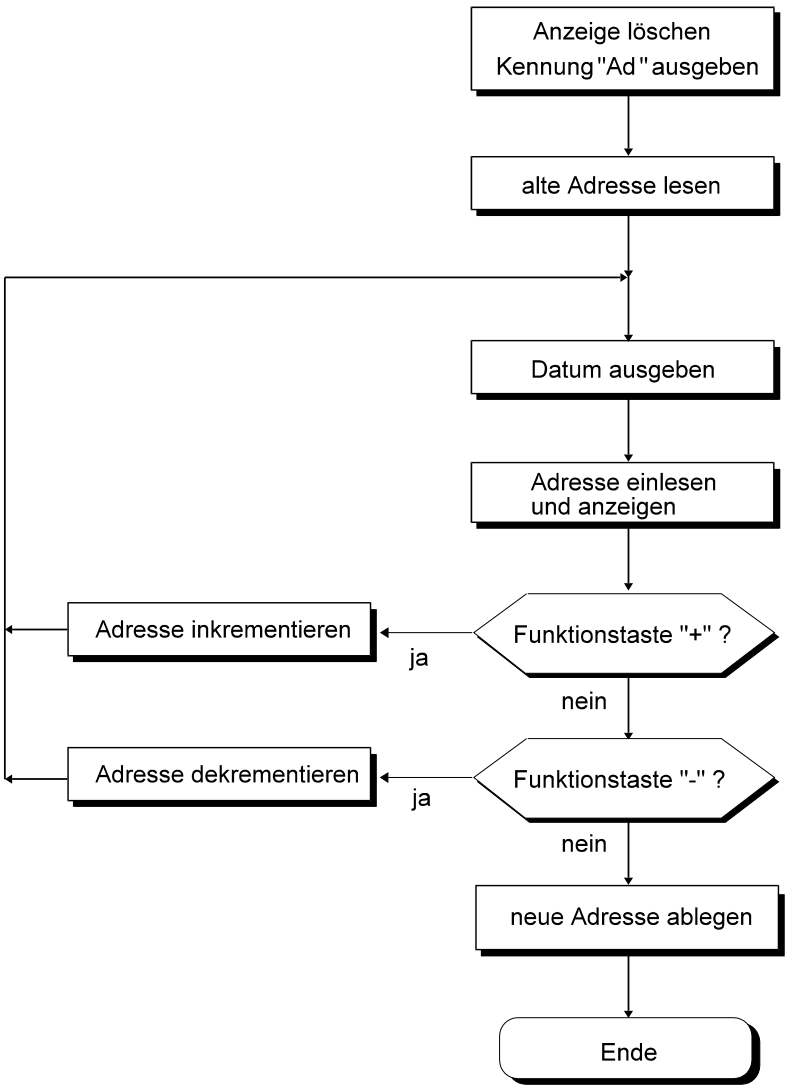


Bild 3.2-6: Flußdiagramm der Routine *ADDRESS*

Bild 3.2-7 zeigt die Routine in Assemblerschreibweise. Dabei ist bei der Benutzung der Routine *SHOWADR* zu beachten, daß das Y-Register im ersten Befehl gelöscht wird. Wenn Sie also nicht unbedingt eine neue Adresse eingeben möchten, springen Sie zum zweiten Befehl der Routine, was in diesem Fall ohne weiteres möglich ist, da *SHOWADR* keine Register auf den Stapel rettet. Da das Löschen des Y-Registers (LDY #0) vier Bytes beansprucht, muß eine neue Einsprungadresse berechnet werden.

Anmerkung: Sie können diese Routine anstelle der "Standard"-Routine benutzen, allerdings mit einigen Einschränkungen.

```
; Routine    ADDRESS:    Adresse editieren
;
; Ausgabe der Kennung und Holen der Adresse
;
ADDRESS: PSHS   A,X,Y,CC        ; Register retten
         JSR    CLRDISP         ; Anzeige löschen
         LDD    #$775E          ; 7-Segmentcodes für "Ad"
         LDX    #6              ; im Operationsfeld
         JSR    SHOWD           ; anzeigen
         LDY    <ADR            ; zu editierende Adresse holen
                                ; zum Zeichen "<" siehe das
                                ; Programmierhandbuch
; Adresse editieren
;
W:       LDB    ,Y ; zugehöriges Datum in Akku B
         LDX    #0              ; im Datenfeld
         JSR    SHOWB7SG        ; anzeigen
         LDX    #2              ; Anzeige im Adreßfeld
         LDU    SHOWADR+1       ; Startadresse aus Sprungtabelle
         JSR    4,U             ; Aufruf ohne Löschen von Y
;
; Ergebnis von SHOWADR interpretieren
;
         CMPB   #$81            ; letzte Taste
         BHI    R               ; >$81?, dann fertig
         BEQ    $2              ; =$81(Taste '-')?, dann dekrementieren
         LEAY   2,Y             ; Zeiger inkrementieren (um 2, da
                                ; wieder dekrementiert wird)
         LEAY   -1,Y            ; Zeiger dekrementieren
         BRA    W               ; und erneut editieren
;
; falls Ergebnis nicht "+" oder "-", geordnetes Beenden des Programms
;
R:       STY    <ADR            ; neue Adresse ablegen
         PULS   A,X,Y,CC        ; Register restaurieren
         RTS                    ; zurück zum Monitor
ADR      EQU    $....           ; 2 Speicherplätze für Adresse
```

Bild 3.2-7: Listing der Routine ADDRESS

Praktische Übung P3.2-3:

1. Ermitteln Sie die effektive Sprungadresse im Befehl „JSR 4,U“ im obigen Programmlisting für den im µR implementierten Monitor. Wie kommt diese Adressierung zustande? Kann der 6809-Assembler den Befehl verarbeiten ?
2. Welche Einschränkungen bzw. Änderungen müssen Sie vornehmen, um diese Routine statt der "Standard"-Routine benutzen zu können?
3. Assemblieren Sie die Routine *ADDRESS* und testen Sie sie aus.
4. Entwerfen Sie nach dem Muster der eben vorgestellten Adreßeingaberoutine eine Dateneingaberoutine.

Bei den Routinen *INSERT* und *DELETE* zum Einfügen und Entfernen von einzelnen Speicherstellen ergibt sich eine Schwierigkeit:

Die nachfolgenden Daten müssen im Speicher verschoben werden. Die Operation des Verschiebens selbst ist einfach, man muß sich aber darüber im Klaren sein, daß nicht der gesamte Speicherinhalt verschoben werden darf, da sonst auch der Stapelspeicher verschoben würde und der Rechner zwangsläufig abstürzt. Außerdem befinden sich oft mehrere Programme oder Programmteile gleichzeitig im Speicher, von denen nur eines geändert werden soll.

Die Frage ist also, welcher Speicherbereich verschoben wird. Wir haben uns dafür entschieden, jeweils 256 Bytes oberhalb der zu editierenden Adresse zu verschieben. In der Regel werden die von Ihnen geschriebenen Programme kürzer sein, und es besteht die Möglichkeit, zu anderen Programmen einen ausreichenden Zwischenraum zu lassen, so daß diese von der Verschiebung nicht betroffen sind.

Eine weitere Lösung des Problems besteht in der Festlegung eines Arbeitsbereichs, außerhalb dessen die Monitorroutinen nicht arbeiten. In diesen "geschützten" Bereich kann dann der Stack angelegt werden.

Praktische Übung P3.2-4:

Schreiben Sie eine Routine für die *INSERT*-Funktion und testen Sie sie aus.

B. Bearbeitung der Prozessor-Register

Mit der Taste R wird die Routine zur Darstellung und zum Editieren der Register des Mikroprozessors aktiviert. Dabei handelt es sich selbstverständlich nicht um die realen Register des Prozessors (‚die zum Betrieb des Monitors benötigt werden und auch auf keinen Fall von außen verändert werden können), sondern um einen reservierten Teil des Systembereichs ($0000-$03FF), der die Werte enthält, die beim Start eines Anwenderprogramms in den Prozessor geladen werden und bei Unterbrechung des Programms die "geretteten" Registerinhalte wieder aufnimmt.

Die Routine besteht, wie aus dem Flußdiagramm im Bild 3.2-8 ersichtlich, aus zwei Hauptteilen: Auswahl eines Registers und Editieren des Inhalts. Sie funktioniert ganz ähnlich wie die *DATA*-Routine, beide unterscheiden sich hauptsächlich in der Funktion der Tasten ’+’ und ‘-’. Während in der *DATA*-Routine lediglich die Adresse inkremen-

tiert bzw. dekrementiert wird, verwaltet die Hilfsroutine zur Parameterauswahl einen Zeiger auf eine Tabelle, der der eigentlichen Editierroutine den Zugriff auf die benötigten Parameter erlaubt. Das sind in diesem Fall die Kennung des Registers (wird im Datenfeld angezeigt), die "Länge" des Registers (1 oder 2 Bytes), die für die Auswahl zwischen *SHOWADR* und *SHOWDATA* gebraucht wird, und ein Verweis auf die Zero-Page-Adresse, in der sich der Registerinhalt befindet.

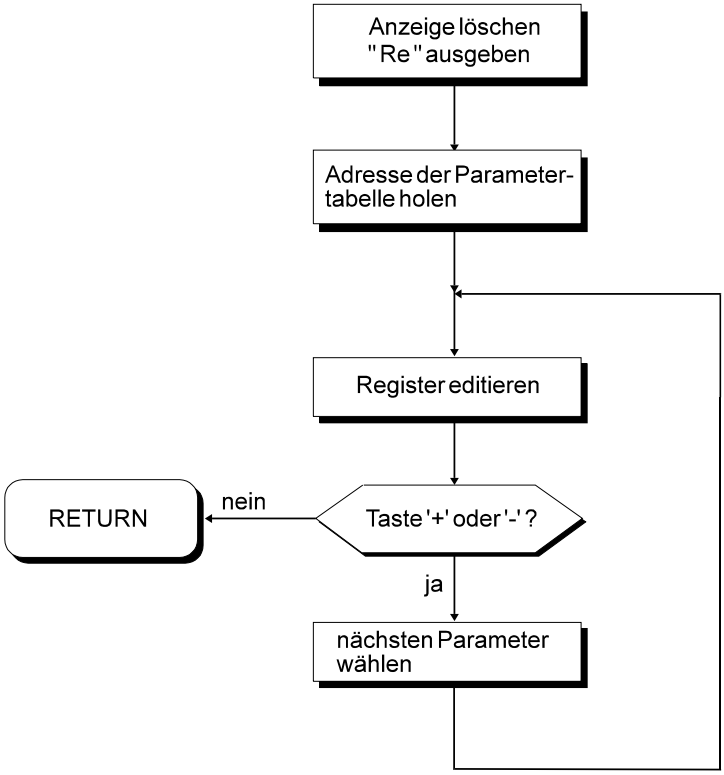


Bild 3.2-8: Flußdiagramm zur Routine *REGIST*

Da diese Routine ebenfalls nicht in der beschriebenen Form implementiert ist, haben wir das Listing einer universell nutzbaren Routine im Bild 3.2-9 angegeben, mit der es beispielsweise auch möglich wäre, die Registerinhalte der Peripheriebausteine zu editieren, und zwar allein durch die Ausgabe der entsprechenden Kennung im Operationsfeld und die Angabe der Basisadresse der benötigten Parametertabelle.

```
; Routine REGIST: Register editieren
; Kennung ausgeben und Startwerte setzen
REGIST:  PSHS   A,X,Y,CC      ; Register retten
         JSR    CLRDISP       ; Anzeige löschen
         LDD    #$5079        ; 7-Segmentcodes für "rE"
         LDX    #6            ;   im Operationsfeld
         JSR    SHOWD         ;   ausgeben
         LDY    #REGPAR       ; Basisadresse der Tabelle
         CLRA                 ;   mit erstem Register beginnen
---------------------------------------------------------------
         JSR    EDIT          ; zur Editierroutine
         PULS   A,X,Y,CC      ; Register restaurieren
```

```
            RTS                     ; zurück in den Monitor

; Editieren
EDIT:   PSHS    A,X,Y               ; Register-Nummer und Basisadresse retten
            LDB     #5              ; 5 Bytes
            MUL                     ;   pro Eintrag
            LEAY    B,Y             ;   zur Basisadresse addieren
            LDD     ,Y++            ; 7-Segmentcode für Kennung
            LDX     #0              ;   im Datenfeld
            JSR     SHOWD           ;   anzeigen
            LDX     #2              ; Editieren im Adreßfeld
            TST     ,Y+             ; 1- oder 2-byte-Daten
            PSHS    Y               ; Zeiger auf Parameter retten
            BEQ     BYTE            ; 0 für 1 Byte
            LDY     [,Y]            ; Datenwort editieren
            LDX     SHOWADR+1       ; dazu Sprung nach SHOWADR ohne
            JSR     4,X             ;   Löschen von Y, s. Bild 3.2-7
            TFR     Y,X             ; Datenwort nach X
            PULS    Y               ; Zeiger wiederherstellen
            STX     [,Y]            ;   und Datum ablegen
EDIT2:      PULS    A,X,Y           ; Register restaurieren
            CMPB    #$81            ; Taste '-' gedrückt ?
            BLS     $1              ; >$81?  (Tastencode für '-')
            RTS                     ;   dann zurück ins Hauptprogramm
            BSR     PARAM           ; sonst Parameter ändern
            BRA     EDIT            ; und wieder editieren
BYTE:       LDY     #0              ; nicht benutzte Anzeigestellen
            JSR     SHOWYD          ;   löschen
            PULS    Y               ; Zeiger auf Datum restaurieren
            LDA     [,Y]            ; Datenbyte editieren
            LDX     SHOWDATA+1      ; dazu Sprung nach SHOWDATA ohne
            JSR     2,X             ; Löschen von A
            STA     [,Y]            ; und wieder ablegen
            BRA     EDIT2           ; zurück nach EDIT
; Parameter ändern
PARAM:PSHS  CC  ; Status retten
            BEQ     MINUS           ; falls letzte Taste = $81
            INCA                    ; nächster Parameter
            CMPA    -1,Y            ; mit Tabellenende vergleichen
            BLE     $1              ; noch nicht überschritten
            CLRA                    ; sonst zum Tabellenanfang
            BRA     R               ; fertig
MINUS:      DECA                        ; vorheriger Parameter
            BPL     $2              ; ·0 ? dann o.k.
            LDA     -1,Y            ; sonst letzter Parameter
R:      PULS    CC  ; Status restaurieren
            RTS                         ; und zurück
```

