Einführung in die Mikroprozessor-Programmierung mit dem

tv-computersystem

6800

Franz MORAT KG

Einführung in die Mikroprozessor-Programmierung mit dem

tv-computersystem

6800

Franz MORAT KG
7821 Eisenbach/Hochschwarzwald 1

Vertrieb:
SVEMOR-Messtechnik GmbH.
7801 Ballr.-Dottingen
Fernsp. (07634) 751 u. 779
Fernsp. 772 989 ysdo

November 1977 3. korrigierte Auflage

Vorwort

Wer Mikroprozessoren einsetzen möchte, muß sie beherrschen. Dies setzt gute Kenntnisse der Hard- und Softwareeigenschaften voraus. Das vorliegende Buch unterstützt Sie bei der Einarbeitung in die Mikroprozessorprogrammierung mit dem TV-Computersystem 6800. Durch viele Beispiele, die mit dem Gerät unmittelbar nachvollziehbar sind, erschließen sich auf anschauliche Weise die Grundlagen der Programmierung auf Maschinenebene. Die Hardwareeigenschaften des Mikroprozessors werden nicht behandelt. Es sei hier auf die Literatur /1/ verwiesen.

Zu Ihrer Unterhaltung und als Anregung für die vielfältigen Möglichkeiten des Geräts sind die Beispielprogramme in Kapitel 7 gedacht. Nachdem Sie die Bedienungsanleitung studiert haben, können Sie diese Programme ohne weitere Vorkenntnisse laufen lassen. Später können Sie sicher selbst weitere Spiele programmieren. Schicken Sie uns Ihre Spielprogramme zu.

Wir wünschen Ihnen viel Spaß und Erfolg!

Inhalt

1. Grundbegriffe der Datenverarbeitung

1.1. Allgemeines

Bei den elektronischen Datenverarbeitungsanlagen gibt es heute ein weites Spektrum. So unterschiedlich Leistung und Ausbau, angefangen bei den Großrechnern über die Mittlere Datentechnik bis zum neuesten technischen Produkt, dem Mikrocomputer, auch sein mag, es liegen immer die gleichen Prinzipien zugrunde. Wir behandeln die wichtigsten Grundbegriffe soweit sie im Rahmen dieses Buches nötig sind.

Der Einsatz elektronischer Datenverarbeitung setzt zwei Komponenten voraus: Rechenanlage und Programm.

Für die gerätetechnische Ausstattung ist der Begriff Hardware üblich, alle zum Betrieb benötigten Programme bezeichnet man als Software.

Ein Rechner verarbeitet Daten in der Weise, wie es ihm ein Programm vorschreibt. Durch verschiedene Programme kann ein Rechner zur Lösung ganz unterschiedlicher Aufgaben eingesetzt werden. Es seien hier nur die Bereiche kommerzielle Datenverarbeitung, technisch-wissenschaftliche DV und Prozeßdatenverarbeitung genannt.

Die Stärken des Computers sind seine universelle Einsetzbarkeit, die Fähigkeit, während des Programmablaufs Entscheidungen zu treffen, die hohe Verarbeitungsgeschwindigkeit und die große Fehlersicherheit. Er löst nicht von sich aus Probleme. Der Mensch muß sie lösen und in Form eines Programmes in einer für den Rechner verständlichen Weise aufbereiten.

Anweisungen, die im Rechner Operationen auslösen, heißen Befehle. Zum Befehlsvorrat gehören beispielsweise arithmetische Befehle, Transportbefehle, Vergleichsbefehle oder Sprungbefehle. Der Befehlsvorrat ist rechnerspezifisch, d. h. er variiert bei den verschiedenen Rechnertypen in Einzelheiten. Eine vom Rechner zu bearbeitende Aufgabe ist nun soweit in Einzelschritte zu zerlegen, daß jeder Schritt durch einen Befehl beschrieben werden kann. Die gesamte Befehlsfolge zur Lösung einer Aufgabe ist das Programm.

1.2. Informationsdarstellung

Rechenanlagen arbeiten aus technischen Gründen mit Elementen, die nur zwei Zustände annehmen können. Diese zwei Zustände repräsentieren die Ziffern 0 und 1. Eine Größe, die nur zwei Werte annehmen kann, heißt Binärzeichen oder Bit (binary digit). Ein Bit ist die kleinste Informationseinheit, es trägt die Information einer Ja- Nein-Entscheidung.

Jede vom Rechner zu verarbeitende Information ist in binärer Form zu verschlüsseln. Ein Rechner muß Zahlen, aber auch Buchstaben und Sonderzeichen, verarbeiten. Ist jedem Zeichen eines Zeichenvorrats in eindeutiger Weise eine Kombination von 0 und 1 zugeordnet, so liegt eine binäre Codierung des Zeichenvorrats vor. Mit zwei Binärstellen sind vier Kombinationen möglich. Allgemein gibt es bei n Stellen 2^n verschiedene Bitmuster. Der Umfang des Zeichenvorrats bestimmt also die Anzahl der zur Verschlüsselung nötigen Binärstellen.

Im Rechner ist eine feste Anzahl von Binärstellen zu einer Verarbeitungseinheit, einem Wort, zusammengefaßt. Die Stellen eines Wortes werden jeweils parallel verarbeitet.

Die Zusammenfassung von 8 Bit heißt Byte. Im TV-Computersystem ist ein Byte die Verarbeitungseinheit des Mikroprozessors. Die Binärstellen eines Bytes numerieren wir von rechts nach links von 0 bis 7 durch.

Das Bitmuster eines Wortes kann verschieden interpretiert werden. Es kann beispielsweise

- o eine Zahl (siehe Kapitel 4),
- o einen Befehl oder Teil eines Befehls,
- o ein oder mehrere Zeichen

darstellen.

Wie ein Bitmuster im Datenbereich zu interpretieren ist, legt der Programmierer fest. Die Codierung der Befehle ist durch die Rechnerstruktur bestimmt. Beispielsweise kann im TV-Computersystem das Bitmuster

1000 1011

die Zahl 139, den Befehl Addiere oder ein graphisches Element darstellen.

1.3. Funktionseinheiten einer Rechenanlage

Um eine erste Vorstellung über die Arbeitsweise eines Rechners zu bekommen, ziehen wir zum Vergleich das Arbeiten mit einem Taschenrechner heran. Es sei eine Zahlenreihe, die auf einem Blatt Papier steht, zu addieren. Man tippt die erste Zahl ein, drückt die Taste für Addition, gibt die nächste Zahl ein usw.

(es kann auch eine andere Reihenfolge der Eingabe erforderlich sein) und notiert schließlich das Ergebnis. Die Zahlen, d. h. die zu verarbeitenden Daten und anschließend das Ergebnis, sind auf dem Papier gespeichert. Für die Verarbeitung übertragen wir die Zahlen in das Rechenwerk des Taschenrechners (Eingabe) und geben dem Rechner durch Drücken der Additionstaste die Anweisung, die eingegebenen Werte zu addieren. Die Ergebnisse sehen wir im Anzeigefeld.

Für einen Rechner, in dem die beschriebenen Schritte automatisch, d. h. ohne Eingriff von außen ablaufen, erklären sich damit bereits die Funktionseinheiten Ein- Ausgabe, Speicher und Rechenwerk. Hinzu kommt das Steuerwerk, das mit dem Bediener des Taschenrechners vergleichbar ist. Es sorgt für die Ausführung der einzelnen Operationen in der vom Programm vorgeschriebenen Reihenfolge.

Ein- Ausgabe

Die Ein- Ausgabeeinheiten stellen die Verbindung des Rechners zur Umwelt her. Auf die zahlreichen Peripheriegerätetypen, wie sie bei größeren Anlagen eingesetzt werden, können wir hier nicht näher eingehen.

Speicher

Der Speicher einer Rechenanlage dient zur Aufnahme von Programmen und Daten in binärer Form. Steuerwerk und Rechenwerk haben direkten Zugriff zum Speicher. Der Speicher besteht aus einer von Null ab durchnumerierten Folge von Speicherzellen oder Worten. Ein Wort ist die kleinste adressierbare Einheit des Speichers. Im TV-Computersystem ist das 1 Byte. Die jeder Speicherzelle eindeutig zugeordnete Nummer heißt Adresse. Über die Adresse kann der Inhalt der betreffenden Zelle gelesen oder überschrieben werden.

Ob in einer Zelle ein Befehl oder ein Datum steht, ist am Inhalt der Zelle nicht ersichtlich. Dies ist Sache der Interpretation des Speicherinhalts. Die Speicherkapazität wird in der Einheit k (Kilo) angegeben, wobei hier k die Bedeutung von $2^{10} = 1024$ hat.

Rechenwerk

Das Rechenwerk führt die arithmetischen und logischen Operationen aus. Es besteht aus einem oder mehreren rechenfähigen Registern, den Akkumulatoren. Die Register nehmen die Operanden bzw. die Ergebnisse der Operation auf. Die Verarbeitungsbreite des Rechenwerks ist die Zahl der parallel verarbeiteten Bits.

Steuerwerk

Das Steuerwerk steuert die Abläufe im Rechner und das Zusammenspiel der verschiedenen Einheiten in der vom Programm vorgeschriebenen Weise, so daß ein Programm ohne Eingriff von außen ablaufen kann.

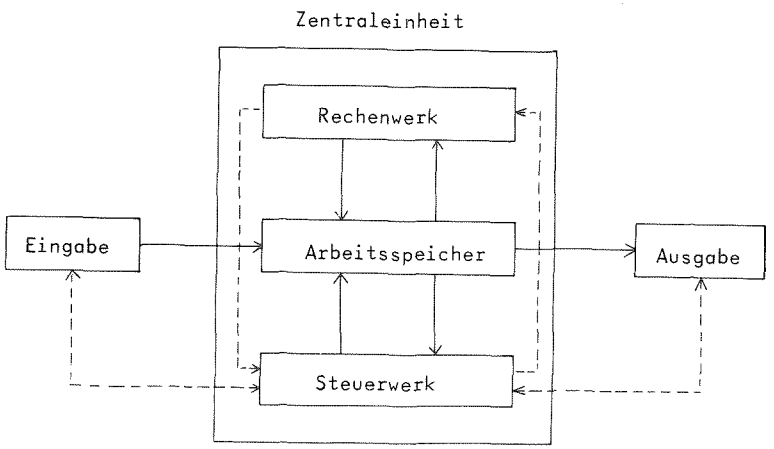


Bild 1.1 Funktionseinheiten einer Rechenanlage
⟶ Daten, ----> Steuersignale

Programmablauf:

Ein Befehl enthält mindestens zwei Angaben:

- die Art der Operation,
- die Adresse des Operanden.

Er besteht aus Operationscode und Adreßteil. Diese Befehlsform (Einadreßbefehl) liegt auch im TV-Computersystem vor. Bei der Verknüpfung von zwei Operanden, wie z. B. bei der Addition, muß der erste Operand bereits im Rechenwerk in einem Akkumulator stehen, den zweiten Operanden bezeichnet der Adreßteil. Dieser steht unter der betreffenden Adresse im Speicher. Das Ergebnis steht anschliessend im Akkumulator.

Der Grundzyklus eines Rechners ist die Verarbeitung eines Befehls.

Die Befehle eines Programms stehen im Speicher in aufeinanderfolgenden Speicherzellen. Ein Register des Steuerwerks, der Befehlszähler, enthält die Adresse des auszuführenden Befehls. Beim Programmstart ist die Adresse des ersten Befehls in den Befehlszähler zu laden. Eine Befehlsausführung läuft nun in folgenden Schritten ab:

- Auslesen des durch den Befehlszähler adressierten Befehls durch das Steuerwerk (Befehlsholphase).
- Interpretation des Operationscodes durch das Steuerwerk.
- Ausführung des Befehls durch die betreffende Funktionseinheit. Das Steuerwerk veranlaßt z. B. eine Addition durch das Rechenwerk.
- Erhöhen des Befehlszählers.

Durch das Erhöhen des Befehlszählers ist die Ausgangssituation für den nächsten Befehlszyklus hergestellt. Beim linearen Programmablauf führt der Rechner die Befehle in der Reihenfolge aus, wie sie im Speicher stehen. Sprungbefehle setzen den Befehlszähler auf eine neue Programmadresse und erlauben damit eine Änderung der Abarbeitungsreihenfolge. Bei bedingten Sprungbefehlen ist die Ausführung des Sprungs von einer Bedingung abhängig. Durch die Sprungbefehle eröffnen sich die vielfältigen Möglichkeiten in der Programmierung eines Rechners.

1.4. Programmieren

Programmieren heißt, die Lösung einer Aufgabe in einer Sprache zu formulieren, die ein Rechner verarbeiten kann. Wie bereits in 1.2 erläutert, verarbeitet ein Rechner nur Binärinformation. Eine Befehlsfolge steht im Speicher also ebenfalls in binärer Form, im sogenannten Maschinencode. Der Befehl Addition könnte beispielsweise durch die Zahl 80 verschlüsselt sein. Im Maschinencode programmierte man nur in den Anfängen der Datenverarbeitung.

Den ersten Fortschritt brachte die Einführung von Assemblersprachen. Hier erhält jeder Befehl eine Abkürzung von einigen Buchstaben, die sich aus der Bezeichnung für die betreffende Operation ableitet, beispielsweise ADD für die Addition. Speicherzellen kann man symbolische Namen zuordnen und im Programm über diese Namen ansprechen. Durch eine sachbezogene Namensgebung wird ein Assemblerprogramm bereits wesentlich besser lesbar als ein in Maschinencode geschriebenes.

Ein in Assemblersprache geschriebenes Programm kann vom Rechner nicht unmittelbar verarbeitet werden. Es muß von einem anderen Programm, dem Assembler, in den Maschinencode übersetzt werden. Der Assembler übersetzt die Befehlsbezeichnungen in den Maschinencode und ersetzt die symbolischen Adressen durch absolute Adressen. In Assemblersprache geschriebene Programme sind maschinenabhängig.

Die Entwicklung führte zu höheren problemorientierten Programmiersprachen (FORTRAN, ALGOL, PL/1, COBOL usw.), die heute bei der Programmierung größerer Anlagen hauptsächlich eingesetzt werden. Diese Sprachen erlauben eine dem Menschen und den Problemen angepaßtere Formulierung von Programmen und sind in gewissem Maße maschinenunabhängig. Bei Mikrocomputersystemen sind sie zur Zeit noch nicht verbreitet. Der Aufwand für die Übersetzerprogramme ist sehr hoch

Von der Aufgabenstellung bis zum Vorliegen der Ergebnisse sind bei der Programmentwicklung verschiedene Phasen zu durchlaufen:

- o Problemanalyse
- o Auswahl eines Lösungsverfahrens
- o Erstellung eines Programmablaufplanes
- o Programmierung
- o Testläufe
- o Produktionslauf

Die Problemanalyse muß zu einer genauen Aufgabenstellung führen. Voraussetzung für den Einsatz des Rechners ist, daß für die betreffende Aufgabe ein Algorithmus bekannt ist. Das ist ein Lösungsverfahren, das jeden Schritt eindeutig beschreibt und nach endlich vielen Schritten endet. Vor allem bei nichtmathematischen Problemen ist zuerst eine Formalisierung der Aufgabenstellung zu suchen.

Der gewählte Lösungsweg wird in Einzelschritte aufgegliedert, deren Zusammenhang am anschaulichsten in einem Programmablaufplan (6.1) darstellbar ist. Es kann dabei mehrere Detaillierungsstufen für die Aufbereitung des Programms geben. Der Programmablaufplan ist die Grundlage für die eigentliche Programmierungsarbeit.

Das Programm wird nun in der zur Verfügung stehenden oder ausgewählten Programmiersprache geschrieben, als Eingabe für ein Übersetzungsprogramm aufbereitet (außer bei Programmierung in Maschinencode) eingegeben, übersetzt und mit Testdaten gestartet.

Ein Programm wird in den seltensten Fällen auf Anhieb fehlerfrei sein. Es folgt eine Reihe von Testläufen mit Auswertung der Testergebnisse, Fehlersuche, Fehlerkorrektur und erneuter Übersetzung, bis das Programm in allen Situationen die erwarteten Ergebnisse liefert. Während der Programmentwicklung sollte immer die Dokumentation auf dem laufenden sein. Dies ist für die Fehlersuche wichtig.

1.5. Mikroprozessoren

Ein Mikroprozessor besteht aus einem oder mehreren Halbleiterbauelementen und übernimmt die Funktion des Rechen- und Steuerwerks. Beim M 6800 sind Register, arithmetische Einheit und Ablaufsteuerung auf einem Chip integriert. Er ist jedoch allein noch nicht arbeitsfähig.

Bildet ein Mikroprozessor zusammen mit einem Speicher und Ein-Ausgabemöglichkeiten ein arbeitsfähiges System, so spricht man von einem Mikrocomputer.

Die Unterschiede zu größeren Anlagen liegen hauptsächlich in der geringeren Verarbeitungsbreite und dem Fehlen höherer arithmetischer Operationen wie Multiplikation, Division oder Gleitkommaoperationen.

Die ersten Mikroprozessoren verarbeiteten 4 Bits. Inzwischen ist die Wortlänge von 8 Bits am verbreitesten, 16 Bit Prozessoren sind auch schon auf dem Markt. Eine Sonderform sind 2 oder 4 Bits verarbeitende Mikroprozessorbausteine, die sich zu größeren Wortlängen zusammenschalten lassen.

Es gibt eine Vielfalt unterschiedlicher Typen, die sich durch Wortlänge, Befehlsvorrat, Zahl der Register, Adressierungsmöglichkeiten, maximale Speicherkapazität, Unterbrechungsmöglichkeiten und Verarbeitungsgeschwindigkeit unterscheiden.

Die Preisentwicklung auf dem Mikroprozessormarkt ermöglicht zunehmend die Ablösung spezieller Schaltungen durch Mikrocomputer. Dadurch verlagert sich die Entwicklung von der Hardware in die Software. Das für den speziellen Anwendungsfall geschriebene Programm steht dann normalerweise in einem Festwertspeicher (ROM, read only memory).

2. Das TV-Computersystem 6800

2.1. Zielsetzung

Durch den technischen Fortschritt auf dem Gebiet der Mikroprozessoren wurde das "Rechenzentrum für jedermann" realisierbar. Das TV-Computersystem wurde in erster Linie als preiswertes Lehrgerät entwickelt um den Einstieg in die Logik des Programmierens mit gutem Wirkungsgrad zu ermöglichen. Wichtige Pluspunkte sind:

- o Die Programmierung auf Maschinenebene fördert die Einsicht in die Wirkungsweise eines Rechners.
- o Durch die optischen Möglichkeiten des Fernsehgeräts können die Abläufe anschaulich verfolgt werden.
- o Durch unmittelbaren Eingriff mit dem Lichtgriffel keine Umwege bei Eingabe und Bedienung.
- o Der Benutzer ist in Arbeitszeit und Lerngeschwindigkeit völlig frei.
- o Die Einzelbefehlsausführung mit Registeranzeige ist eine komfortable Testmöglichkeit.
- o Die vielfältigen Einsatzmöglichkeiten zu Spiel und Unterhaltung motivieren besonders.

2.2. Allgemeiner Aufbau

TV-Computersystem 6800

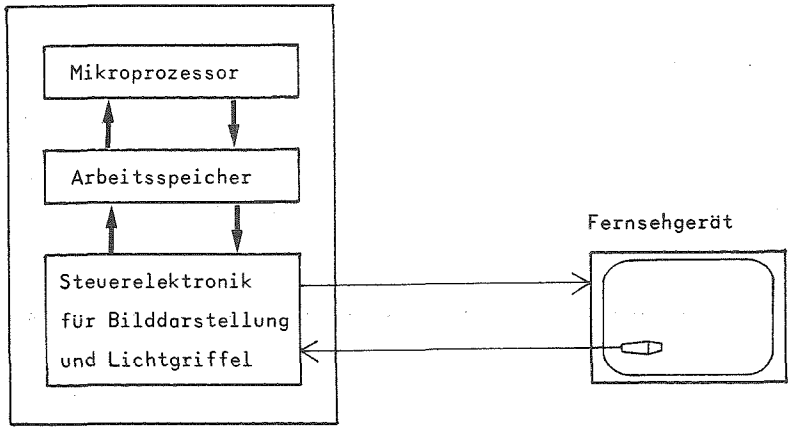


Bild 2.1 Aufbau des TV-Computersystems 6800

Das TV-Computersystem besteht aus 3 Komponenten:

- o Mikroprozessor 6800
- o Arbeitsspeicher
- o Steuerelektronik für die Ein-Ausgabe über ein Fernsehgerät.