Bild 3.2-9: Routine zum Editieren der Register

```
; Parametertabelle für Register
;
TABLEN    FCB    #8              ; 9 Eintragungen von 0 - 8
REGPAR:   FDB    #$0077          ; 1. Register: Kennung " A"
          FCB    #0              ; 0 = 1 Bytes
          FDB    #$00F5          ; Datum in $F5
          FDB    #$007C          ; 2. Register: Kennung " B"
          FCB    #0              ; 1 Byte
          FDB    #$00F6          ; Datum in $F6
          FDB    #$0076          ; 3. Register: Kennung " X"
          FCB    #1              ; 1 = 2 Bytes
          FDB    #$00F8          ; Datum in $F8/$F9
          FDB    $006E           ; 4. Register: Kennung „ Y“
          FCB    1               ; 2 Bytes
          FDB    $00FA           ; Datum in $FA
          FDB    $003E           ; 5. Register: Kennung „ U“
          FCB    1               ; 2 Bytes
          FDB    $00FC           ; Datum in $FC
          FDB    $006D           ; 6. Register: Kennung „ S“
          FCB    1               ; 2 Bytes
          FDB    $00F2           ; Datum in $F2
          FDB    $7358           ; 7. Register: Kennung „PC“
          FCB    1               ; 2 Bytes
          FDB    $00FE           ; Datum in $FE
          FDB    $5E73           ; 8. Register: Kennung „DP“
          FCB    0               ; 1 Byte
          FDB    $00F7           ; Datum in $F7
          FDB    $5858           ; 9. Register: Kennung „CC“
          FCB
    0     ; 1 Byte
          FDB    $00F4           ; Datum in $F4
```

Bild 3.2-9: Routine *REGIST* zum Editieren der Register (Fortsetzung)

Praktische Übung P3.2-5:

Assemblieren Sie die Routine *REGIST* und testen Sie sie.

C. Laden und Sichern von Programmen und Datenbereichen

Der Monitor bietet mit den Routinen *LOAD* und *SAVE* die Möglichkeit, Programme und Daten über die V.24-Schnittstelle zu einem PC zu übertragen und dort auf der Festplatte oder einer Floppy Disk zu speichern[4]. Diese Routinen gehören zu den aufwendigsten und umfangreichsten des Monitors. Wir wollen hier nur (eine mögliche Version für) die *LOAD*-Routine besprechen. (Die *SAVE*-Routine arbeitet analog dazu.)

Die Übertragung von Programmen und Daten erfolgt als ASCII-Zeichen im sogenannten **Intel-Hex-Format**, das im Bild 3.2-10 erklärt wird.

:	BA	ZAA_H	ZAA_L	ZK	DB_1	DB_2	DB_3	...	DB_{BA}	PS
:	10	04	00	00	BD	F1	10	...	39	3F
:	10	04	10	00	86	37	CC	...	7E	F4
:	10	04	20	00	8E	00	20	...	8E	BD
:	02	04	30	00	35	39	5E	(Prüfsumme)		
:	00	00	00	01	FF	(Prüfsumme)				

Bild 3.2-10: Beispiel für eine Übertragung im Intel-Hex-Format

Die Übertragung in diesem Format ist zeilenweise orientiert. Jede Zeile beginnt mit dem Startzeichen ':' (ASCII-Code $3A). Danach folgt - wie alle anderen Daten in hexadezimaler Form - die Anzahl BA der in der Zeile übertragenen Datenbytes DB_i. Es folgen zwei Bytes, die die Anfangsadresse ZAA (ZAA_H,ZAA_L) der übertragenen Datenzeile im Speicher angeben. Das nächste Byte ZK gibt durch ZK=$01 an, daß die letzte Zeile des Datenblockes übertragen wird. Alle vorherigen Zeilen werden durch ZK=$00 gekennzeichnet. Nach den Datenbytes DB_i wird zum Abschluß jeder Zeile eine Prüfsumme PS übertragen, die als Zweierkomplement der 8-bit-Summe aller vorausgehenden Bytes der Zeile (außer der Kennung ':') berechnet wird.

Selbsttestaufgabe S3.2-3:

Welchen Wert berechnet der Empfänger einer Übertragung im Intel-Hex-Format als Summe über alle Bytes einer Zeile (außer der Kennung ':'), in der kein Übertragungsfehler aufgetreten ist ?

Bild 3.2-11 zeigt das Flußdiagramm der Routine *Load*. Nicht gezeigt ist die Initialisierung der V.24-Schnittstelle, die bereits nach dem Einschalten des Rechners (*Power on Reset*) oder nach jedem Betätigen der Funktionstaste C (*Clear*) durchgeführt wird[5]. (Das Unterprogramm zum Empfang der übertragenen ASCII-Zeichen ist im Flußdiagramm 3.2-11 grau unterlegt dargestellt.)

Die empfangenen Daten werden im Programm *LOAD* vom ASCII-Code in die hexadezimale Form umgewandelt.[6] Tritt dabei ein Fehler auf, wird die *LOAD*-Routine mit einer Fehlermeldung: 'FEhLEr hE' ('hE': Hexadezimal-Umwandlung) beendet. Eine Fehlermeldung: 'FEhLEr PS' ('PS': Prüfsumme) wird auch erzeugt, wenn die vom Empfänger neu berechnete Prüfsumme einen Übertragungsfehler anzeigt.

[4] Ihre Benutzung wurde bereits in Kapitel 1, Schnitt 1.2.1, und in Kapitel 2 beschrieben.

[5] In Kapitel 4 wird die V.24-Schnittstelle und ihre Programmierung ausführlich beschrieben.

[6] Im Flußdiagramm 3.2-10 sind alle Funktionsblöcke mit '*)' gekennzeichnet, in denen diese Umwandlung stattfindet.

Praktische Übung P3.2-6:

Disassemblieren Sie - mit Hilfe des Programms DS9 - die *LOAD*-Routine (ab Adresse $FE00). Analysieren Sie das Programm und fügen Sie ausführliche Kommentare bei.

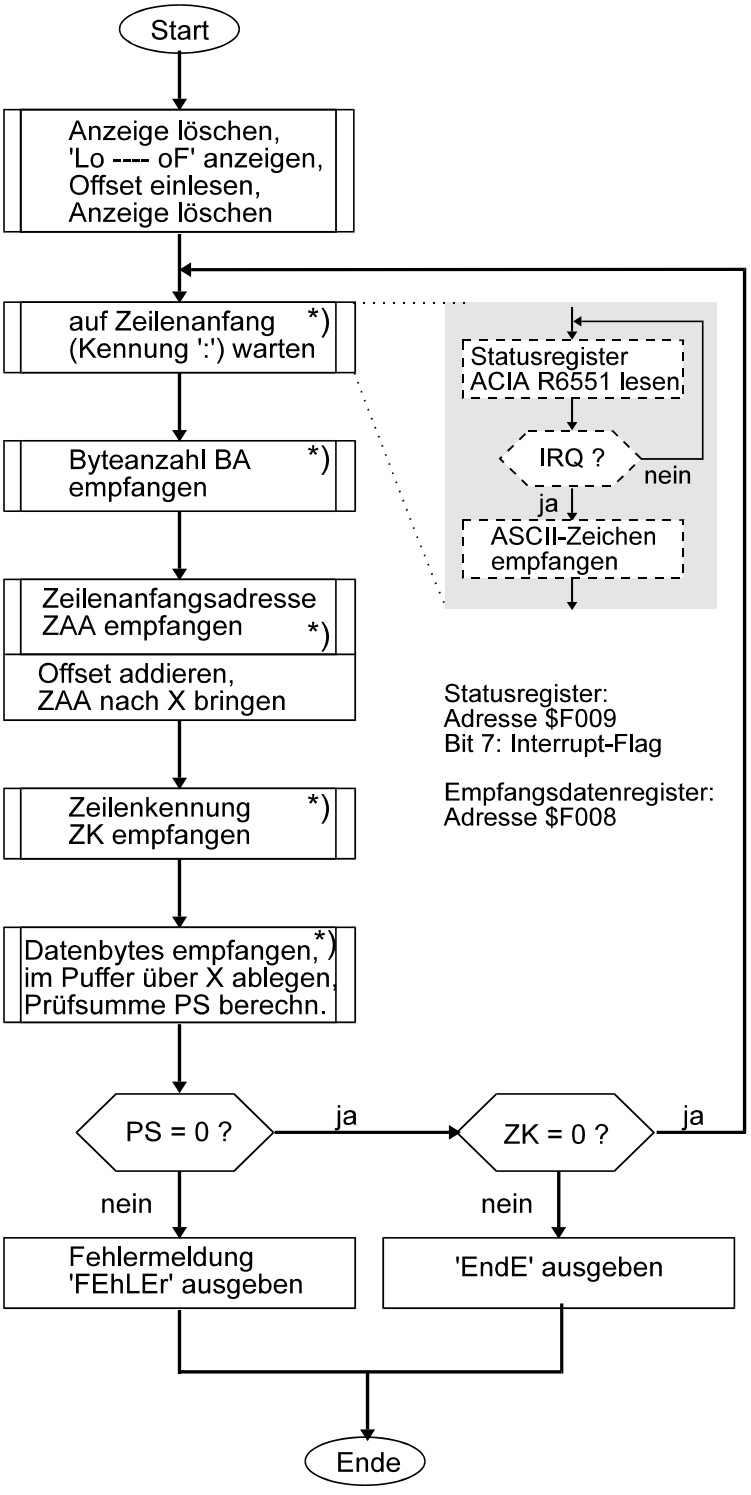


Bild 3.2-11: Flußdiagramm der Routine *LOAD*[7]

[7] Durch *) sind alle Funktionsblöcke gekennzeichnet, in denen eine ASCII- nach hexadezimal Umwandlung stattfindet.

D. Ausführung der Anwenderprogramme

Zur Kontrolle der auf dem Rechner laufenden Programme gehört nicht nur der Start eines Programms, sondern genauso seine kontrollierte Beendigung. Zu diesem Zweck müssen geeignete Maßnahmen ergriffen werden, um den Rechner beim Start eines Programms in einen definierten Ausgangszustand zu versetzen und den Zustand des Systems bei der Beendigung oder beim Abbruch des Programms zu speichern, um später Rückschlüsse auf dessen Verhalten ziehen zu können. Insbesondere reicht es nicht aus, ein Anwenderprogramm durch einen Sprung zu seiner Startadresse zu aktivieren.

Der normale Programmstart geht folgendermaßen vor sich: Nachdem die Startadresse eingegeben und bei Bedarf ein Unterbrechungspunkt gesetzt wurde, setzt die *GO*-Routine den Stackpointer S auf den vorgewählten Wert und führt einen Software-Interrupt SWI2 aus. Damit werden die Inhalte aller Register des Prozessors (außer dem Stackpointer S) auf den Stapel gelegt und - da sie im Benutzerprogramm nicht benötigt werden - Platz für die Registerinhalte geschaffen, die vom Anwenderprogramm gebraucht werden. Diese werden nun aus dem Systembereich in den Stapel kopiert. Mit dem Befehl RTI holt der Prozessor nun alle Register vom Stapel, einschließlich des PC-Registers, das dann die Startadresse des Programms enthält.

Die *Trace*-Funktion ist eine der interessantesten Routinen des Monitors, deshalb geben wir sie in Bild 3.2-10 vollständig wieder - wenn auch in einer anderen als der im Monitor implementierten Form. Sie startet das Programm prinzipiell auf die gleiche Art und Weise wie die *GO*-Routine. Der wesentliche Unterschied zwischen beiden ist die Vorbereitung zur Unterbrechung des Programms nach der Ausführung eines einzigen Befehls.

Dazu wird der Zähler #2 des Timer-Bausteins MC6840[8] so eingestellt, daß während der Abarbeitung des ersten Programmbefehls eine NMI-Anforderung an der CPU eintrifft. Diese unterbricht daraufhin das laufende Programm am Ende des momentan ausgeführten Befehls, rettet alle Register auf den Stapel und verzweigt in die NMI-Routine.

Dort wird die Beendigung des Programms fortgesetzt: Die Registerinhalte werden vom Stapel in den Systembereich zurückkopiert, der Stackpointer auf den Wert vor dem Eintreffen des NMI zurückgesetzt und die entsprechende Meldung auf dem Display ausgegeben. Damit ist die *TRACE*-Routine beendet und der Monitor bereit, neue Aufgaben durchzuführen, z.B. die Inhalte der Register anzuzeigen etc.

Mit der vollständigen Beschreibung der *TRACE*-Routine haben Sie schon eine Methode kennengelernt, aus dem Anwenderprogramm in den Monitor zurückzukehren. Die beiden anderen Rückkehrroutinen, SWI1 zur Bearbeitung eines Breakpoints und die Taste F4 zum Abbruch eines Programms von der Tastatur aus, funktionieren grundsätzlich genauso, wobei zu beachten ist, daß die Taste F4 eine maskierbare Unterbrechungsanforderung IRQ erzeugt, die im Gegensatz zum nicht maskierbaren Interrupt NMI der *TRACE*-Routine nur dann ausgeführt wird, wenn der Interrupt nicht durch das Setzen des I-Flags im Statusregister der CPU blockiert wurde[9].

---

[8] Dieser Baustein wir in Kapitel 5 ausführlich beschrieben.

[9] vgl. Abschnitt 1.3.3 in Kapitel 1

```
; Routine TRACE: Einzelschrittausführung
TI1CNT:   EQU   $F018
TI2CNT:   EQU   $F019
TI2MSB:   EQU   $F01C
TI2LSB:   EQU   $F01D
RECTAB:   EQU   $00F4
PCREG:EQU   $00FE
CCREG:    EQU   $00F4

TRACE:    SWI3      ; Einen Programmbefehl ausführen
;         Hier wird nach dem NMI fortgefahren
          LDA   TI2CNT      ; NMI abschalten durch Lesen von St.Reg. 2
          LDA   #$01        ;
          STA   TI1CNT      ; Timer Reset, sperren
          JSR   Kopie       ; Registersatz vom Stack in Zero-Page kop.
RETRAC:   JSR   CLRDISP     ; Anzeige löschen
          LDD   #$3150      ; "Tr"
          LDX   #6          ;  im Operationsfeld
          JSR   SHOWD       ;  anzeigen
TLOOP:    LDY   #PCREG      ; Startadresse nach Y
          LDB   ,Y          ; zugehöriges Datum
          LDX   #0          ;  im Datenfeld
          JSR   SHOWB7SG    ;  anzeigen
          LDX   #2          ; im Adreßfeld
          JSR   SHOWADR     ;  Startadresse editieren
          CMPB  #$80        ; zurück mit Taste '+' ?
          BNE   +4          ; falls nein, weiter
          LEAY  1,Y         ; Startadresse inkrementieren
          BRA   TLOOP+4     ; erneut editieren
          CMPA  #$81        ; zurück mit Taste '-' ?
          BNE   +4          ; sonst zurück zur Hauptschleife
          LEAY  -1,Y        ; Startadresse dekrementieren
          BRA   TLOOP+4     ; erneut editieren
          LDB   ,Y          ; Datum
          LDX   #0          ;  im Datenfeld
          JSR   SHOWB7SG    ;  anzeigen
          STY   #PCREG      ; neue Startadresse ablegen
          LBRA  MLOOP       ; zurück zur Hauptschleife
; Start des Programms für TRACE
SWI3:     LDA   <CCREG      ; Statusregister
          ORA   #$90        ; E- und I-Flag setzen
          STA   <CCREG      ; wieder ablegen
; Kopieren der Register vom Systembereich in den Stapel
          TFR   S,Y         ; Y ist Ziel für die Register
          LDX   #REGTAB     ; X ist die Quelle
```

Bild 3.2-10: Listing der *TRACE*-Routine

```
            LDB     #$C         ; Länge der Tabelle
            LDA     ,X+         ; Datum holen
            STA     ,Y+         ;   und auf dem Stapel ablegen
            DECB                ; bis 12 Bytes kopiert,
            BNE     -7          ; wiederholen
; Timer setzen
            LDA     #$C3        ; Zähler #2, Interrupt zum NMI zulassen
            STA     TI2CNT      ; in Kontrollregister 2
            CLRA                ; 0 nach
            STA     TI2MSB      ;   Zähler #2, MSB-Buffer
            LDA     #$D         ; 13 nach
            STA     TI2LSB      ;   Zähler #2, LSB-Buffer
            CLRA                ; Reset abschalten,
            STA     TI1CNT      ; Zähler #2 läuft los
            RTI                 ; Sprung in das Programm
```

Bild 3.2-11: Listing der *TRACE*-Routine (Fortsetzung)

Selbsttestaufgabe S3.2-4:

Vergleichen Sie den Programmabbruch bei Erreichen eines Breakpoints (SWI) mit dem bei der Einzelschrittausführung (NMI). Wo liegt der wesentliche Unterschied ?

Die selbstprogrammierbaren Tastenfunktionen sind zwar mit den Routinen *INSERT*, *DELETE* und *DUMP* vorbelegt, können aber jederzeit auf eigene Routinen umgeleitet werden, indem die Startadresse in der Sprungtabelle des Kommandointerpreters gegen die Startadressen Ihrer eigenen Routinen ausgetauscht werden. Sie finden die entsprechenden Adressen bei der Beschreibung der Routinen in Kapitel 1.
Da der Kommandointerpreter, der die Routinen aktiviert, etwas anders arbeitet, als der in dieser Kurseinheit beschriebene, müssen Sie die Routinen mit dem Befehl JMP $E38F abschließen. Wie schon weiter vorn beschrieben, akzeptiert der Monitor einen eventuell schon im Akku B stehenden neuen Befehl nicht, der Rücksprung endet daher in der Tastaturabfrage des Monitors, und Sie müssen die gewünschte Funktionstaste noch einmal drücken.

Als Beispiel wollen wir die Taste F1 mit der Funktion *FILL* belegen, die einen Speicherbereich zwischen zwei anzugebenden Bereichsgrenzen mit einem bestimmten Datum füllt. Wir benötigen dazu die Angaben über Bereichsanfang und Bereichsende und das Datum.

Für die Auswahl der Parameter verwenden wir praktischerweise die Routine, die auch zum Editieren der Register benutzt wird. Da sie gegenwärtig im Rechner nicht zur Verfügung steht, müssen wir sie selbst eintippen (Sie haben sie ja wahrscheinlich schon längst assembliert). Das Format der Tabelle entspricht ebenfalls exakt dem der vorgestellten Routine. Sie ist in Bild 3.2-12 dargestellt.

```
; Parametertabelle für FILL
;
            ORG     $1800           ; Startadresse der Tabelle
TABLEN:     FCB     2               ; Anzahl der Einträge -1
FILPAR:     FDB     $7C77           ; Bereichsanfangs-Kennung "bA"
            FCB     1               ; 2-byte-Datum
            FDB     #BA             ; bei Adresse $1810
            FDB     $7C79           ; Bereichsende-Kennung "bE"
            FCB     1               ; 2-byte-Datum
            FDB     #BE             ; bei Adresse $1812
            FDB     $5E77           ; Datums-Kennung "dA"
            FCB     0               ; 1-Byte-Datum
            FDB     #DA             ; bei Adresse $1814
;
            ORG     $1810           ; Daten für die FILL-Routine
BA:         FDB     $400            ; vorgegebener Bereichsanfang
BE:         FDB     $4FF            ; vorgegebenes Bereichsende
DA:         FCB     $FF             ; vorgegebenes Datum
```

Bild 3.2-12: Parametertabelle für die *FILL*-Routine

Es fehlt noch die eigentliche Routine. Zunächst entwickeln wir das Flußdiagramm nach Bild 3.2-13:

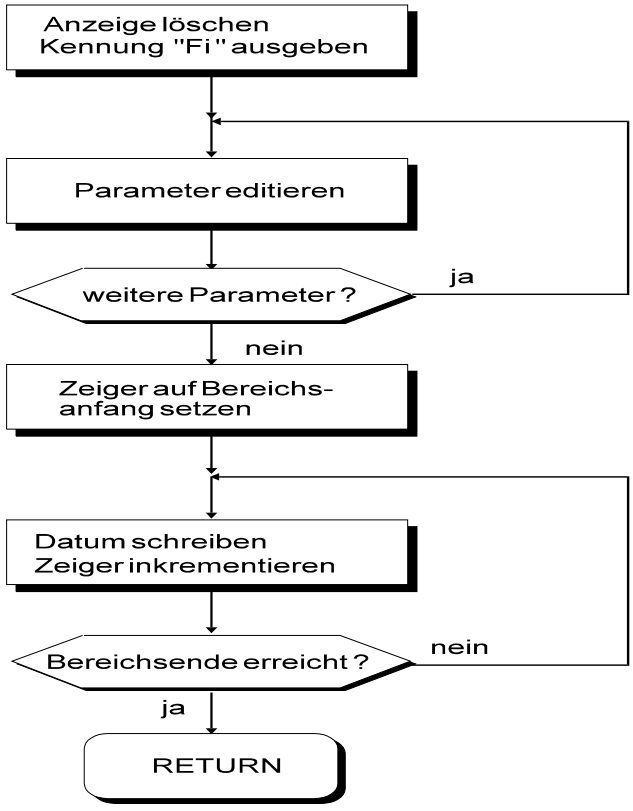


Bild 3.2-13: Das Flußdiagramm der FILL-Routine

Damit gestaltet sich das Assemblerprogramm recht einfach. Dies ist im Bild 3.2-14 dargestellt.

```
; Routine FILL    (Taste F1)
;
        ORG     $1820
;
FILL:   PSHS    A,X,Y,CC        ; Register retten
        JSR     CLRDISP         ; Anzeige löschen
        LDD     #$7104          ; 7-Segmentcode für "FI"
        LDX     #6              ;   im Operationsfeld
        JSR     SHOWD           ;   anzeigen
        LDY     #FILPAR         ; Y zeigt auf die Tabelle
        CLRA                    ; ersten Parameter
        JSR     EDIT            ;   editieren
        CMPB    #$89            ; zurück mit F1?
        BNE     R               ; sonst Ende
        LDX     BA              ; Bereichsanfang in X
        LDA     DA              ; Datum in A
LOOP:   STA     ,X+             ;   in den Speicher schreiben
        CMPX    BE              ; Bereichsende erreicht ?
        BNE     LOOP            ; falls nein, wiederholen
        STA     ,X              ; letztes Datum schreiben
        LDY     #$7954          ; 7-Segment-Code für
        LDD     #$5E79          ;   "EndE" in Y und D
        LDX     #2              ;   im Adreßfeld
        JSR     SHOWYD          ;   anzeigen
R:      PULS    A,X,Y,CC        ; Register restaurieren
        JMP     $E38F           ; zurück zum Monitor
```

Bild 3.2-14: Die *FILL*-Routine in Assemblerschreibweise

Praktische Übung P3.2-7:

Assemblieren Sie das Programm und testen Sie es aus, indem Sie die Taste F1 neu belegen.

3.2.5 Interruptbehandlung

Oft muß ein Rechner auf Ereignisse reagieren, die unabhängig vom laufenden Programm ("asynchron") auftreten. Grundsätzlich gibt es hierzu zwei Möglichkeiten, das Auftreten dieser Ereignisse zu erkennen. Die erste besteht in der regelmäßigen Abfrage der Statusregister der angeschlossenen Peripheriebausteine auf ein solches Ereignis. Der Nachteil dieser Methode liegt auf der Hand: Das laufende Programm muß regelmäßig in eine entsprechende Routine springen, die die Peripherie auf Unterbrechungsanforderungen testet. Diese Art der Programmierung ist nicht nur

umständlich, in vielen Fällen sogar unmöglich, es wird auch ein großer Teil der Rechenkapazität des Prozessors auf Abfragen verschwendet, die nicht zu einer Unterbrechung führen. Außerdem kann es unter Umständen relativ lange dauern, bis auf die Unterbrechungsanforderung reagiert wird.

Aus diesem Grund besitzen Mikroprozessoren spezielle Eingangsleitungen, an die die Signale zur Unterbrechungsanforderung angelegt werden können. Der 6809-Prozessor stellt drei Leitungen für diese Signale zur Verfügung, die mit IRQ, FIRQ und NMI bezeichnet.

Die Arbeitsweise der Interrupts wurde im Einzelnen schon in Kapitel 1, Abschnitt 1.3.2, vorgestellt. Der Monitor benutzt zwei der drei Hardware-Interrupts, nämlich IRQ und NMI, und alle drei Software-Interrupts (SWI).

Zur Rückkehr aus einer Unterbrechung besitzt der Prozessor den speziellen Befehl RTI (*ReTurn from Interrupt*), der automatisch die zu Beginn der Unterbrechung gesicherten Register wieder vom Stapel holt. Er benutzt dazu das E-Bit des Statusregisters, das angibt, ob alle Register gerettet wurden (E = 1) oder nur die Register CC und PC.

Zurück zu den Hardwareunterbrechungen. Ein reales Computersystem kommt mit den drei zur Verfügung stehenden Interruptleitungen in der Regel bei weitem nicht aus. Selbst die Hardware des vergleichsweise kleinen Praktikumsrechners kann neun verschiedene Interrupts erzeugen, außerdem können extern weitere Geräte angeschlossen sein, die Unterbrechungsanforderungen an den Rechner schicken. Da jede dieser Anforderungen eine individuelle Behandlung verlangt, dem Prozessor aber nur drei Hardware-Interruptvektoren[*)] zur Verfügung stehen, ist es zunächst notwendig, die Herkunft eines Interrupts zu klären, um dann zu der speziellen Behandlungsroutine zu verzweigen. Diese Aufgabe wird von drei Auswertungsroutinen übernommen, die ebenfalls zum Monitorprogramm gehören und den drei Unterbrechungsleitungen zugeordnet sind. Die Startadressen der Auswertungsroutinen stehen in einem Stapel von µP-Vektoren, in dem auch der PC-Startwert und die SWI-Vektoren zu finden sind (s. Kapitel 1, Tabelle 1.3-1).