Der Mikroprozessor vereinigt in sich die Komponenten Rechenwerk und Steuerwerk, wie wir sie im vorigen Abschnitt kennengelernt haben. Alle für die Programmierung in diesem System wesentlichen Eigenschaften wie Befehlsvorrat, Adressierungsarten und Informationsdarstellung sind durch den Mikroprozessor bestimmt.

Es ist noch bemerkenswert, daß das Gerät ohne Systemsoftware unmittelbar für den Benutzer programmierbar ist, da die Ein-Ausgabefunktionen hardwaremäßig gelöst wurden.

2.3. Mikroprozessor 6800

Der Mikroprozessor 6800 ist ein 8 Bit Prozessor. Er besitzt einen Befehlsvorrat von 72 Befehlen bzw. unter Berücksichtigung der verschiedenen möglichen Adressierungsarten 197 Befehle. Die Befehlslänge ist variabel und beträgt 1,2 oder 3 Bytes. Das erste Byte gibt den Operationscode an. Dieser spezifiziert sowohl die auszuführende Operation als auch den Adressierungsmodus. Der maximale Speicherausbau beträgt 64 k Bytes. Ferner gibt es verschiedene Interruptmöglichkeiten.

2.3.1. Register

o *Akkumulatoren A und B*

Die Akkumulatoren A und B sind die eigentlichen Rechenregister mit einer Verarbeitungsbreite von 8 Bits. Die beiden Register sind - mit geringfügigen Ausnahmen - gleichwertig, d.h. fast alle Befehle, die sich auf einen Akkumulator beziehen, gibt es in gleicher Weise für ACCA und ACCB.

Bei allen Operationen mit zwei Operanden steht der erste Operand, bzw. nach der Befehlsausführung das Ergebnis, in einem Akkumulator. Dagegen können sich Befehle mit nur einem Operanden auf einen Akkumulator oder auf eine Speicherzelle beziehen.

In den Akkumulatoren werden insbesondere die arithmetischen Operationen Addition und Subtraktion ausgeführt sowie logische Verknüpfungen. Für Multiplikation und Division gibt es keine Befehle. Diese Operationen sind durch Programme zu realisieren.

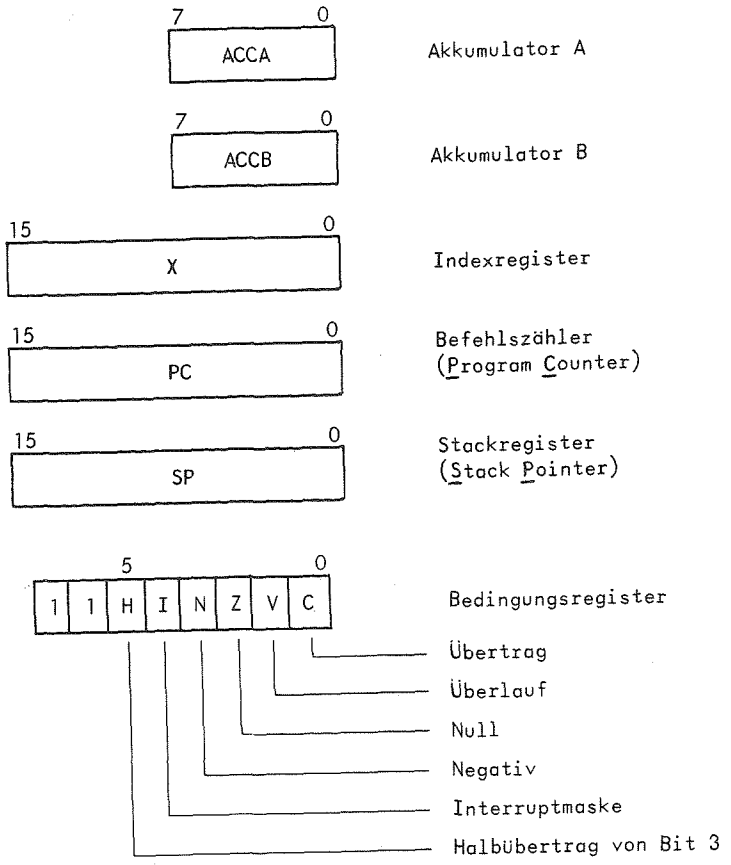


Bild 2.2 Register des 6800

o Indexregister X

Das Indexregister ist ein 16 Bit Register. Bei den 16 Bit Registern bezeichnen wir die Bitstellen 0 - 7 als niederwertiges und die Stellen 8 - 15 als höherwertiges Byte. Der Inhalt des Indexregisters wird als positive Zahl aufgefaßt. Es ist also ein Zahlenbereich von 0 bis 2^{16} - 1 darstellbar.

Das Indexregister wird in erster Linie im indizierten Adressierungsmodus eingesetzt. Es ist damit der gesamte Adressenraum von 64 k mit indizierter Adressierung zu erreichen.

Es ist ferner als Zähler in diesem Zahlenbereich geeignet, während es für andere arithmetische Operationen mit doppelter Länge (16 Bit) nicht einsetzbar ist.

o Befehlszähler PC (Program Counter)

Im Befehlszähler - ebenfalls 16 Bit umfassend - steht jeweils die Adresse des nächsten auszuführenden Befehls. Die Adresse des ersten Befehls eines Programmes wird bei Programmstart (siehe 3.3) automatisch aus den für die Startadresse fest vorgesehenen Speicherzellen 3FE, 3FF in den Befehlszähler geladen.

Nach Ausführung eines Befehls erhöht sich der Befehlszähler automatisch um 1, 2 oder 3, je nachdem ob der Befehl 1, 2 oder 3 Bytes belegt , so daß er wieder die Adresse des nächsten Befehls enthält.

Unbedingte Sprungbefehle bzw. bedingte Sprungbefehle mit erfüllter Bedingung, setzen den Befehlszähler auf die Adresse des Sprungziels.

o Stackregister SP (Stack Pointer)

Das 16 Bit Stackregister ist für die Aufnahme der aktuellen Adresse im Stack vorgesehen. Der Stack kann im Speicher an beliebiger Stelle plaziert werden. Daher ist dieses Register per Programm - vorzugsweise im ersten Befehl - zu laden. Der Stack wird für Unterprogramme, Interrupt und Einzelbefehlsausführung benötigt.

o Bedingungsregister

Von den 8 Binärstellen des Bedingungsregisters sind nur 6 belegt. Die Bits 6 und 7 sind immer auf 1 und haben keine weitere Bedeutung.

Durch die Ausführung eines Befehls können ein oder mehrere Stellen im Bedingungsregister gesetzt oder gelöscht werden. Einige Befehle verändern das Bedingungsregister nicht. Die genaue Wirkung eines Befehls auf das Bedingungsregister ist aus der letzten Spalte der Befehlsliste im Anhang B abzulesen. Das Bedingungsregister zeigt gewisse Eigenschaften des Ergebnisses einer Befehlsausführung an, die über bedingte Sprungbefehle abgefragt werden können.

Beispielsweise "Ergebnis ist negativ".

2.3.2. Stack

Der englische Begriff Stack heißt übersetzt Kellerspeicher oder Stapel.

Der Stack ist ein Speicherbereich, in dem das Abspeichern und Abholen von Daten nach dem Prinzip "last in - first out" erfolgt.

Der zuletzt gespeicherte Wert wird als erster wieder abgerufen. Es kann normalerweise immer nur auf das zuletzt abgelegte Wort zugegriffen werden.

Von der Zugriffsmöglichkeit her ist der Stack mit einem Kartenstapel vergleichbar, bei dem nur oben Karten weggenommen oder aufgelegt werden dürfen.

Der Stack dient zum Aufbewahren von:

- o Daten und Zwischenergebnissen
- o Unterprogrammrücksprungadressen
- o Registerinhalten bei Unterbrechungen.

Das Stackkonzept ist wesentlich für die Verarbeitung geschachtelter Strukturen wie z.B. geschachtelte Unterprogrammaufrufe (siehe 6.6.1).

Es gibt zwei Realisierungsmöglichkeiten. Bei manchen Rechnern ist ein gesonderter Speicher fester Länge für den Stack vorgesehen. Dessen Länge begrenzt die maximale Schachtelungstiefe.

Beim Mikroprozessor 6800 ist der Stack Teil des normalen Arbeitsspeichers. Er kann an beliebiger Stelle im Speicher liegen und beliebig lang werden.

Zur Adressierung ist ein eigenes Register, das Stackregister, vorgesehen. Es enthält immer die Adresse der ersten freien Speicherstelle im Stack. Befehle, die sich auf Daten im Stack beziehen, adressieren diese über das Stackregister. Beim Abspeichern (PUSH) erniedrigt sich das Stackregister automatisch um eins, beim Abholen (PULL) erhöht es sich um eins.

Vor dem ersten Zugriff auf den Stack ist das Stackregister mit der Anfangsadresse zu laden. Es ist die maximale Belegung abzuschätzen und ausreichend Platz freizuhalten, sonst werden möglicherweise Befehle oder Daten überschrieben.

Man beachte, daß das Abspeichern von höheren zu niedrigeren Adressen fortschreitet. Deshalb ist es günstig, den Stack an das Ende des freien Speichers zu legen (z.B. Adresse 3F5).

Im Vorgriff auf Kapitel 5 seien noch die Befehle genannt, die das Stackregister und/oder den Stack verändern:

LDS, STS, DES, INS, TXS, TSX, PSH, PUL,
JSR, BSR, RTS, SWI, WAI, RTI.

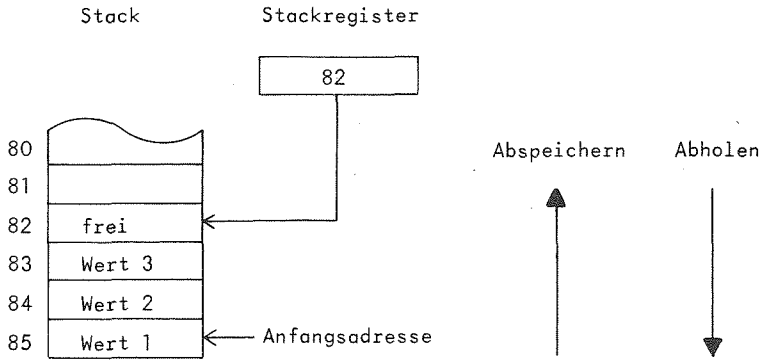


Bild 2.3 Zugriff auf Daten im Stack

2.3.3. Interrupt

Ein Interrupt ist eine Unterbrechung des laufenden Programms zu einem unvorhergesehenen Zeitpunkt. Ein Interrupt signalisiert ein Ereignis, das vorrangig abgehandelt werden soll. Bei der Abwicklung des Ein- Ausgabeverkehrs und besonders in der Prozeßdatenverarbeitung spielt die Interruptverarbeitung eine große Rolle.

Beim Eintreffen eines Interruptsignals wird das laufende Programm vorübergehend unterbrochen und eine Interrupt-Routine gestartet. Nach Abschluß dieses Programms geht die Regie an das unterbrochene Programm zurück.

Dies verlangt, daß sämtliche Registerinhalte vor dem Start der Interrupt-Routine gerettet und am Ende wieder geladen werden.

Der Mikroprozessor 6800 speichert bei einem Interrupt alle Register mit Ausnahme des Stackregisters automatisch auf dem Stack ab. Ein Interruptprogramm muß mit dem Befehl RTI (Rückkehr von Interrupt) enden, der die Register wieder zurück lädt. Die Reihenfolge der Register im Stack ist in 3.5 beschrieben.

Es ist zu klären, was passiert, wenn während des Ablaufs eines Interruptprogramms ein weiteres Interruptsignal eintrifft. Dies ist grundsätzlich nicht verboten. Der Programmierer kann vom Programm aus jedoch die Annahme eines Interrupts zeitweise oder auch generell unterbinden. Hierzu dient ein setz- und löschbares Maskenbit im Bedingungsregister: die Interruptmaske I. Bei gesetztem Maskenbit wird ein anstehender Interrupt nicht erkannt. Das Maskenbit wird bei Eintritt in die Interrupt-Routine automatisch gesetzt. Weiteres hierzu siehe Kapitel 6.7.

Im TV-Computersystem 6800 läßt sich der Interrupt über einen virtuellen Schalter mit dem Lichtgriffel auslösen. In Spielprogrammen ermöglicht der Interrupt beispielsweise eine Art Dialog zwischen Spieler und Rechner. Nach Eingabe seines Zuges signalisiert der Spieler dem Rechner über einen Interrupt, daß er den nächsten Zug berechnen soll. Das Interruptprogramm erarbeitet den Zug des Rechners und kehrt anschließend in eine Warteposition im Hauptprogramm zurück, bis der nächste Zug berechnet werden soll (siehe 7.6.3.).

Der M 6800 hat zwei Interruptebenen: den beschriebenen maskierbaren Interrupt und einen nichtmaskierbaren Interrupt. Mit letzterem wurde die Einzelbefehlsausführung realisiert (siehe 3.5). Er ist im TV-Computersystem für andere Zwecke nicht verwendbar.

Der Software-Interrupt löst einen Interrupt per Befehl (SWI) durch das eigene Programm aus. Er kann für Testzwecke verwendet werden, da er ebenfalls die Registerinhalte im Stack ablegt. Durch die Einzelbefehlseinrichtung kommt dem SWI keine größere Bedeutung zu.

In diesem Zusammenhang sei noch der Befehl "Warten auf Interrupt" (WAI) erwähnt. Er lädt die Registerinhalte in den Stack und versetzt das System in den Wartezustand, bis einer der beiden Hardwareinterrupts ansteht.

Für die Startadressen der Interruptprogramme und des Hauptprogramms sind am Ende des Speichers feste Zellen reserviert. Diese müssen die Absolutadressen (16 Bit) des jeweiligen Programms enthalten.

Man nennt diesen Speicherbereich auch Interruptvektor. Es sind dies die letzten 8 Bytes des Speichers, d.h. auf dem Bildschirm die letzten 4 Zeilen auf Seite 1F. Das linke Byte nimmt jeweils den höherwertigen, das rechte Byte den niederwertigen Anteil der

Adresse auf. Tabelle 2.4 gibt eine Übersicht über die Startadressen, Bild 2.5 zeigt den internen Ablauf bei der Interruptverarbeitung.

Adresse	reserviert für	Zeile
3F8, 3F9	Startadr. Interruptprogramm	C
3FA, 3FB	" Softwareinterruptprogramm	D
3FC, 3FD	" Einzelbefehlsprogramm	E
3FE, 3FF	" Hauptprogramm	F

Tabelle 2.4 Speicherbelegung für Startadressen

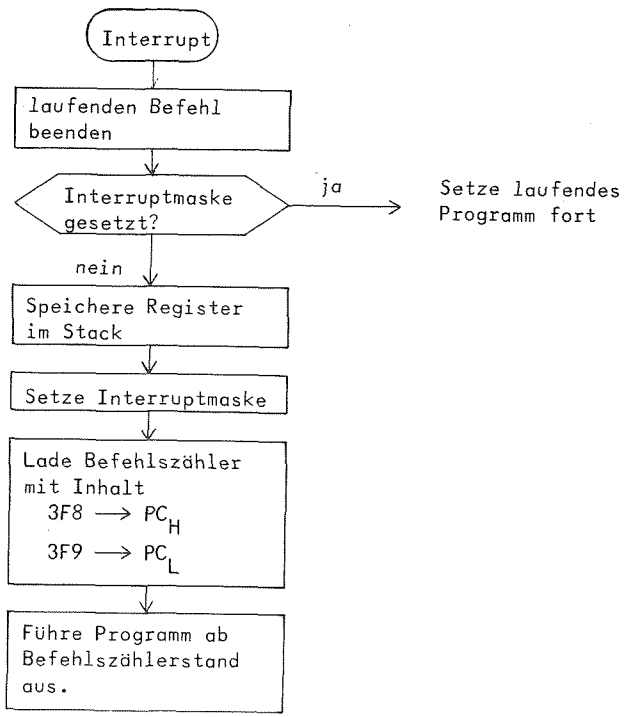


Bild 2.5 Ablauf Interruptsequenz

2.4. Speicher

Das Gerät ist mit 1 k Byte Speicher ausgestattet. Damit geht der gültige Adressenbereich von 0 bis 3FF. Mit Ausnahme der letzten 8 Bytes - das sind die Adressen 3F8 bis 3FF - die für die Aufnahme von Start- und Interruptadressen fest belegt sind, ist der Speicher für den Benutzer frei verfügbar. Er kann Programme, Daten und Stack in beliebige Bereiche legen.

Auf dem Fernsehbild werden jeweils 32 Bytes bit- oder tetradenweise dargestellt. Einen solchen Speicherausschnitt bezeichnen wir als Seite.

Es gibt insgesamt also 32 Speicherseiten. Die Nummer der gerade dargestellten Seite wird am rechten Bildrand hexadekadisch angezeigt.

Es handelt sich um einen Halbleiterspeicher, der seine Information beim Abschalten der Stromversorgung verliert. RAM ist die Abkürzung von Random-Access-Memory und bedeutet Schreib-Lesespeicher mit wahlfreiem Zugriff.

. Eingabe und Ausgabe

ı handelsübliches Schwarz-Weiß-Fernsehgerät dient in Verbindung
t einem Lichtgriffel als Ein- und Ausgabegerät.

ıe Eingabe erfolgt durch optischen Kontakt des Lichtgriffels mit
en auf dem Bildschirm dargestellten Steuerpunkten. Damit kann
.nformation bit- oder tetradenweise unmittelbar in den Speicher
eingeschrieben werden. Da der Fernseher gleichzeitig Ein- und Ausgabegerät ist, ist die Wirkung einer Eingabe sofort zu sehen und zu überprüfen. Die Eingabe eines Programms erfolgt tetradenweise in Maschinencode.

Neben der Eingabe in den Speicher gibt es am rechten Bildrand "Funktionstasten", die ebenfalls über den Lichtgriffel betätigt werden.

Die Ergebnisse eines Programms sollten möglichst auf einer Seite liegen, damit sie ohne dauerndes Umschalten beobachtbar sind. Diese "Anzeigeseite" muß der Benutzer selbst einstellen. Vom Programm aus ist die Anzeige einer bestimmten Seite nicht steuerbar.

3. Anleitung zum Arbeiten mit dem TV-Computersystem 6800

3.1. Bedienungselemente

Auf dem Bildschirm des Fernsehgeräts befinden sich Steuer- und Anzeigeelemente. Die Steuerelemente sind kleine quadratische Felder, die durch den optischen Kontakt mit dem Lichtgriffel einen Schaltvorgang bewirken. Rechts des Steuerfeldes befindet sich das Anzeigefeld. Dieses stellt den jeweiligen Zustand des Schalters dar. Anhand von Bild 3.1 werden die Bedienungsmöglichkeiten kurz erläutert.

Auf der linken Bildschirmseite wird ein 32-Byte-Ausschnitt aus dem Speicher abgebildet. Bei Tetradendarstellung kann über die Steuerfelder jeweils eine beliebige Tetrade in den Speicher geschrieben werden. Bei Bitdarstellung kann mit dem Lichtgriffel jedes der 256 angezeigten Bits invertiert werden.

Auf der rechten Bildschirmseite befinden sich die Bedienungsfelder in zwei Spalten angeordnet. Die linke Spalte dient der Tetradenauswahl. Neben den Steuerfeldern stehen von oben nach unten die Tetraden 0 bis F. Über die Steuerfelder kann die entsprechende Tetrade angewählt werden. Die ausgewählte Tetrade erscheint in der rechten Spalte in Höhe der Tetrade 4.