Am Beispiel der IRQ-Auswertungsroutine, deren Flußdiagramm Sie in Bild 3.2-15 finden, sehen Sie, wie das System funktioniert. Nacheinander werden die Peripheriebausteine daraufhin getestet, ob das Interrupt-Flag im Statusregister gesetzt ist und, falls ja, ob der Baustein den Interrupt auch an den Prozessor weitergeleitet hat. Das ist nötig, da es die Möglichkeit gibt, die Interrupt-Ausgangsleitung eines Peripheriebausteins abzuschalten, so daß ein im Statusregister angezeigter Interrupt gar nicht zum Prozessor weitergeleitet wird.

Die Abfrage wird fortgesetzt, bis die Quelle der Unterbrechung gefunden ist. Dann wird über eine Sprungtabelle, die sich im Systembereich befindet, die entsprechende Behandlungsroutine aufgerufen[10]. Dadurch können Sie bei Bedarf eigene Routinen zur Bedienung der Unterbrechungen benutzen, indem Sie die Startadressen Ihrer eigenen Routinen in die Tabelle eintragen. Vom Monitor benutzt werden derzeit nur der IRQ über den Eingang CA1 von Port A des Parallel-Schnittstellenbausteins

*) vgl. Tabelle 1.3-1 in Kapitel 1

10 Diese Tabelle der Interruptvektoren finden Sie im Anhang dieses Kapitel s.

MC6821[11], der das Signal der *BREAK*-Taste weiterleitet, und der NMI von Zähler #2 von Baustein MC6840 für die *TRACE*-Funktion. Alle anderen Vektoren zeigen auf die Adresse $E31C und damit auf den Befehl RTI ($3B), der unter dieser Adresse abgespeichert ist.

Für den Fall, daß mehrere Geräte an den Bus angeschlossen sind, die den gleichen Interrupt auslösen können, hat der Benutzer dafür Sorge zu tragen, daß der Vektor für eine Unterbrechung von außerhalb des Rechners auf eine weitere Auswertungsroutine zeigt, die nach oben dargestelltem Verfahren das auslösende Gerät ermittelt und die weitere Ausführung einleitet.

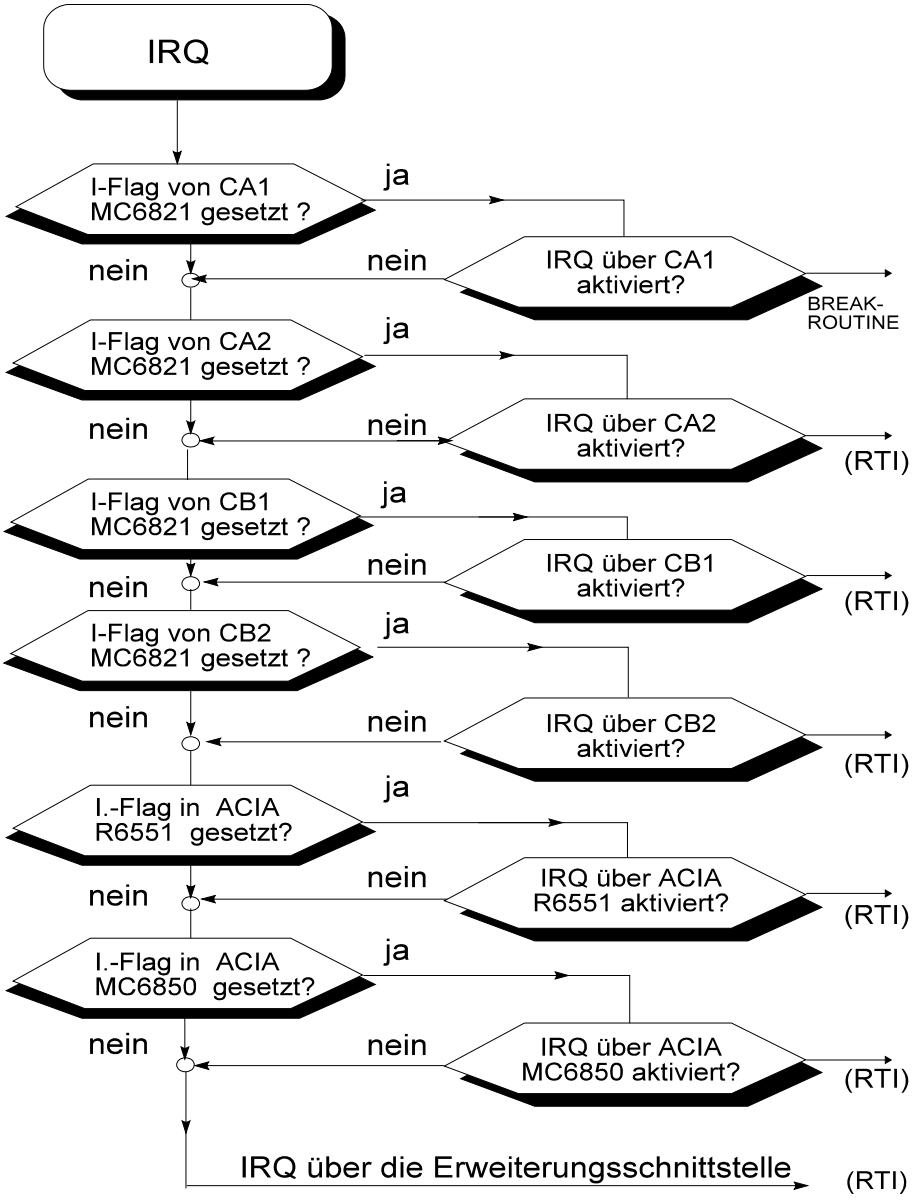


Bild 3.2-15: Flußdiagramm der Routine zur Auswertung des IRQs

[11] Die Schnittstellen-Bausteine werden Sie in Kapitel 4 und Kapitel 5 kennenlernen.

In der Interrupt-Vektorentabelle finden sich im übrigen auch die Startadressen der SWI-Routinen, die alle drei vom Monitor benutzt werden: SWI1 kennzeichnet einen Breakpoint im Anwenderprogramm, SWI2 führt den Programmstart und SWI3 die *TRACE*-Funktion durch. Um Ihnen die Möglichkeit zu geben, wenigstens SWI2 und SWI3 ebenfalls zu benutzen, werden auch diese auf ihren Ursprung hin untersucht: Durch die Abfrage der Rücksprungadresse auf dem Stapel wird festgestellt, ob der SWI aus dem RAM- oder ROM-Bereich kam. Ist sie kleiner als $8000, handelt es sich um einen vom Benutzer erzeugten SWI, anderenfalls wurde er vom Monitor ausgelöst.

Zum Abschluß wollen wir Ihnen noch ein typisches Beispiel für den Unterschied zwischen der Abarbeitung einer Routine mit wiederholter Abfrage eines Bausteins oder per Interrupt geben: Die Ausgabe einer Zeichenkette an die serielle Schnittstelle. Beide Flußdiagramme sind in Bild 3.2-16 und Bild 3.2-17 dargestellt. Während im ersten Fall der Prozessor während der gesamten Übertragungszeit mit der Ausgabe beschäftigt und damit für andere Aufgaben blockiert ist, wird er bei der zweiten Methode jeweils nur für wenige Mikrosekunden für die Ausgabe eines Zeichens beansprucht und ist während der restlichen Zeit für andere Aufgaben verfügbar[12].

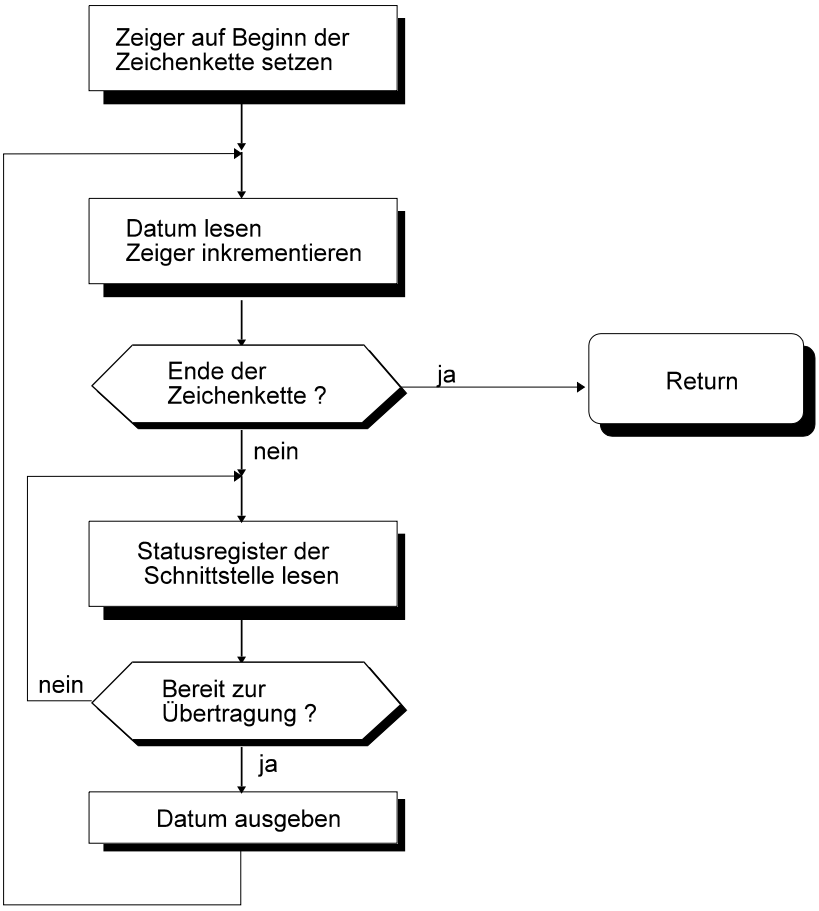


Bild 3.2-16: Ausgabe mit zyklischer Abfrage des Statusregisters

[12] In Kapitel 4 werden Sie die benötigten Schnittstellen-Bausteine kennenlernen. Damit werden Sie in der Lage sein, die eben beschriebenen Programme selbst zu schreiben.

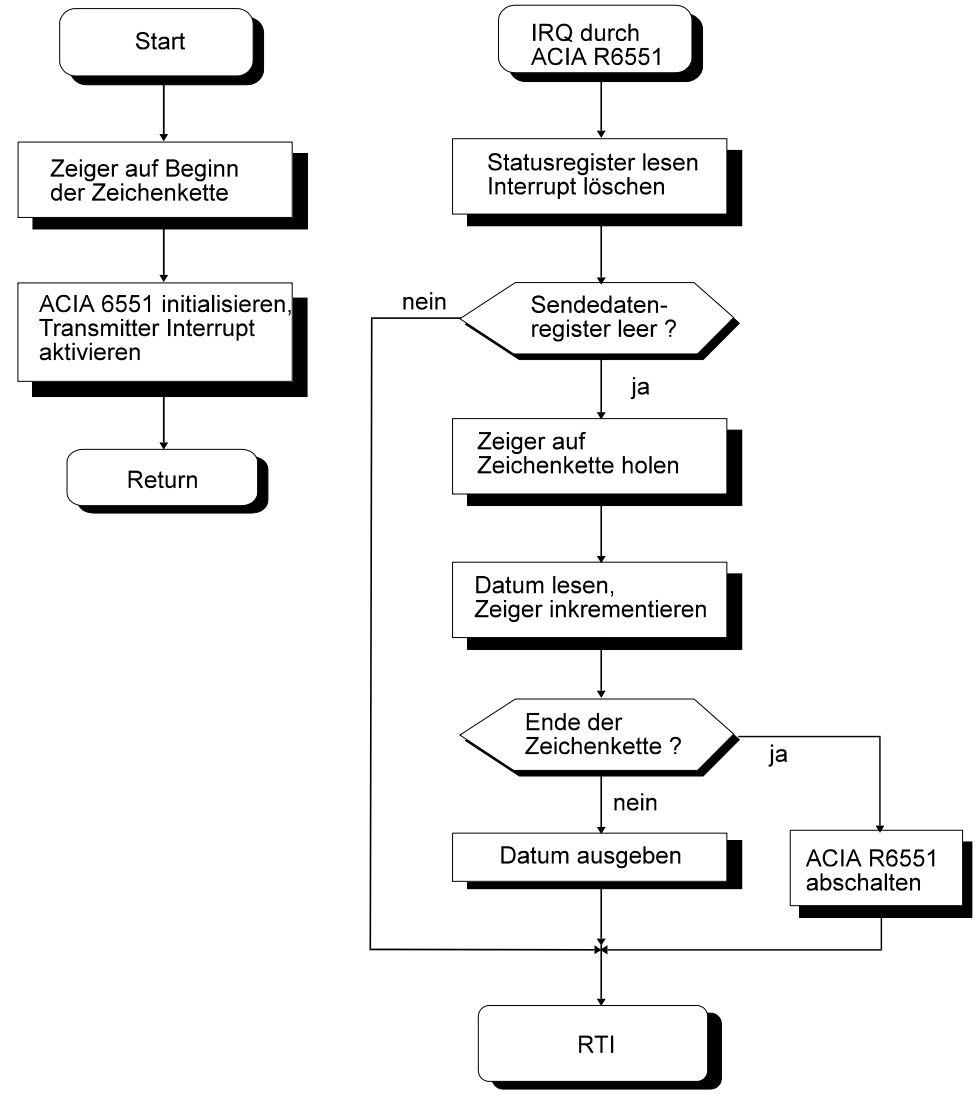


Bild 3.2-17: Ausgabe mit Interruptsteuerung

Lösungsvorschläge zu den Praktischen Übungen

Zu P3.1-1:

1. Es ist zu sehen, wie nach Betätigen der Taste 'C' das Monitorprogramm zunächst die Speicherzellen $00-$FF initialisiert. Danach wird mit Hilfe eines Datenpuffers in den Zellen $80-$90 die Anzeige aufgebaut. Jede Betätigung einer weiteren Funktionstaste zeigt den wiederholten Zugriff auf diesen Pufferbereich.
2. Das Programm realisiert drei Hexadezimalzähler in den Akkumulatoren A und B sowie der Speicherzelle $00. Diese werden im Hauptprogramm (HP) zunächst initialisiert und dann in HP sowie zwei ineinander geschachtelten Unterprogrammen (SUB1,SUB2) aufwärtsgezählt. Vor jedem Unterprogrammaufruf wird der entsprechende Zählerzustand auf dem Stack abgelegt und nach der Rückkehr aus dem Unterprogramm von dort wieder geladen. Auf der tiefsten Schachtelungstiefe wird in einem weiteren Unterprogramm (SUB3) eine Zeitverzögerung mit der Routine DLY1MS erzeugt (Y=$0040=64 ms). (In DLY1MS wird der eingegebene Verzögerungswert zunächst auf dem Stack gerettet. RA: Rücksprung-Adressen.)