Die Steuer- und Anzeigefelder der rechten Spalte haben der Reihe nach folgende Funktionen:

- Seite mit angewählter Tetrade setzen. Dies ist nur bei Tetradenanzeige möglich. Ist die angewählte Tetrade 0, so läßt sich die dargestellte Speicherseite löschen. Läßt man im eingeschalteten Zustand dieses Schalters den Seitenzähler einmal ganz durchlaufen, so ist der gesamte Speicher gelöscht bzw. vorbesetzt.

- Seitendarstellung Bit/Tetrade
  Hiermit läßt sich die Darstellung der Speicherseite zwischen Bit und Tetraden umschalten.

- Mit den folgenden zwei Steuerfeldern läßt sich der Seitenzähler vorwärts und rückwärts zählen. Auf diese Weise können die 32 Seiten angewählt werden.

- Die folgenden beiden Steuerfelder dienen dem unmittelbaren Setzen der beiden Zählerstellen. Neben den beiden Steuerfeldern wird der Zählerstand angezeigt.

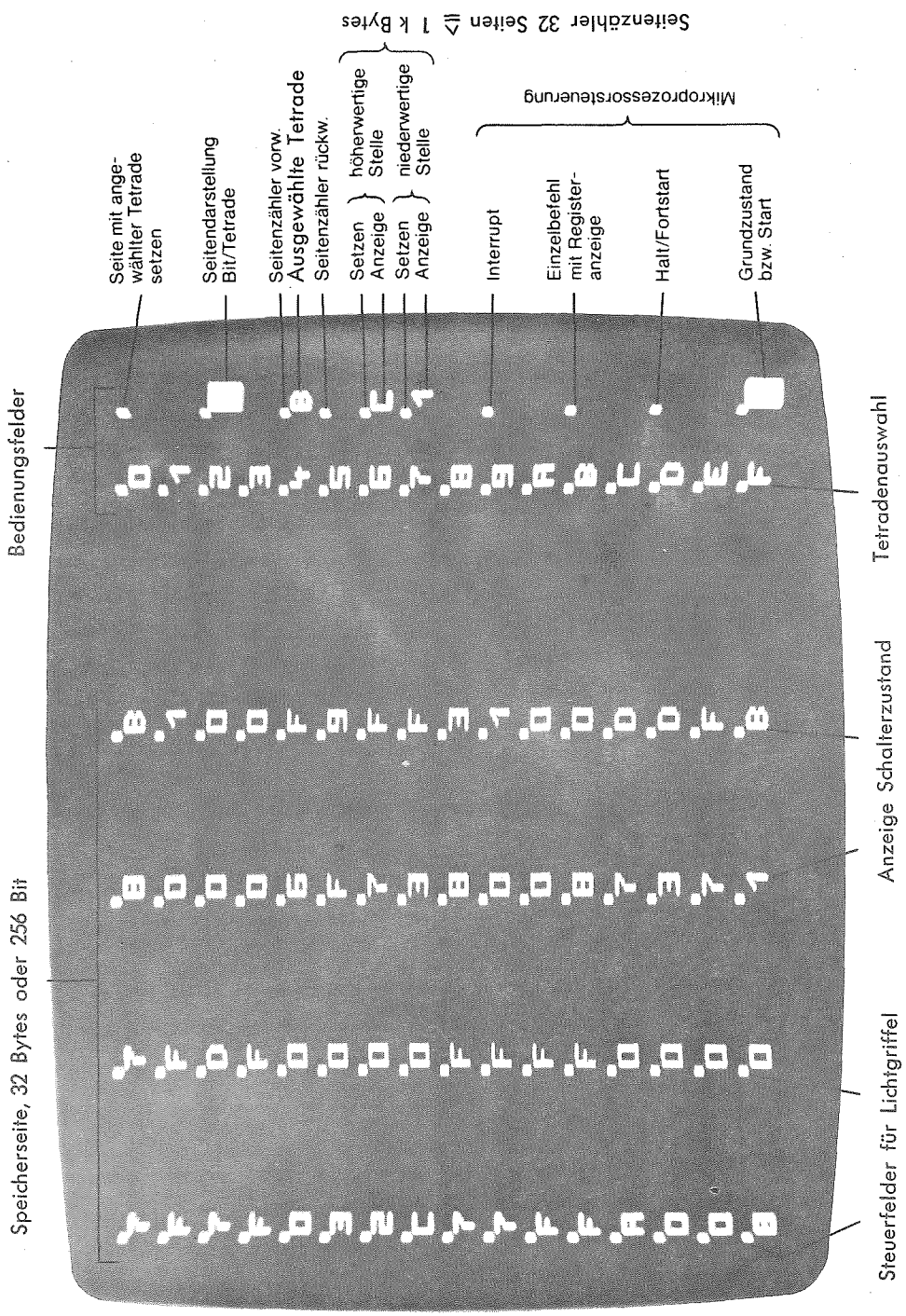
Die letzten vier Felder dienen der Mikroprozessorsteuerung.

- Interrupt zur Unterbrechung des gerade laufenden Programms und Start eines Interruptprogramms.

- Einzelbefehlsausführung mit Registeranzeige.

- Halt - Fortstart

- Grundzustand - Start



3.2. Betriebszustände

Zur Steuerung des Mikroprozessors gibt es vier Funktionstasten mit folgender Bedeutung:

- o Grundzustand - Start
- o Halt - Fortstart
- o Einzelbefehl
- o Interrupt

Nach dem Einschalten befindet sich das Gerät im Grundzustand, d.h. alle vier Tasten sind ausgeschaltet.

Die Start- und die Halttaste zeigen einen Zustand des Mikroprozessors an und bewirken beim Ein- und Ausschalten einen Zustandsübergang. Für den Programmstart betätigt man die Starttaste. Der Prozessor geht in den Zustand "Rechnen" über und bleibt in diesem Zustand, bis er

a) durch Ausschalten der Starttaste wieder in den Grundzustand versetzt wird, oder

b) durch Einschalten des Halts in den Haltezustand übergeht.

Durch den Halt wird das gerade laufende Programm an einer beliebigen Stelle unterbrochen, wobei der aktuelle Prozessorstatus erhalten bleibt. Durch Aufheben des Halts wird das Programm an der Unterbrechungsstelle fortgesetzt.

Die Einzelbefehlsausführung setzt voraus, daß der Prozessor gestartet wurde und sich im Haltezustand befindet. Das Einschalten der Einzelbefehlstaste hebt intern für die Dauer einer Befehlsausführung den Halt auf. Der nächste Befehl wird ausgeführt und

die Registerinhalte werden anschließend auf dem Stack abgespeichert. Beim Ausschalten der Einzelbefehlstaste werden die Register vom Stack wieder übernommen. Damit ist die Möglichkeit von Registervorbesetzungen gegeben. Weiteres hierzu siehe 3.5.

Bei Betätigen der Interrupttaste wird im Einschaltmoment ein Interruptsignal erzeugt, während das Ausschalten ohne Wirkung ist. Die Verarbeitung eines Interrupts setzt voraus, daß der Prozessor gestartet und die Startadresse eines Interruptprogramms in den Zellen 3F8, 3F9 gespeichert wurde. Befindet sich der Prozessor im Zustand "Rechnen", so wird die Interruptverarbeitung nach Beendigung des gerade laufenden Befehls angestoßen, sofern nicht die Interruptmaske gesetzt ist. Befindet sich der Prozessor im Zustand "Halt", so speichert er ein eintreffendes Interruptsignal intern. Nach Aufheben des Halts wird dann unmittelbar das Interruptprogramm gestartet.

Zusammenfassung:

Die vier Schalter zur Mikroprozessorsteuerung lassen 16 Kombinationen der Schalterstellungen zu. Von diesen sind jedoch nur einige sinn- bzw. wirkungsvoll. Tabelle 3.2 zeigt nochmals eine Übersicht.

Nr.	Inter-rupt	Einzel-befehl	Halt	Grundzu-stand	Bemerkung
1	0	0	0	0	Grundzustand, Prozessor nicht aktiv
2	0	0	0	1	Prozessor rechnet
3	0	0	1	0	Halt, nur sinnvoll im Übergang zur Stellung 4 für Einzelbefehl ab Programmanfang
4	0	0	1	1	Halt, auch Voraussetzung für Einzelbefehl
5	0	1	1	1	Einzelbefehl
6	1	0	0	1	Interrupt, nur nach Start sinnvoll
7	1	0	1	1	Interrupt bei gleichzeitigem Halt wird erst nach Lösen des Halts wirksam.

Tabelle 3.2. Betriebszustände des Mikroprozessors

3.3. Programmstart

Vor dem Programmstart ist die Startadresse - das ist die Adresse des ersten Befehls - in den letzten 2 Bytes des Speichers einzutragen. Bei unserem Speicherausbau von 1 k Byte sind das die Adressen 3FE und 3FF, die letzte Zeile von Seite 1F. Der Inhalt der Speicherzellen 3FE und 3FF wird beim Start in den Befehlszähler geladen. Anschließend beginnt der normale Befehlsablauf.

Die Registerinhalte sind bei Programmstart undefiniert mit Ausnahme der Interruptmaske im Bedingungsregister. Diese ist immer gesetzt, da der Programmstart intern als spezieller Interrupt aufgefaßt wird.

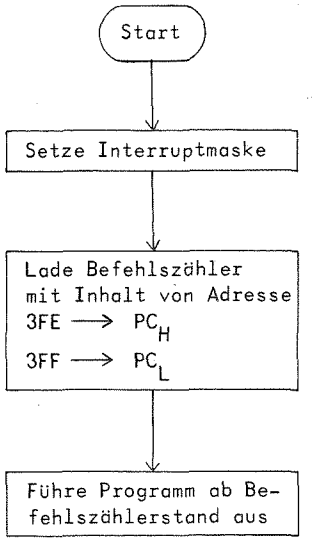


Bild 3.3 Startsequenz

3.4. Programmstop

Der Mikroprozessor besitzt keinen Stop-Befehl, das heißt, er kann nicht vom Programm aus in den Halt- bzw. Grundzustand versetzt werden.

Für zyklische Programme, wie z.B. die Programme in 7.1 zur Erzielung optischer Effekte auf dem Bildschirm, ist das ohne Bedeutung. Diese Programme sollen beliebig lange laufen bis der Benutzer von sich aus den Prozessor stoppt. Anders verhält es sich bei linearen Programmen. Als letzter Befehl eines solchen Programms muß mit Hilfe eines unbedingten Sprungbefehls durch einen sogenannten "Sprung auf sich selbst" ein dynamischer Stop programmiert werden.

Die einfachste Möglichkeit bietet Befehl BRA -2, in Maschinencode 20FE. Zur Erläuterung siehe Abschnitt 5.3.7 über relative Adressierung.

3.5. Einzelbefehlsausführung

Der Einzelbefehl mit Anzeige der Registerinhalte ist ein wichtiges Hilfsmittel zum Austesten der Programme. In TV-Computersystem wird der nichtmaskierte Interrupt hierzu ausgenützt, da beim Interrupt automatisch alle Register im Stack abgelegt werden.

Zur richtigen Durchführung des Einzelbefehls ist in den Speicherzellen 3FC, 3FD die Startadresse des Interruptprogramms einzutragen. Dieses Einzelbefehls-Interruptprogramm muß aus genau 2 Befehlen bestehen, wobei der zweite Befehl ein RTI (Rücksprung aus Interrupt) sein muß. Der erste Befehl wird normalerweise ein Leerbefehl NOP sein. Er kann aber beispielsweise auch durch einen STS (Speichere Stackregister) ersetzt werden. Dadurch wird der Inhalt des Stackregisters ebenfalls bei jedem Einzelbefehl abgelegt.

Wir empfehlen, das kurze Einzelbefehlsprogramm mit NOP und RTI standardmäßig auf die Adressen 3F6, 3F7 zu legen. Als Startadresse ist dann in den Adressen 3FC, 3FD (vorletzte Zeile auf Seite 1F) die Adresse 3F6 einzutragen. In den Beispielprogrammen in Kapitel 7 berücksichtigen wir dies und legen den Stack meist unmittelbar davor auf Adresse 3F5.

Adresse		
3F6, 3F7	0 2 3 B	(NOP, RTI)
3F8, 3F9	.....	Interrupt
3FA, 3FB	.....	Softwareinterrupt
3FC, 3FD	0 3 F 6	Einzelbefehl
3FE, 3FF	.....	Startadresse

Bild 3.4. Vorschlag für Einzelbefehlsprogramm auf letzter Seite

Die Reihenfolge der Register im Stack zeigt Bild 3.5. Es empfiehlt sich, das Stackregister *mit einer ungeraden* Adresse zu laden, damit Befehlszähler und Indexregister auf dem Bildschirm in einer Zeile stehen.

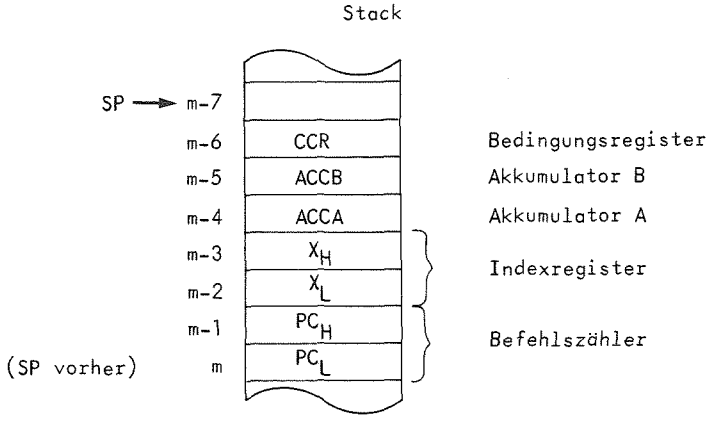


Bild 3.5 Reihenfolge der Register im Stack

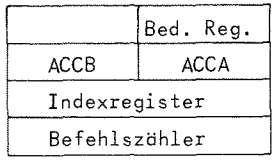


Bild 3.6 Register auf dem Bildschirm bei ungerader Stackadresse

Ist ein Programm von Beginn an per Einzelbefehl zu durchlaufen, so muß im ersten Befehl unbedingt das Stackregister geladen werden (LDS). Es ist dann zuerst der Halt und anschließend die Starttaste einzulegen. Danach ist mit dem Einzelbefehl fortzufahren.

Soll erst von einer bestimmten Stelle an im Programm mit dem Einzelbefehl fortgefahren werden, ist an diese Stelle ein WAI (Warten auf Interrupt) zu setzen. Nach dem Start läuft das Programm bis zum Befehl WAI und geht in einen Wartezustand über. Es ist dann die Halttaste einzuschalten und mit Einzelbefehl fortzufahren.

Ein zusätzlicher Testkomfort bietet die Möglichkeit, die bei der Einzelbefehlsausführung angezeigten Registerinhalte mittels Lichtgriffel zu verändern. Beim Ausschalten der Einzelbefehlstaste lädt das System die Register mit den im Stack stehenden Daten. Es ist so leicht möglich, die Register vorzubesetzen.

3.6. Befehlsformular

Die Eingabe der Befehle erfolgt im TV-Computersystem im Maschinencode direkt in den Speicher. Die Programmierung im Maschinencode ist jedoch beschwerlich, da man sich die Bedeutung vieler Ziffernkombinationen merken müßte.

Aus diesem Grund programmieren wir zunächst in einer Art Assemblersprache. Alle Befehle erhalten eine Abkürzung von 3 oder 4 Buchstaben, die sich aus der englischen Bezeichnung der entsprechenden Operation ableitet. Eine Abänderung der im Original englischen Befehlsabkürzungen erschien uns nicht sinnvoll. Im Anhang A findet sich eine alphabetische Zusammenstellung der Befehle.

Im Unterschied zur Programmierung größerer Anlagen müssen wir in einem zweiten Schritt das symbolische Programm von Hand in den Maschinencode übersetzen. Dazu ermittelt man in der Befehlsliste (Anhang B) für jeden Befehl den Operationscode und bestimmt aus der Speicherbelegung die den symbolischen Namen zugeordneten Adressen. Hierbei ist die Tabelle in Anhang D, die die Zuordnung von absoluten Adressen zu Speicherseiten wiedergibt, behilflich.

Das Befehlsformular soll einerseits die Programmierung, andererseits die anschließende tetradenweise Eingabe in den Rechner unterstützen. Da die Übersetzung nicht maschinell erfolgt, gibt es keine strengen Formvorschriften für das Ausfüllen des Formulars.

Auf einem Blatt hat genau eine Speicherseite Platz. In der Spalte "Nr" steht die dezimale in der Spalte "Byte" die hexadezimale Durchnumerierung der 32 Bytes einer Seite. Letztere ist besonders für die Ermittlung der absoluten Adressen von Sprungzielen bzw. Daten hilfreich.

In der Spalte "Marke" trägt man im Programm die Namen von Sprungzielen, im Datenbereich die Namen der Variablen ein. Dadurch werden diese Bezeichnungen definiert und einer festen absoluten Adresse zugeordnet. Es versteht sich, daß eine Bezeichnung in einem Programm nur einmal definiert werden kann.

Die Spalte "OPEX" nimmt die externe Bezeichnung des Operationscodes auf. In Spalte "A" trägt man den Adressierungsmodus ein, sofern es für einen Befehl mehrere Adressierungsarten gibt. Die Adressierungsart ist für die Umsetzung des externen Operationscodes in die interne Darstellung eine wesentliche Angabe. In Spalte "Symbolische Adresse" steht der Adreßteil eines Befehls. Es können hier symbolische Adressen, die an anderer Stelle in der Spalte "Marke" definiert sind referiert werden, man kann aber auch Zahlenwerte

eintragen.

Die Spalte "Code" nimmt die Interndarstellung der Befehle auf. Sie enthält die Angaben, die im Speicher einzutragen sind.

Die Kommentarspalte dient Dokumentationszwecken. Eine ausführliche Kommentierung erhöht für den Programmierer selbst den Überblick und das Verständnis für sein Programm und erleichtert einem fremden Leser die Einarbeitung .

Beim Eintragen des Programms in symbolischer Form ist auf die Befehlslänge zu achten, die 1, 2 oder 3 Bytes betragen kann. Sobald man mit den Befehlen und Adressierungsarten vertraut ist, ist dies nicht mehr schwierig.

3.7. Erstes Programm

Noch vor der ausführlichen Behandlung der einzelnen Befehle sollten Sie ein erstes kleines Programm ausprobieren und mit dem Gerät vertraut werden. Das Programm soll zwei zweistellige Dezimalzahlen addieren. Das Programm legen wir auf Seite 0 mit Startadresse 0, die Daten und den Stack auf Seite 1. Die beiden Summanden und das Ergebnis sollen untereinander in den rechten Bytes der ersten drei Zeilen von Seite 1 stehen. Der Stack liegt am Ende von Seite 1.

Seite: 0

Addition von zwei zweistelligen Dezimalzahlen

Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	Start	LDS	I	3F	8E	Stack liegt am Ende von
1					00	Seite 1; ist für Einzel-
2					3F	befehl notwendig
3		LDAA	D	Summand 1	96	1. Summand → ACCA
4					21	
5		ADDA	D	Summand 2	9B	ACCA + 2. Summand →
6					23	ACCA
7		DAA			19	Dezimalangleichung
8		STAA	D	Summe	97	ACCA → Summe
9					25	
A	Stop	BRA	R	Stop	20	dynamischer Stop
B					FE	

Seite: 1

Addition, Datenbereich

Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0					00	
1	Summand1				34	hier verschiedene Werte
2					00	
3	Summand2				15	eintragen
4					00	Programm errechnet Summe
5	Summe				00	und speichert sie hier ab.

Bild 3.7 Additionsprogramm 1. Version

Gehen Sie am besten in folgenden Schritten vor:

- Mindestens die Seiten 0, 1 und 1F löschen, am besten aber den gesamten Speicher löschen. (Seitenzähler mit angewählter Tetrade 0 und eingeschalteter Taste "Tetrade setzen" einmal ganz durchlaufen lassen.)

- Eingabe des Programms auf Seite 0.

- Eingabe des Einzelbefehlprogramms auf Seite 1F gemäß Bild 3.4. Die Startadresse 0 ist durch das Löschen bereits richtig geladen.

- Auf Seite 1 in der ersten und zweiten Zeile rechtes Byte zwei Zahlen eintragen.

- Programm starten.