	Adresse	OpCode		Mnemocode		Bemerkung
1	; **Stack-Beobachtung**					
2		DLY1MS		EQU	$F160	; Verzögerungsroutine
3						
4	0400			ORG	$0400	
5	0400	CC 10 20	HP	LDD	#$1020	; Zähler Z1 in Akku A, Z2 in B
6	0403	0F 00		CLR	$00	; Zähler Z3 in $00
7	0405	4C	L	INCA		; Z1 inkrementieren
8	0406	34 02		PSHS	A	; Z1 auf den Stack
9	0408	BD 04 20		JSR	SUB1	
10	040B	35 02	RA1	PULS	A	; Z1 vom Stack
11	040D	20 F6		BRA	L	; Endlosschleife
12						
13	0420			ORG	$0420	
14	0420	5C	SUB1	INCB		; Z2 inkrementieren
15	0421	34 04		PSHS	B	; Z2 auf den Stack
16	0423	BD 04 40		JSR	SUB2	
17	0426	35 04	RA2	PULS	B	; Z2 vom Stack
18	0428	39		RTS		; Rücksprung
19						
20	0440			ORG	$0440	
21	0440	0C 00	SUB2	INC	$00	; Z3 inkrementieren
22	0442	96 00		LDA	$00	; Z3 nach Accu A
23	0444	34 02		PSHS	A	; Z3 auf den Stack
24	0446	BD 04 60		JSR	SUB3	
25	0449	35 02	RA3	PULS	A	; Z3 vom Stack
26	044B	39		RTS		; Rücksprung

```
27
28  0460                        ORG  $0460
29  0460  10 8E 00 40  SUB3    LDY  #$0040      ; Verzögerungswert Yd
30  0464  BD F1 60             JSR  DLY1MS      ; Verzögerungsroutine
31  0467  39           RA4     RTS              ; Rücksprung
```

Die Beobachtung des Stack-Bereichs zeigt die folgende Speicherzellenbelegung:

Adresse				YdH		YdL	RA4		RA3H
1FF0	FF	88	Z4	00		40	04	67	04
1FF8	49	Z3	04	26		Z2	04	0B	Z1
	RA3L		RA2			RA1			

Zu P3.1-2:

Der Adreßbereich $1040 -$1047 wird in der folgenden Routine zum Anzeige-Puffer gemacht, indem das X-Register auf den Wert $1040 gesetzt wird. Dieser Puffer wird zunächst mit der Routine CLRBUF (s. Abschnitt 1.2.2) "gelöscht". Danach wird er byte-weise unter Beachtung etwaiger Überträge ins nächste Byte inkrementiert. Mit der Routine SHOWDBUF wird nach jeder Erhöhung der Puffer in die Anzeige übertragen.

Adresse	**OpCode**		**Mnemo-Code**		**Bemerkung**
0400	8E 10 40		LDX	#$1040	; X setzen
0403	BD F1 30		JSR	CLRDBUF	; Löschen des Puffers
0406	86 00	L:	LDA	#0	; Bytezähler initialisieren
0408	6C 86	M:	INC	A,X	; Byte inkrementieren
040A	26 05		BNE	N	; zur Anzeigeroutine
040C	4C		INCA		; Bytezähler inkrementieren
040D	81 08		CMPA	#8	; letztes Byte ?
040F	26 F7		BNE	M	; nächstes Byte
0411	BD F1 33	N:	JSR	SHOWDBUF	; Anzeigeroutine
0414	10 8E 00 01		LDY	#1	; Y mit 1 ms laden (1 kHz)
0418	BD F1 60		JSR	DLY1MS	; Verzögerung
041B	7E 04 06		JMP	L	

Zu P3.1-3:

Adresse	Opcode		Mnemocode		Bemerkung
0400	BD F1 10		JSR	CLRDISP	; Löschen der Anzeige
0403	8E F0 10		LDX	#$F010	; X initialisieren
0406	86 07	N:	LDA	#7	; Spaltenzähler initialisieren
0408	E6 86		LDB	A,X	; Spaltenwert nach B bringen
040A	EA 86	M:	ORB	A,X	; Spalten „verodern“
040C	C4 0F		ANDB	#$F	; Bits 7-4 maskieren
040E	F7 F0 20		STB	$F020	; in die Anzeige bringen
0411	4A		DECA		; nächste Spalte
0412	2A F6		BPL	M	; falls nicht letzte Spalte
0414	2B F0		BMI	N	; falls letzte Spalte

Zu P3.2-1:

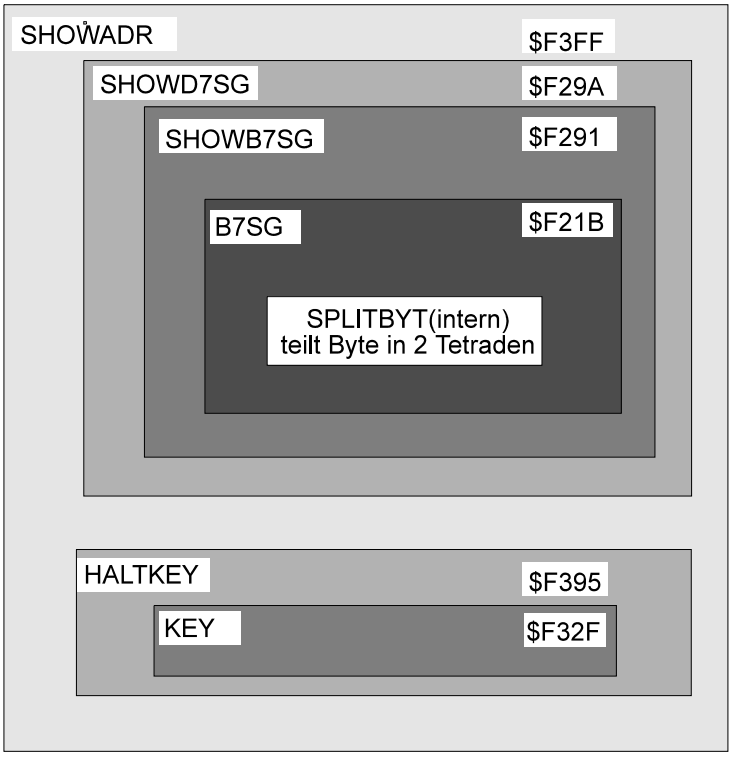
Auf das Flußdiagranm der Routine können wir in diesem Fall verzichten,da wir uns an der Interpreterschleife nach Bild 3.2-2 orientieren. Das Assemblerlisting sieht dann so aus:

Adresse	OpCode		Mnemocode		Bemerkung
;	**Kommandointerpreter**				
1800			ORG	$1800	
1800	8E 00 02	KOM:	LDX	#2	; Anzeige im Adreßfeld
1803	BD F1 56		JSR	SH0WADR	; Kommandoeingabe
1806	C1 80		CMPB	#$80	; letzte Taste = "+"?
1808	26 F9		BNE	-7	; sonst neues Kommando
180A	8E 18 30		LDX	#KOMTAB	; Beginn der Sprungtabelle
180D	CC FF FF		LDD	#$FFFF	; Zeichen für Tabellenende
1810	10 AC 81	NAECHST:	CMPY	,X + +	; Vergleich Befehl mit Tabelle
1813	27 12		BEQ	GEFUNDEN	; falls gleich, gefunden
1815	30 02		LEAX	2,X	; nächste Eintragung
1817	10 A3 84		CMPD	,X	; Tabellenende erreicht?
181A	26 F4		BNE	NAECHST	; sonst weitertesten
181C	8E 00 00		LDX	#0	; im Datenfeld
181F	CC 00 5B		LDD	#$5B	; " ?"
1822	BD F1 16		JSR	SHOWD	; ausgeben
1825	20 DC		BRA	KOM	; zur Kommandoeingabe
1827	AD 94	GEFUNDEN:	JSR	[,X]	; Sprung zur Routine
1829	BD F1 10		JSR	CLRDISP	; Anzeige löschen
182C	20 D5		BRA	KOM	; zur Kommandoeingabe
1830			ORG	$1830	
1830	41 44	KOMTAB:	FDB	'AD'	; ASCII-Code des Bezeichners
1832	E3 B0		FDB	ADDRESS	; Startadresse der Routine
1834	44 41		FDB	'DA'	;

```
1836  E3 F0          FDB   DATEN      ;
1838  52 45          FDB   'RE'       ;
183A  E5 3C          FDB   REGIST     ;
183C  47 4F          FDB   'GO'       ;
183E  E4 30          FDB   GO         ;
1840  53 41          FDB   'SA'       ;   für jede Routine:
1842  E9 00          FDB   SAVE       ;
1844  4C 4F          FDB   'LO'       ;   Bezeichner
1846  EA 00          FDB   LOAD       ;   Startadresse
1848  54 52          FDB   TR'        ;
184A  EC A0          FDB   TRACE      ;
184C  49 4E          FDB   'IN'       ;
184E  EC 0E          FDB   INSERT     ;
1850  44 45          FDB   'DE'       ;
1852  EC 36          FDB   DELETE     ;
1854  44 55          FDB   'DU'       ;
1856  EC 5B          FDB   DUMP       ;
1858  FF FF          FDB   $FFFF      ; Ende der Sprungtabelle
```

Der wesentliche Unterschied zwischen beiden Routinen besteht in der Ausführung der Tabelle, die hier sowohl die Bezeichner der Monitorroutinen als auch deren Startadressen aufnehmen muß.

Zu P3.2-2: 1.



2.

Routine **INDATA**

Eingaberegister:

Ausgaberegister: A: Datum, B:Funktionstaste

Funktion: Liest die Tastatur und packt die beiden zuletzt eingegebenen Hex-Zeichen nach A. Die Routine wird durch eine beliebige Funktionstaste verlassen, deren Code in B zur Verfügung steht. Initialisierung von A mit $00.

Adresse	OpCode		Mnemocode		Bemerkung
F3B6			ORG	$F3B6	
F3B6	4F		CLRA		; Initialisierung des Datums
F3B7	8D DC	L:	JSR	HALTKEY	; auf eine Taste warten
F3B9	2A 01		BPL	W	; falls Hexadezimalziffer
F3BB	39		RTS		; zurück mit Funktionstastencode
F3BC	34 04	W:	PSHS	B	; Ziffer auf den Stack retten
F3BE	48		LSLA		; | Zahl um eine Ziffer
F3BF	48		LSLA		; | nach links schieben
F3C0	48		LSLA		; |
F3C1	48		LSLA		
F3C2	AA E0		ORA	,S+	; neue Ziffer einfügen
F3C4	20 F1		BRA	L	; nächste Taste

Zu P3.2-3:

1. Die tatsächliche Startadresse ist zu ermitteln, indem Sie in der Sprungtabelle für die Hilfsroutinen bei SHOWADR nachsehen. Dort steht der Befehl "JMP $F3FF“, die Startadresse steht also in den Speicherstellen SHOWADR+1 und SHOWADR+2. Da der erste Befehl der Routine,`“LDY #0“ (hexadezimal: 10 8E 00 00), nicht ausgeführt werden soll, muß zu der Startadresse vier addiert werden. Es ergibt sich die absolute Sprungadresse $F403.
   Der Sprungbefehl müßte richtig JMP [SHOWADR+1]+4 lauten. Ein 6809-Assembler kann diesen Ausdruck aber nicht verarbeiten, da der Prozessor diese Adressierung nicht kennt. Der Assembler seinerseits kann die absolute Adresse nicht errechnen, da der Inhalt der angegebenen Adresse SHOWADR+1 bei der Assemblierung nicht unbedingt bekannt sein muß und außerdem - falls es sich um Speicherzellen im RAM handelt - zur Laufzeit verändert sein könnte. Daher ist der Umweg über das U-Register zur Berechnung der Startadresse notwendig. Alternativ könnte man die Adresse “zu Fuß“ berechnen (siehe oben) und den Sprung mit einer absoluten Adresse versehen: JMP >$F4E4.

2. Folgende Anpassungen der Routine sind notwendig:

- Der Wert für ADR muß $75 sein, da die anderen Monitorroutinen zur Speicherbearbeitung auch auf diese Adresse zugreifen.
- Da der Monitor vor dem Sprung in die Routine keine Rücksprungadresse in die Interpreterschleife auf dem Stack ablegt, muß sie nicht mit RTS, sondern mit JMP $E325 abgeschlossen werden. Hier beginnt die Interpreterschleife des Monitors mit der Tastaturabfrage.
- Der Wert in den Speicherstellen $51, $52, die die Startadresse für die ADDRESS-Routine enthalten, muß gegen die Startadresse Ihrer Routine ausgetauscht werden.