- Programm mit anderen Werten erneut starten und einen Durchlauf mit Einzelbefehlsausführung verfolgen. Registerdarstellung am Ende von Seite 1 wie in Bild 3.6 beschrieben.

Wenn Sie Zahlen addieren, deren Summe dreistellig ist, so liefert das Programm ein falsches Ergebnis. Diese erste Fassung berücksichtigt keinen Übertrag. Wir geben eine verbesserte Lösung an.

Seite: 0

Additionsprogramm mit Berücksichtigung eines Übertrags

Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	Start	LDS	I	3F	8E	Stack liegt am Ende von
1					00	Seite 1
2					3F	
3		CLR	E	Summe -1	7F	Lösche Byte, das Über-
4					00	trag aufnimmt
5					24	
6		LDAA	D	Summand1	96	1. Summand → ACCA
7					21	
8		ADDA	D	Summand2	9B	ACCA + 2. Summand
9					23	→ ACCA
A		DAA			19	Dezimalangleichung
B		STAA	D	Summe	97	ACCA → Summe
C					25	
D		BCC	R	Stop	24	Übertrag ?
E					03	nein → Stop
F		INC	E	Summe-1	7C	ja : Erhöhe Übertrags-
10					00	byte
11					24	
12	Stop	BRA	R	Stop	20	dynamischer Stop
13					FE	

4. Zahlensysteme

4.1. Allgemeines

Jeder ist mit dem dezimalen Zahlensystem vertraut. Es gibt die zehn Ziffern 0, 1, 2 ..., 9. Eine Zahl größer 9 wird als Kombination mehrerer Ziffer dargestellt. Nehmen wir beispielsweise die Zahl 3509. Sie bedeutet ausführlich geschrieben:

$$3 \cdot 1000 + 5 \cdot 100 + 0 \cdot 10 + 9$$

oder $3 \cdot 10^3 + 5 \cdot 10^2 + 0 \cdot 10^1 + 9 \cdot 10^0$

Der Wert einer Ziffer hängt von der Stelle ab, an der sie steht. Man spricht von Einer-, Zehner-, Hunderterstelle usw. Das Dezimalsystem ist ein sogenanntes Stellenwertsystem. Von rechts nach links nimmt der Wert von Stelle zu Stelle um den Faktor 10 zu. Die Zahl 10 ist die Basis des Dezimalsystems. Ein Zusammenhang besteht noch zwischen der Basis und der Anzahl der Ziffern: es gibt 10 Ziffern (0-9).

Nun ist das Dezimalsystem nicht die einzige Möglichkeit der Zahlendarstellung. Jede ganze Zahl größer gleich 2 könnte als Basis B eines Zahlensystems genommen werden. Das Prinzip, wie man größere Zahlen bildet, bleibt das gleiche wie beim Dezimalsystem: jeder Ziffernstelle ordnet man eine Potenz der Basis zu. Ferner gilt allgemein, daß es in einem Zahlensystem zur Basis B die Ziffern 0 bis (B-1) gibt. Bei Systemen mit $B > 10$ müssen für die Ziffern 10, 11, 12 ... neue einstellige Symbole eingeführt werden. Man nimmt hierzu die Buchstaben A, B, C usw.

Mit n Stellen können im Dezimalsystem 10^n Zahlen, beispielsweise für n = 3 die Zahlen von 0 bis 999, dargestellt werden. Allgemein sind in einem Zahlensystem zur Basis B mit n Stellen B^n Zahlen

darstellbar.

Wir behandeln im folgenden nur ganze Zahlen (Festpunktzahlen). Auf Gleitkommazahlen gehen wir nicht ein, da der Mikroprozessor 6800 keine Gleitkommaarithmetik besitzt.

4.2. Dualsystem

4.2.1 Umwandlung dual ⟶ dezimal

Rechenanlagen arbeiten aus technischen Gründen im Zahlensystem zur Basis 2, dem Dualsystem. Nach obigem kann es im Dualsystem nur zwei Ziffern geben: 0 und 1. Bei einer mehrstelligen Dualzahl sind den einzelnen Positionen Zweierpotenzen zugeordnet. Damit können wir bereits Dualzahlen in Dezimalzahlen umrechnen.

Beispiel: $110101 = 1\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$

$= 32 + 16 + 0 + 4 + 0 + 1$

$= 53$

Wenn gleichzeitig mit verschiedenen Zahlensystemen gearbeitet wird, schreibt man, um Mehrdeutigkeiten zu vermeiden, die Basis als tiefgestellte Zahl hinzu.

Beispielsweise $110101_2 = 53_{10}$.

Damit wird z.B. die Zahl 11 erklärt als 11_{10} oder $11_2 = 3_{10}$.

4.2.2. Umwandlung dezimal ⟶ dual

Zu einer gegebenen Dezimalzahl ermittelt man die entsprechende Dualzahl durch das sogenannte Divisionsverfahren. Man dividiert die Dezimalzahl bzw. die entstehenden Quotienten fortlaufend durch 2 (Basis des Zielsystems), bis der Quotient Null wird. Die Reste der Division, 0 oder 1, geben von unten nach oben gelesen die gesuchte Dualzahl.

Beispiel: 29_{10}

29	:	2	=	14	Rest	1
14	:	2	=	7	"	0
7	:	2	=	3	"	1
3	:	2	=	1	"	1
1	:	2	=	0	"	1

↑ 11101

$29_{10} = 11101_2$

Machen wir übungshalber die Probe, indem wir von der Dualzahl ausgehen und die Dezimalzahl bestimmen.

$11101_2 = 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$

$= 16 + 8 + 4 + 0 + 1$

$= 29_{10}$

4.2.3. Rechnen im Dualsystem

Das Rechnen im Dualsystem verläuft ganz analog zum Rechnen im Dezimalsystem. Man muß nur beachten, daß 0 und 1 die einzigen Ziffern des Dualsystems sind.

Addition

Ein Übertrag zur nächsthöheren Stelle tritt bei der Addition im Dezimalsystem auf, wenn die größte Ziffer 9 überschritten wird. Im Dualsystem gilt entsprechend, daß bei Überschreiten der höchsten Ziffer 1 ein Übertrag auftritt. Wir geben eine Tabelle an, in der neben den beiden Summandenziffern S1 und S2 noch ein Übertrag Ü von der vorherigen Stelle berücksichtigt ist. Das Ergebnis besteht aus Summe sowie Übertrag für die nächste Stelle.

S1	S2	Ü	Summe	Übertrag
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabelle 4.1 Duale Addition

Beispiel:			
	1. Summand	0 0 1 1 1	7
	2. Summand	+ 0 1 1 0 1	+ 13
	Übertrag	1 1 1 1	1
	Summe	1 0 1 0 0	20

Subtraktion

Der Übertrag bedeutet hier, ähnlich wie im Dezimalsystem, "eins geliehen". M ist die Abkürzung für Minuend, S für Subtrahend.

M	S	Ü	Differenz	Übertrag
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabelle 4.2 Duale Subtraktion

Beispiel:			
	Minuend	1 0 1 0	10
	Subtrahend	- 0 1 0 1	- 5
	Übertrag	1 0 1	1
	Differenz	0 1 0 1	5

In Rechenanlagen wird die Subtraktion auf die Addition der Zweierkomplements zurückgeführt, siehe 4.2.4.

Multiplikation

Formal wird die Multiplikation wie im Dezimalsystem ausgeführt. Die Multiplikationsregeln für einstellige Dualzahlen sind:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

Beispiel: $7 \cdot 13 = 91$

```
0 1 1 1  ·  1 1 0 1
           0 1 1 1
         0 0 0 0
       0 1 1 1
     0 1 1 1
-----------------
   1 0 1 1 0 1 1
```

Die Multiplikation im Dualsystem kann also auf die Operationen Addition und Stellenverschiebung zurückgeführt werden. Die maximale Stellenzahl des Ergebnisses ist gleich der Summe der Stellenzahlen von Multiplikand und Multiplikator.

Division

Beispiel: $90 : 5 = 18$

```
1 0 1 1 0 1 0  :  1 0 1  =  1 0 0 1 0
- 1 0 1
-------
  0 0 0 1
      1 0
      1 0 1
    - 1 0 1
    -------
      0 0 0 0
```

4.2.4. Zweierkomplement

Das Zweierkomplement ist für die Darstellung negativer Dualzahlen im Rechner von Bedeutung. Wie schon erwähnt, sind im Dualsystem bei n Stellen 2^n Zahlen darstellbar. Sollen positive und negative Zahlen dargestellt werden, so ist der Zahlenbereich aufzuspalten. Es finden verschiedene Darstellungen negativer Zahlen Anwendung. Wir beschränken uns auf die Diskussion des Zweierkomplements, da dies im Mikroprozessor 6800 verwendet wird.

Das Zweierkomplement $\bar{z}$ einer n-stelligen (positiven oder negativen) Dualzahl z ist definiert als die Ergänzung von z zur Zweierpotenz 2^n:

$$\bar{z} = 2^n - z$$

Ist z positiv, so gibt $\bar{z}$ die Darstellung von -z an und umgekehrt liefert $\bar{z}$ bei negativem z den zugehörigen positiven Wert.

Beispiel: Das Zweierkomplement von z = 0001 1100 (28_{10}) ist

$\bar{z} = 2^8 - z$

```
  1 0 0 0 0  0 0 0 0
-   0 0 0 1  1 1 0 0
-------------------
    1 1 1 0  0 1 0 0        -28(10)
```

Es gibt noch ein anderes einfaches Schema zur Ermittlung des Zweierkomplements: man invertiert die Dualzahl z bitweise und addiert anschließend 1 hinzu.

Obiges Beispiel nach dieser Regel berechnet ergibt:

```
              0 0 0 1  1 1 0 0
invertiert    1 1 1 0  0 0 1 1
            +                1
-------------------------------
              1 1 1 0  0 1 0 0        -28(10)
```

Die Aufteilung des Zahlenbereichs ist beim Zweierkomplement nicht symmetrisch zu Null. Bei n = 8 geht der Zahlenbereich von -128 bis +127. Das höchstwertige Bit b_7 kann als Vorzeichen betrachtet werden, b_7 = 0 für positive und b_7 = 1 für negative Zahlen.