Sie werden feststellen, daß Sie zur Anwahl der nächsten Monitorfunktion die entsprechende Funktionstaste zweimal betätigen müssen. Das ist notwendig, weil die Tastaturabfrage des Monitors andere Tastencodes liefert als die HALTKEY-Routine.

3.

```
; Routine ADDRESS:  Adresse editieren
; Ausgabe der Kennung und Holen der Adresse

1075                    ADR       EQU     $75

;                                 ORG     $1000
1000    34 33       ADDRESS:      PSHS    A,X,Y,CC      ; Register retten
1002    BD F1 10                  JSR     CLRDISP       ; Anzeige löschen
1005    CC 77 5E                  LDD     #$775E        ; 7-Segmentcode für “Ad"
1008    8E 00 06                  LDX     #$6           ;  im Operationsfeld
100B    BD F1 16                  JSR     SHOWD         ;  anzeigen
100E    10 9E 75                  LDY       <ADR  ; zu editierende Adresse holen
        ; das Zeichen "<" kennzeichnet die DP-Register-Adressierung
```

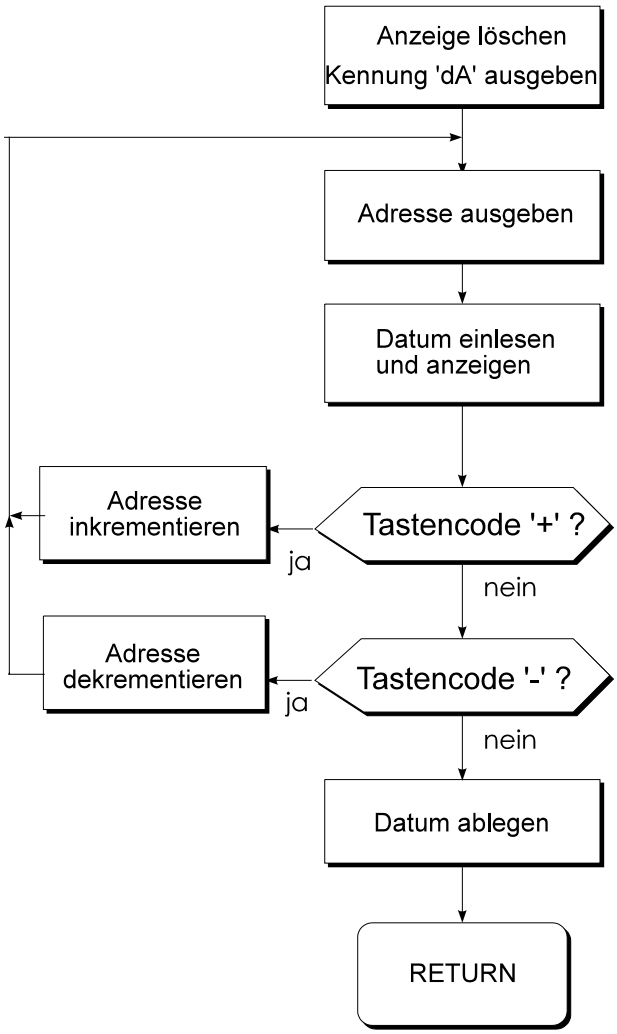
```
        ; Adresse editieren
1011    E6 A4       W:      LDB     ,Y          ; zugehöriges Datum in Akku B
1013    8E 00 00            LDX     #0          ;    im Datenfeld
1016    BD F1 20            JSR     SHOWB7SG    ;    anzeigen
1019    8E 00 02            LDX     #2          ; Anzeige im Adreßfeld
101C    12                  NOP
101D    BE F1 57            LDU     SHOWADR+1   ; Startadresse aus Sprung
                                                ; tabelle.
1020    AD 64               JSR     4,U         ;Aufruf ohne Löschen von Y

        ; Ergebnis von SHOWADR interpretieren
1022    C1 81               CMPB    #$81        ; letzte Taste
1024    22 08               BHI     R           ;    >$81?, dann fertig
1026    27 02               BEQ     $2          ; =$81("-")?, dekrementieren
1028    31 22               LEAY    2,Y         ; Zeiger inkrementieren (um2,
                                                ; da wieder dekrementiert wird)
102A    31 3F               LEAY    -1,Y        ; Zeiger dekrementieren
102C    20 E3               BRA     W           ; und erneut editieren
        ; falls Ergebnis nicht "+" oder "-", Ende
102E    10 9F 75    R:      STY     <ADR        ; neue Adresse ablegen
1031    35 33               PULS    A,X,Y,CC    ; Register restaurieren
1033    39                  RTS     ; zurück zum Monitor
```

4. Wir gehen nach derselben Methode vor, die bei der ADDRESS-Routine angewandt wurde. Die Anforderungen an die DATA-Routine:

- Löschen der Anzeige und Ausgabe der Kennung "Da" im Operationsfeld,
- Ausgabe des zu editierenden Datums und der zugehörigenAdresse,
- Eingabe des neuen Datums bzw. Inkrementieren/Dekrementieren der Adresse mit +/-,
- Rückkehr aus der Routine, wenn eine andere Taste gedrückt wurde als eine Ziffer oder "+" bzw. "-".

Das Flußdiagramm:



Das Programm:

Routine **DATA**

Eingaberegister:
Ausgaberegister: B: enthält neue Funktionstaste
Funktion: Das Datum, auf das der Adressenzeiger des Monitors zeigt, kann editiert werden. Mit den Tasten '+', '-' kann die Adresse inkrementiert/dekrementiert werden.

```
1        CLRDISP    EQU    $F110          ; Anzeige löschen
2        SHOWD      EQU    $F116          ; D in die Anzeige
3        SHOWD7SG   EQU    $F123          ; D hexadezimal in die Anzeige
4        SHOWDATA   EQU    $F150
5        ADR        EQU    $75

14   0000   34 73          PSHS   A,X,Y,U,CC    ; Register retten
15   0002   BD F1 10       JSR    CLRDISP       ; Anzeige löschen
16   0005   CC 5E 77       LDD    #$5E77        ; "dA"
17   0008   8E 00 06       LDX    #6            ;  im Operationsfeld
```

```
18   000B  BD F1 16        JSR   SHOWD          ;  anzeigen
19   000E  10 9E 75        LDY   <ADR       ; Adresse aus Systembereich
20   0011  1F 20     LOOP: TFR   Y,D          ;  nach D
21   0013  8E 00 02        LDX   #2           ;  im Adreßfeld
22   0016  BD F1 23        JSR   SHOWD7SG     ;  anzeigen
23   0019  A6 A4           LDA   ,Y           ; zu editierendes Datum
24   001B  8E 00 00        LDX   #0           ;  im Datenfeld
25   001E  FE F1 51        LDU   SHOWDATA+1   ; Startadresse + 2
26   0021  AD 42           JSR   2,U          ;  editieren
27   0023  A7 A4           STA   ,Y           ;  und wieder ablegen
28   0025  C1 81           CMPB  #$81         ; letzte Taste "+" oder "-"?
29   0027  22 08           BHI   ENDE         ; falls nein
30   0029  27 02           BEQ   $2           ; letzte Taste = "-"
31   002B  31 22           LEAY  2,Y        ; "+": Adresse+2 (da wieder
33                                           ;  dekrementiert wird)
34   002D  31 3F           LEAY  -1,Y       ; "-": Adresse dekrementieren
35   002F  20 E0           BRA   LOOP       ; nächstes Datum editieren
36         ;
37   0031  10 9F 75  ENDE: STY   <ADR         ; neue Adresse ablegen
38   0034  35 73           PULS  A,X,Y,U,CC   ; Register restaurieren
39   0036  39              RTS                ; zurück
```

Die Tasten "+“ und “-“ wirken hier genauso wie bei der ADDRESS-Routine auf die Adresse und nicht auf das zu editierende Datum ein. Das ist sinnvoll, da im allgemeinen aufeinanderfolgende Speicherplätze editiert werden, die so bequem angewählt werden können. Es kommt dagegen selten vor, daß der Inhalt einer Speicherzelle inkrementiert bzw. dekrementiert werden soll.

Zu P3.2-4:

```
   ; Routine INSERT
   ; Die 255 Bytes über der aktuellen Adresse werden um einen Platz nach oben
   ; der freigewordene Platz wird mit $FF. Dann wird die Adresse inkrementiert und in
   ; die DATA-Routine gesprungen, so daß das gewünschte Datum eingegeben
   ; werden kann.
   ;
EC0E                       ORG   $EC0E
   ;
EC0E  34 17        INSERT: PSHS  A,B,X,CC     ; Register retten
EC10  BD F1 10             JSR   CLRDISP      ; Anzeige löschen
EC13  8E 00 06             LDX   #$06         ; S6 selektieren
EC16  CC 06 54             LDD   #$0654       ; „in“
```

```
EC19  BD F1 16          JSR   SHOWD       ;  ausgeben
EC1C  9E 75             LDX   <ADO        ; aktuelle Adresse
EC1E  C6  01            LDB   #1          ;  um 1
EC20  3A                ABX               ;  erhoehen
EC21  9F  75            STX   <ADO        ;  und speichern
EC23  50                NEGB              ; B  =  $FF
EC24  3A                ABX             ; X zeigt auf höchste Adresse
EC25  A6  82      ILOOP: LDA   ,-X          ; X dekrementieren, Datum
EC27  A7  01            STA   1,X       ; 1 Platz nach oben schieben
EC29  9C75              CMPX  <ADO        ; aktuelle Adresse erreicht?
EC2B  26  F8            BNE   ILOOP       ; wenn nicht, wiederholen
EC2D  86  FF            LDA   #$FF        ; $FF in
EC2FA7  84              STA   ,X        ; freigewordeneSpeicherplatz
EC31  35  17            PULS  A,B,X,CC    ; Register restaurieren
EC33  13                SYNC              ; Ende
```

Zu P3.2-5:

Da die Routine bereits in Bild 3.2-9 vollständig beschrieben wird, wird hierzu kein Lösungsvorschlag angegeben.

Zu P3.2-6:

Das folgende Listing können Sie erhalten, indem Sie aus dem Praktikumsrechner (mit Hilfe des Programms TERM9.EXE) oder aus dem 6809-Emulator den Speicherbereich $FE00-$FEFF in eine Datei übertragen und diese mit dem Programm DS9.EXE disassemblieren. Studieren Sie es aufmerksam unter Beachtung der Kommentare.

```
Adresse  OpCode       Mnemocode       Bemerkung
   1
   2 009B             ZK        EQU   $9B       ; Zeilenkennung
   3 009C             ZL        EQU   $9C       ; Zeilenlänge
   4 009D             PS        EQU   $9D       ; Prüfsumme
   5 009E             LO        EQU   $9E       ; Lade-Offset
   6 F008             DRACIA    EQU   $F008     ; Datenregister der ACIA
   7 F009             SRACIA    EQU   $F009     ; Statusregister der ACIA
   8 F110             CLRDISP   EQU   $F110
   9 F116             SHOWD     EQU   $F116
  10 F120             SHOWB7SG  EQU   $F120
  11 F156             SHOWADR   EQU   $F156
  12 E38F             MONITOR   EQU   $E38F
  13
```

```
14 FE00                        ORG   $FE00
15 FE00   BD FE CE  V24LOAD    JSR   OFFSET     ; 'oF' ausgeben,
                                                ; Lade-Offset einlesen
16 FE03   BD F1 10     JSR     CLRDISP          ; Anzeige löschen
17 FE06   7F 00 9B             CLR   >ZK        ; Zeilenkennung löschen
18 FE09   7F 00 9D             CLR   >PS        ; Prüfsumme löschen
19 FE0C   BD FE 46  LOOP_Z     JSR   WAIT_Z     ; auf Zeilenanfang warten
20 FE0F   8D 48                BSR   INBYTE  ; Byte empfangen und umwandeln
21 FE11   F7 00 9C             STB   >ZL        ; als Zeilenlänge abspeichern
22 FE14   8D 43                BSR   INBYTE  ; Byte empfangen und umwandeln
23 FE16   1F 98                TFR   B,A      ; als Zeilenanfangsadresse ZAAH
                                                ;  nach Akku A
24 FE18   8D 3F                BSR   INBYTE ; ZAAL empfangen und umwandeln
25 FE1A   F3 00 9E             ADDD  >LO        ; Lade-Offset zu ZAA addieren
26 FE1D   1F 01                TFR   D,X        ; Zeilenanfangsadresse nach X
27 FE1F   8D 38                BSR   INBYTE  ; Byte empfangen und umwandeln
28 FE21   F7 00 9B             STB   >ZK     ; als Zeilenkennnung abspeichern
29 FE24   8D 66                BSR   INDATA    ; Daten der Zeile empfangen und
30                                              ; im Puffer mit X abspeichern

31 FE26   8D 31                BSR   INBYTE  ; Byte empfangen und umwandeln
32 FE28   7D 00 9D             TST   >PS        ; Prüfsumme testen
33 FE2B   26 71                BNE   FEHLER; falls ungleich $00, Fehlerausgabe
34 FE2D   7D 00 9B             TST   >ZK        ; letzte Zeile ?
35 FE30   27 DA                BEQ   LOOP_Z    ; nein: Zeilenschleife bearbeiten
36
37 FE32   8E 00 06  ENDE       LDX   #$0006     ; Anzeigestelle S6 anwählen
38 FE35   CC 79 54             LDD   #$7954     ; Code für 'En' nach Akku D
39 FE38   BD F1 16             JSR   SHOWD      ; darstellen in der Anzeige
40 FE3B   30 1E                LEAX  -2,X       ; X um 2 dekr., S4 anwählen
41 FE3D   CC 5E 79             LDD   #$5E79     ; Code für 'dE' nach Akku D
42 FE40   BD F1 16             JSR   SHOWD      ; darstellen in der Anzeige
43 FE43   7E E3 25             JMP   MONITOR    ; Sprung zum Monitor
44
45 FE46   BD FE 4E  WAIT_Z     JSR   INCHAR     ; Zeichen über V24 empfangen
46 FE49   C1 3A                CMPB  #$3A       ; Vergleich mit ':'
47 FE4B   26 F9                BNE   WAIT_Z     ; kein Zeilenanfang gefunden
48 FE4D   39                   RTS              ; Rücksprung
49 FE4E   F6 F0 09  INCHAR     LDB   SRACIA   ; Lesen des ACIA-Statusregisters
50 FE51   C5 08                BITB  #$08     ; Testen des IRQ-Flags (kann entf.)
51 FE53   27 F9                BEQ   INCHAR  ; Schleife, bis IRQ (dann hier BPL)
52 FE55   F6 F0 08             LDB   DRACIA     ; Lesen des ASCII-Zeichens
53 FE58   39                   RTS              ; Rücksprung
54 FE59   34 12     INBYTE     PSHS  A,X        ; A,X auf dem Stack retten
55 FE5B   8D 1B                BSR   WACHAR     ; auf 1. Zeichen warten
```

```
                                                    ;  und umwandeln
56 FE5D  58                    LSLB          ; 1. Hex-Tetrade links verschieben
57 FE5E  58                    LSLB
58 FE5F  58                    LSLB
59 FE60  58                    LSLB
60 FE61  34 04                 PSHS  B       ; 1. Tetrade auf dem Stack sichern
61 FE63  8D 13                 BSR   WACHAR  ; auf 2. Zeichen warten
                                             ;   und umwandeln
62 FE65  EA E0                 ORB   ,S+     ; mit 1. Tetrade verodern und
                                             ;   Stack restaurieren
63 FE67  1F 98                 TFR   B,A     ; Byte nach Akku A bringen
64 FE69  BB 00 9D              ADDA  >PS     ; zur Prüfsumme addieren
65 FE6C  B7 00 9D              STA   >PS     ; und abspeichern
66 FE6F  8E 00 00              LDX   #$0000  ; S0 der Anzeige anwählen
67 FE72  BD F1 20              JSR   SHOWB7SG; Byte anzeigen
68 FE75  35 12                 PULS  A,X     ; A,X restaurieren
69 FE77  39                    RTS           ; Rücksprung
70 FE78  BD FE 4E  WACHAR      JSR   INCHAR  ; ASCII-Zeichen einlesen
71 FE7B  BD FE 7F              JSR   ASC2HEX ; ASCII hexadezimal umwandeln
72 FE7E  39                    RTS           ; Rücksprung
73 FE7F  C0 30     ASC2HEX     SUBB  #$30    ; Offset von $30 abziehen
74 FE81  C1 0A                 CMPB  #$0A    ; Ziffer oder Buchstabe ?
75 FE83  25 02                 BLO   ZIFFER  ; Sprung, falls Ziffer
76 FE85  C0 07                 SUBB  #$07    ; Buchstaben-Offset ($07) abziehen
77 FE87  C1 10        ZIFFER   CMPB  #$10    ; Buchstabe aus (A,B,...,F)
78 FE89  24 18                 BHS   FEHLER+5; Fehlermeldung
79 FE8B  39                    RTS           ; Rücksprung
80 FE8C  34 02        INDATA   PSHS  A       ; A retten
81 FE8E  B6 00 9C              LDA   >ZL     ; Zeilenlänge laden
82 FE91  27 08                 BEQ   RUECK   ; letztes Datum gelesen
83 FE93  BD FE 59     EING     JSR   INBYTE  ; Byte empfangen
84 FE96  E7 80                 STB   ,X+     ; im Puffer über X abspeichern
85 FE98  4A                    DECA          ; Datenzähler dekrementieren
86 FE99  26 F8                 BNE   EING    ; neues Datum empfangen
87 FE9B  35 02        RUECK    PULS  A       ; A restaurieren
88 FE9D  39                    RTS           ; Rücksprung
89 FE9E  CC 73 6D     FEHLER   LDD   #$736D  ; 'PS' Fehlermeldung
90 FEA1  20 03                 BRA   WEITER
91 FEA3  CC 74 79              LDD   #$7479  ; 'hE' Fehlermeldung
92 FEA6  BD F1 10     WEITER   JSR   CLRDISP ; Anzeige löschen
93 FEA9  8E 00 06              LDX   #$0006  ; S6 in der Anzeige anwählen
94 FEAC  34 06                 PSHS  A,B     ; A,B retten
95 FEAE  CC 71 79              LDD   #$7179  ; 'FE' in S7,S6
96 FEB1  BD F1 16              JSR   SHOWD   ;  ausgeben
97 FEB4  30 1E                 LEAX  -2,X    ; X:=X-2, S4 anwählen
98 FEB6  CC 74 38              LDD   #$7438  ; 'hL' in S5,S4
```

```
 99 FEB9   BD F1 16            JSR   SHOWD      ;  ausgeben
100 FEBC   30 1E               LEAX  -2,X       ; X:=X-2, S2 anwählen
101 FEBE   CC 79 50            LDD   #$7950     ; 'Er' in S3,S2
102 FEC1   BD F1 16            JSR   SHOWD      ;  ausgeben
103 FEC4   30 1E               LEAX  -2,X       ; X:=X-2, S0 anwählen
104 FEC6   35 06               PULS  A,B        ; A,B restaurieren
105 FEC8   BD F1 16            JSR   SHOWD      ; 'PS' oder 'hL' ausgeben
106 FECB   7E E3 25            JMP   MONITOR    ; Sprung in den Monitor
107 FECE   BD F1 10   OFFSET   JSR   CLRDISP    ; Anzeige löschen
108 FED1   8E 00 00            LDX   #$0000     ; S0 in der Anzeige anwählen
109 FED4   CC 5C 71            LDD   #$5C71     ; Code 'oF' nach Akku D
110 FED7   BD F1 16            JSR   SHOWD      ; ausgeben
111 FEDA   30 02               LEAX  2,X        ; X:=X+2, S2 anwählen
112 FEDC   BD F1 56            JSR   SHOWADR    ; Offset einlesen
113 FEDF   10 BF 00 9E         STY   >LO        ; Abspeichern als Lade-Offset
114 FEE3   30 04               LEAX  4,X        ; S6 anwählen
115 FEE5   CC 38 5C            LDD   #$385C     ; Code ‘Lo’ nach D
116 FEE8   BD F1 16            JSR   SHOWD      ; ausgeben
117 FEEB   39                  RTS              ; Rücksprung
```

Machen Sie sich bitte gründlich mit der Funktion ASC2HEX (ab $FE7F) vertraut.

Zu P3.2-7:

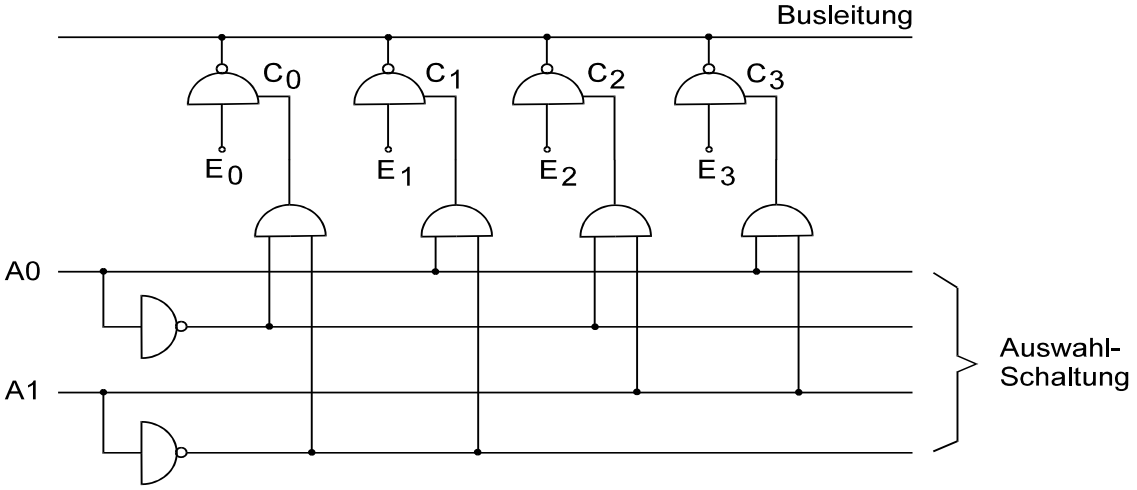
Um die Funktionstaste ‘F1’ mit der der FILL-Routine neu zu belegen, müssen Sie lediglich die Startadresse $1820 in die Speicherzellen $006C,$006D eintragen (s. KE1, Abschnitt 1.2.1).

Diese Seite bleibt aus technischen Gründen frei !

Lösungsvorschläge zu den Selbsttestaufgaben

Zu S3.1-1:

Die Ansteuerung der "Control"-Eingänge C3-C0 geschieht durch die Ausgänge eines 2-auf-4-Demultiplexers. Die Adreßsignale A0 und A1 dienen zur Selektion eines seiner 4 Ausgänge.



Zu S3.1-2:

(Hinweis: Zur besseren Lesbarkeit werden hier die Indizes der Adreßleitungen tiefgestellt, also z.B. hier A_{15} anstelle A15 wie im Schaltplan.)

1.

$CS = A_{15}A_{14}A_{13}A_{12}\overline{(A_{11} v A_{10})}\,\overline{(A_9 v A_8)}\,\overline{(A_7 v A_6)}$

$= A_{15}A_{14}A_{13}A_{12}\overline{A_{11}}\,\overline{A_{10}}\;\overline{A_9}\;\overline{A_8}\;\overline{A_7}\;\overline{A_6}$

$= A_{15}A_{14}A_{13}R$ mit

$R = A_{12}\,\overline{A_{11}}\,\overline{A_{10}}\,\overline{A_9}\;\overline{A_8}\;\overline{A_7}\;\overline{A_6} = A_{12}\overline{(A_{11} v A_{10} v A_9 v A_8 v A_7 v A_6)}$

CS selektiert damit den Adreßbereich $F000 - $F03F. Bild 3.1-1 können Sie entnehmen, daß in diesem Bereich die Adressen aller Peripherie-Komponenten untergebracht sind.

2. Für die Funktion des 3-auf-8-Demultiplexers 74138 (IC38 im Gesamtschaltplan) gilt laut Datenblatt im Anhang für die Ausgänge Y_j , j=0,..,7:

Y_i=0 genau dann, wenn für die Gate-Eingänge G1, G2A, G2B und für die Auswahleingänge (*Select*) A,B,C die folgenden Bedingungen gelten:

G1=1, G2A=0, G2B=0, $A*2^0+B*2^1+C*2^2 = i$.

Dem Gesamtschaltplan entnimmt man:

$G1 = \overline{CS}$, G2A = G2B = GND, A = A_{13}, B = A_{14}, C = A_{15}.

Damit folgt für die nicht - invertierten Auswahlsignale CS0 - CS :

$CS0 = \overline{A_{15}}\ \overline{A_{14}}\ \overline{A_{13}}\ \overline{CS} = \overline{A_{15} v A_{14} v A_{13} v A_{15} A_{14} A_{13} R}$

$= \overline{A_{15} v A_{14} v A_{13}} = \overline{A_{15}}\ \overline{A_{14}}\ \overline{A_{13}}$ Adreßbereich: $0000 - $1FFF

Komponente: 8-kbyte-RAM

$CS1 = \overline{A_{15}}\ \overline{A_{14}}\ A_{13}$ Adreßbereich: $2000 - $3FFF

Komponente: 1. Speichererweiterung

$CS2 = \overline{A_{15}}\ A_{14}\ \overline{A_{13}}$ Adreßbereich: $4000 - $5FFF

Komponente: 2. Speichererweiterung

$CS3 = \overline{A_{15}}\ A_{14}\ A_{13}$ Adreßbereich: $6000 - $7FFF

$CS4 = A_{15}\ \overline{A_{14}}\ \overline{A_{13}}$ Adreßbereich: $8000 - $9FFF

$CS5 = A_{15}\ \overline{A_{14}}\ A_{13}$ Adreßbereich: $A000 - $BFFF

$CS6 = A_{15}\ A_{14}\ \overline{A_{13}}$ Adreßbereich: $C000 - $DFFF

Da an der Kette von UND-Gattern des IC35 keine Brücken gesetzt sind, stimmt das Signal CS7 mit dem Ausgang Y_7 des Demultiplexers 74138 überein. Also folgt:

$CS7 = A_{15}A_{14}A_{13}\overline{CS} = A_{15}A_{14}A_{13}\overline{A_{15}A_{14}A_{13}R}$

$= (A_{15}A_{14}A_{13} v \overline{R})A_{15}A_{14}A_{13} = A_{15}A_{14}A_{13}R$

$= A_{15}A_{14}A_{13}\ (\overline{A_{12}A_{11}A_{10}\ A_9\ A_8\ A_7\ A_6}$

$= A_{15}A_{14}A_{13}\ (\overline{A_{12}} v A_{12}\ (A_{11} v A_{10} v A_9 v A_8 v A_7 v A_6))$

$= A_{15}A_{14}A_{13}\overline{A_{12}} v A_{15}A_{14}A_{13}A_{12}(A_{11} v A_{10} v A_9 v A_8 v A_7 v A_6)$

Adreßbereich: $E000 - $EFFF und $F040 - $FFFF

Komponente: 8-kbyte - EPROM

Durch das Signal CS wird aus dem Bereich des EPROM-Speichers der Adreßbereich $F000 - $F03F der Peripherie ausgeblendet. CS wird nun zur Erzeugung der weiteren Auswahlsignale CS8 - CS13 herangezogen. Für das Signal am Gate-Eingang G des 2-auf-4-Demultiplexers 74139 (IC25) ergibt sich aus dem Schaltplan:

$G = A_5 v A_4 v \overline{CS} = \overline{\overline{A_5}\ \overline{A_4}}\ CS$. Damit folgt:

$CS8 = \overline{A_3}\ \overline{A_2}\ \overline{G} = \overline{A_3}\ \overline{A_2}\ \overline{A_5}\ \overline{A_4}\ CS$

$= A_{15}A_{14}A_{13}A_{12}\overline{A_{11}}\overline{A_{10}}\overline{A_9}\ \overline{A_8}\ \overline{A_7}\ \overline{A_6}\ \overline{A_5}\ \overline{A_4}\ \overline{A_3}\ \overline{A_2}$

Adreßbereich**:** $F000 - $F003

Komponente: Parallel-Schnittstelle

$CS9 = \overline{A_3}\ A_2\ \overline{G} = A_3\ A_2\ \overline{A_5}\ \overline{A_4}\ CS,$ Adreßbereich: $F004-$F007

$CS10 = A_3\ \overline{A_2}\ \overline{G} = A_3\ \overline{A_2}\ \overline{A_5}\ \overline{A_4}\ CS\ ,$ Adreßbereich: $F008-$F00B

Komponente: V24-Schnittstelle

$CS11 = A_3\ A_2\ \overline{G} = A_3\ A_2\ \overline{A_4}\ \overline{A_5}\ CS$, Adreßbereich: $F00C-$F00F
Komponente: Kassettenrecorder-Interface (wird nicht mehr unterstützt)

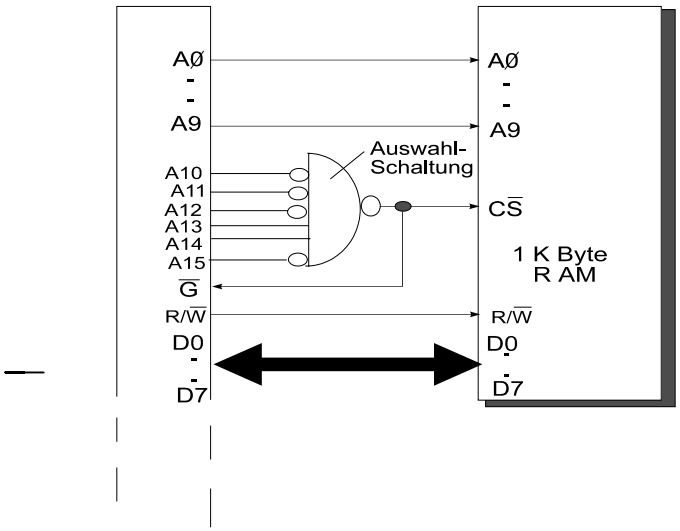
$CS12 = \overline{\overline{A_4}\ \overline{CS} \vee A_5} = A_4\ \overline{A_5}\ CS$, Adreßbereich: $F010-$F01F
Komponenten: Tastatur und Timer ($F018-$F01F)

$CS13 = \overline{\overline{A_5}\ \overline{CS} \vee A_4} = \overline{A_4}\ A_5\ CS$, Adreßbereich: $F020-$F02F
Komponente: 7-Segment-Anzeige

Zu S3.1-3:

1. Der Baustein 74541 enthält 8 nicht-invertierende Tristate-Leitungstreiber mit Schmitt-Trigger-Eingängen. Die Steuereingänge aller 8 Treiber sind mit-einander verbunden. Sie werden aktiviert, wenn an beiden Gate-Eingängen G1 und G2 des Bausteins L-Potential angelegt wird. Dieser Baustein kann für unidirektionale Busleitungen benutzt werden.
Der Baustein 74245 enthält 8 bidirektionale Tristate-Leitungstreiber mit Schmitt-Trigger-Eingängen. Für jede Leitung sind zwei Treiber antiparallel geschaltet. Von ihnen wird durch den Eingang Dir jeweils genau einer angesprochen.
Die Aktivierung des gesamten Bausteins erfolgt durch L-Potential am Gate-Eingang G. Dieser Baustein eignet sich besonders zum Einsatz an bidirektionalen Datenbussen.

2. Externe Speichererweiterung



Zu S3.1-4:

(Beachten Sie die verschiedenen Bezeichnungen der Ein- und Ausgänge im Datenblatt und dem Gesamtschaltplan.)

1. Datenbuswort: D7 D6.....D1 D0 = $A7 = 1010 0111
   IC29: Dateneingänge : DI1=DI2=DI3=1, DI4=0,
   Steuereingänge: ME=$\overline{S}$=0, WE=$\overline{W}$=0,
   Adreßeingänge : A=B=C=1, D=0 .
   IC31: Dateneingänge : DI1=DI3=0, DI2=DI4=1,
   Steuereingänge: ME=0, WE=0,
   Adreßeingänge : A=B=C=1, D=0 .
2. Der benutzte Speicherbaustein 74289 invertiert jedes eingeschriebene Datum. Also erhält man beim Lesen der Speicherzelle 7:
   IC29: Datenausgänge : DO1=DO2=DO3=0, DO4=1 .
   IC31: Datenausgänge : DO1=DO3=1, DO2=DO4=0,
   also das Datenbuswort: D7 D6.....D1 D0 = $58 = 0101 1000 .
3. Die Inversion der Information im Speicher wird durch den invertierenden Treiber-Baustein 74540 (IC1) auf der Anzeige-Platine (s. Gesamtschaltplan im Anhang) rückgängig gemacht.

Zu S3.1-5:

Die Rücksetztaste C wirkt direkt auf einen Eingang der CPU 6809. Sie setzt den Prozessor hardwaremäßig in den Grundzustand zurück. Da sie in jedem Prozessorzustand wirksam sein muß, darf sie nicht erst vom Prozessor in einem Programm abgefragt werden.
Ähnlich verhält es sich mit der Break-Taste F4. Diese finden Sie am Eingang 40 (CA1) des Port-Bausteins MC6821 in der Parallel-Schnittstelle. Sie bewirkt über die IRQ-Leitung eine Hardware-Unterbrechung des µPs. Voraussetzung dafür ist, daß der Port-Baustein solche Interrupts durchläßt. In KE4 wird Ihnen gezeigt, wie man durch das Setzen eines bestimmten Bits solche Interrupts verhindern kann.

Zu S3.1-6:

1. Durch die niederwertige Tetrade A3-A0 des Adreßbusses wird der Ausgang Y_7 des Demultiplexers angesprochen. Dieser ist jedoch in der Ansteuerschaltung der Tastatur nach Bild 3.1-4 nicht benutzt worden, so daß die Ausgabe der Adresse $F017 keine Wirkung auf die Tastatur hat.
2. Die 3. Spalte der Tastatur von links im Bild 3.1-4 wird durch den Ausgang Y_2 des Demultiplexers angewählt. Daher muß die Adresse $F012 auf den Adreßbus gegeben werden.
   Gleichzeitiges Drücken der Tasten ‘5’ und ‘9’ führt auf den Datenbusleitungen D1 und D2 zu einem "Kurzschluß" gegen Masse. Deshalb wird der Wert $X9 = xxxx 0110 eingelesen. (X = x = unbestimmt.)

Zu S3.2-1:

Die Routinen der Bibliothek eines Entwicklungssystems sind auf dem Zielsystem nicht verfügbar, sie könnten also nicht aufgerufen, sondern müßten in das Programm hineinkopiert werden (wie Macros in Assemblerprogrammen). Dazu müssen jedoch mehrere Voraussetzungen erfüllt sein:

1. Es müssen Anfangs- und Endadresse der Routine bekannt sein. (Die Aufrufadresse muß nicht die Anfangsadresse sein !).
2. Alle Unterroutinen, die von der Hauptroutine aufgerufen werden, müssen mitkopiert werden.
3. Sämtliche Routinen müssen lageunabhängig geschrieben sein oder angepaßt werden. Es ist also sehr fraglich, ob der Aufwand gerechtfertigt ist, oder ob es nicht günstiger ist, die Routinen vollständig (neu) zu programmieren.

Zu S3.2-2:

Nach einem Reset ist der Inhalt des 'S'-Registers nicht definiert, der Stapelzeiger zeigt also auf eine beliebige Adresse. Liegt sie im Bereich eines Anwenderprogramms, so würde dieses durch das Ablegen der Register des Prozessors durch eine Interruptroutine zerstört. Falls sie im ROM-Bereich oder in einem nicht belegten Adreßbereich liegt, könnten die Register gar nicht gesichert werden, und eine Programmfortsetzung nach Beendigung der Unterbrechung wäre unmöglich. Auf keinen Fall ist der einwandfreie Betrieb des Rechners gewährleistet.

Zu S3.2-3:

Die vom Empfänger berechnete Prüfsumme PS_E über alle Bytes einer fehlerfrei übertragenen Zeile ergibt PS_E = \$00, denn für jede Zeile $Z=(B_1,B_2,...,B_n)$ gilt:

$$PS_E = \sum^{n} B_i = \sum^{n-1} B_i + PS_S = \sum^{n-1} B_i + \left(\sum^{n-1} B_i\right)_{2k} = \sum^{n-1} B_i + \left(2^8 - \sum^{n-1} B_i\right) = \$00.$$

Zu S3.2-4:

Die Verfahrensweise beim Programmabbruch ist im wesentlichen die gleiche. Bei der Beendigung der BREAKPOINT-Routine muß jedoch (neben dem Austausch des SWI-Befehls im Programm) das PC-Register um eins vermindert werden, da der PC auf die dem SWI-Befehl folgende Adresse zeigt und nicht mehr auf den ursprünglichen OpCode des Programms. Bei der TRACE-Routine muß außerdem die Interrupt-Anforderung gesperrt werden.

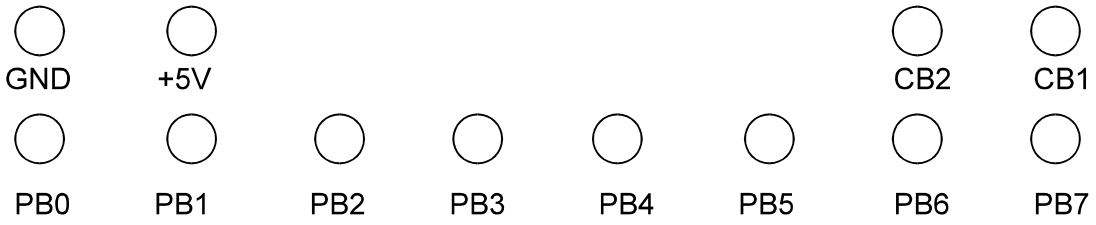
Diese Seite bleibt aus technischen Gründen frei !

Anhang A: Interruptvektor-Tabelle

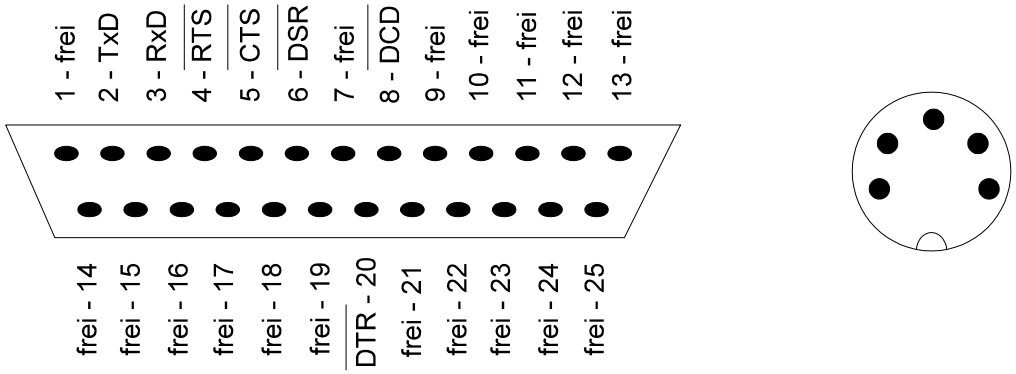
Adressen	Quelle, Baustein	Vorbelegung	Funktion
$002C, $002D	Timer #1, MC6840	$E667	RTI
$002E, $002F	Timer #3, MC6840	$E667	RTI
$0030, $0031	Timer #2, MC6840	$E670	Trace
$0032, $0033	NMI Steckerleiste	$E667	RTI
$0034, $0035	FIRQ Steckerleiste	$E667	RTI
$0036, $0037	CA1, MC6821, Port PA	$E520	Break
$0038, $0039	CB1, MC6821, Port PB	$E667	RTI
$003A, $003B	CA2, MC6821, Port PA	$E667	RTI
$003C, $003D	CB2, MC6821, Port PB	$E667	RTI
$003E, $003F	IRQ, MC6850	$E667	RTI
$0040, $0041	IRQ, R6551, V.24	$E667	RTI
$0042, $0043	IRQ Steckerleiste	$E667	RTI
$0044, $0045	SWI1 (SWI)	$EC40	Breakpoint
$0046, $0047	SWI2 aus ROM	$E4E0	GO-Routine
$0048, $0049	SWI2 aus RAM	$E667	RTI
$004A, $004B	SWI3 aus ROM	$E640	TRACE-Routine
$004C, $004D	SWI3 aus RAM	$E667	RTI

Anhang B: Steckerbelegungen des Praktikumsrechners

a) Belegung der vergoldeten Buchsen



b) V.24-Schnittstelle: 25-polige Steckerleiste 5-polige DIN-Buchse



c) 64-polige Steckerleiste