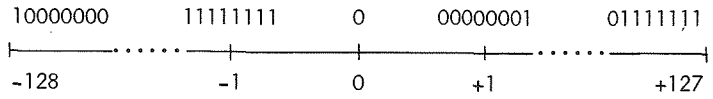


Bild 4.3 Zahlenbereich für n = 8 bei Zweierkomplement

dezimal	dual	hexadezimal
-1	1111 1111	FF
-2	1111 1110	FE
-3	1111 1101	FD
-4	1111 1100	FC
-5	1111 1011	FB
-6	1111 1010	FA
-7	1111 1001	F9
-8	1111 1000	F8
-9	1111 0111	F7
-10	1111 0110	F6

Tabelle 4.4 Die Zahlen von -1 bis -10 im Zweierkomplement

Rechnerintern wird die Subtraktion auf die Addition des Zweierkomplements zurückgeführt. Hierbei bleibt ein Übertrag an der höchsten Stelle unberücksichtigt.

Beispiel: 35 - 7 = 28 durch Addition des Zweierkomplements von -7 (siehe Tabelle 4.4)

```
  0 0 1 0  0 0 1 1        35
+ 1 1 1 1  1 0 0 1      - 7
-----------------------------
  0 0 0 1  1 1 0 0        28
```

4.3. Hexadezimalsystem

Die Basis des Hexadezimalsystems ist 16. Es hat neben dem Dualsystem eine besondere Bedeutung. Eine hexadezimale Ziffer gibt durch ein Zeichen den Zustand von vier Bits wieder. Daher auch die Bezeichnung Tetrade. Seine Beherrschung ist für die Arbeit mit dem TV-Computersystem wichtig. Adressen und Zahlen sind in hexadezimaler Form einzugeben.
Die 16 Ziffern des Hexadezimalsystems sind:

0, 1, 2, ... 8, 9, A, B, C, D, E, F.

Die Buchstaben A bis F entsprechen den Dezimalzahlen 10 - 15.

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

Die Umwandlung vom Hexadezimalsystem ins Dezimalsystem folgt dem schon bekannten Schema:

Beispiel: $4B_{16} = 4 \cdot 16^1 + B \cdot 16^0$

$= 64 + 11$

$= 75_{10}$

Für die Umrechnung dezimal ⟶ hexadezimal setzt man ebenfalls das Divisionsverfahren ein, wobei jetzt durch die Basis 16 dividiert wird. Die Reste der Division größer 9 sind in die entsprechenden hexadezimalen Ziffern umzusetzen. Dann ergeben die Reste von unten nach oben gelesen die gesuchte Zahl.

Beispiel: 435_{10}

435 : 16 = 27 Rest 3

27 : 16 = 1 " 11 = B ↑ $1B3_{16}$

1 : 16 = 0 " 1

Probe: $1B3_{16} = 1 \cdot 16^2 + 11 \cdot 16^1 + 3 \cdot 16^0$

$= 256 + 176 + 3$

$= 435_{10}$

4.4. Zusammenhang Dual- Hexadezimalzahlen

Die Umwandlung hexadezimal ⟶ dual ist einfach dadurch gegeben, daß man jede Tetrade durch die entsprechende vierstellige Dualzahl wiedergibt.

$8D_{16} = 1000\ 1101_2$

Umgekehrt erhält man aus einer Dualzahl die Hexadezimalzahl indem man von rechts nach links jeweils 4 Stellen zusammenfaßt und die betreffende Tetrade bestimmt. Die Tetradenform bietet also eine kürzere Schreibweise für Dualzahlen.

$1010/0110/1111_2 = A6F_{16}$

Wir geben nochmals eine Übersicht über die besprochenen Zahlensysteme:

Bezeichnung	Basis	Ziffern
dual	2	0, 1
dezimal	10	0, 1, 2, ... 8, 9
hexadezimal	16	0, 1, 2, ... , 9, A, B, C, D, E, F

4.5. Binär codierte Dezimalzahlen (BCD)

Neben den bisher behandelten Zahlensystemen kann der Rechner auch im Dezimalsystem arbeiten, allerdings ist die Realisierung der arithmetischen Operationen aufwendiger als im Dualsystem. Wie in 1.2. bereits erläutert, ist für die rechnerinterne Verarbeitung jeder Zeichenvorrat binär zu verschlüsseln. Für die Codierung der zehn Dezimalziffern 0 - 9 werden vier Bits benötigt, wobei von den 16 möglichen Kombinationen nur 10 belegt sind. Wird in einem Byte im linken und rechten Halbbyte je eine Dezimalziffer dargestellt, so spricht man von einer gepackten BCD-Darstellung, nimmt ein Byte nur eine Ziffer auf, von ungepackter Darstellung. Wie der Mikroprozessor 6800 die Arithmetik bei BCD-Zahlen realisiert, behandelt Abschnitt 5.7.6.

dual	dezimal	hexadezimal	BCD
0000	0	0	0000
0001	1	1	0001
0010	2	2	0010
0011	3	3	0011
0100	4	4	0100
0101	5	5	0101
0110	6	6	0110
0111	7	7	0111
1000	8	8	1000
1001	9	9	1001
1010	10	A	0001 0000
1011	11	B	0001 0001
1100	12	C	0001 0010
1101	13	D	0001 0011
1110	14	E	0001 0100
1111	15	F	0001 0101

Tabelle 4.5 Dual-, Dezimal-, Hexadezimal- und BCD-Zahlen von 0 - 15

5. Die Befehle des M 6800

5.1. Allgemeines

Eine alphabetische Übersicht über die verfügbaren Befehle gibt Anhang A. Alle für die Programmierung im Maschinencode notwendigen Angaben stehen in kompakter Form in der Befehlsliste in Anhang B. Es ist dort für jeden Befehl aufgeführt:

- o die Wirkung des Befehls in einer Kurzschreibweise,
- o die Codierung, aufgegliedert nach den verschiedenen Adressierungsarten,
- o die Länge je nach Adressierungsart,
- o die Wirkung auf das Bedingungsregister.

In diesem Kapitel beschreiben wir die Wirkung ausführlicher und erwähnen Besonderheiten und Anwendungsfälle. Die Auswirkung auf das Bedingungsregister wird nicht immer erwähnt.

Es sei grundsätzlich darauf hingewiesen, daß Sie die Wirkung jedes Befehls durch ein kurzes Programm in Einzelschrittausführung anhand der Registeranzeige nachprüfen können. Die Beherrschung des Befehlsrepertoires ist Voraussetzung für effektives Programmieren.

Die Abarbeitung eines Befehls läuft in folgenden Schritten ab: Der Befehlszähler weist auf den auszuführenden Befehl. In der Befehlsholphase wird das erste Byte - der Operationscode - ausgelesen und interpretiert. Im Operationscode ist die Operation und der Adressierungsmodus verschlüsselt. Anschließend wird - sofern es kein Registerbefehl ist - die Adresse ermittelt und der Operand beschafft. Es folgt die Ausführung der Operation. Gleichzeitig wird der Befehlszähler je nach Befehlslänge um 1, 2 oder 3 erhöht.

5.2. Bedingungsregister

Während der Ausführung von Befehlen können Markierungsbits im Bedingungsregister gesetzt oder gelöscht werden. Diese zeigen Eigenschaften des Ergebnisses an und können über bedingte Sprungbefehle abgefragt werden.

b_5	b_4	b_3	b_2	b_1	b_0
H	I	N	Z	V	C

Bedingungsregister

H = Halbübertrag (Half-carry)
Wird gesetzt, wenn bei einer Addition ein Übertrag von b_3 nach b_4 auftritt, andernfalls gelöscht. Der Halbübertrag ist für die Dezimalangleichung DAA von Bedeutung.

I = Interruptmaske (Interrupt Mask)
Wird durch einen Hardware- oder Softwareinterrupt, bei Programmstart und durch Befehl SEI gesetzt. Befehl CLI löscht Bit I.

N = Negativ (Negative)
N hat den gleichen Wert wie Bit 7 des Ergebnisses.

Z = Null (Zero)
Z ist gesetzt, wenn das Ergebnis = 0 (Bit 0 - Bit 7) und sonst gelöscht.

V = Überlauf (OVerflow)
V wird gesetzt, wenn bei einer arithmetischen Operation ein Überlauf auftritt, sonst gelöscht.

C = Übertrag (Carry)
C wird bei einem Übertrag vom höchstwertigen Bit b7 gesetzt, andernfalls gelöscht.

5.3. Adressierungsarten

5.3.1. Allgemeines

Die Adressierungsart bestimmt den Zugriff auf den Operanden. Für Operanden in Speicher gibt es

- die direkte Adressierung für den Speicherbereich von 0 - 255,
- die extended Adressierung mit einer 16-Bit-Adresse für den gesamten Adreßraum von 64 k,
- die indizierte Adressierung für die dynamische Adressenberechberechnung.

Für Operanden in Registern ist die Adressierung implizit durch den Operationscode gegeben. Konstante Werte stehen im Immediate-Modus unmittelbar im Adreßteil. Programmadressen in Sprungbefehlen können relativ zum aktuellen Befehlszählerstand oder ebenfalls durch eine 16-Bit-Adresse oder indiziert adressiert werden.

Man beachte, daß die Adressen in hexadekadischer Form einzugeben sind!

5.3.2. Implizite Adressierung

Die implizite Adressierung spricht

- einen Operanden in einem Register,
- zwei Operanden in zwei Registern,
- Operanden im Stack

an.

Der Operationscode spezifiziert die Operation und die Register, so daß keine weiteren Adreßangaben benötigt werden. Die Befehle mit impliziter Adressierung sind 1 Byte lang.

Beispiele sind: ABA, TAB, CLI, INX, TXS, PUL, RTS.

5.3.3. Immediate Adressierung

Bei diesem Adressierungsmodus gibt der Adreßteil den Wert des Operanden unmittelbar an.

Abhängig davon, ob sich der Befehl auf ein 8-Bit-Register (ACCA, ACCB) oder ein 16-Bit-Register (X, SP) bezieht, sind die entsprechenden Befehle 2 oder 3 Bytes lang. 3 Bytes benötigen die Befehle LDX, LDS und CPX bei immediate Adressierung.

Man wendet diesen Adressierungsmodus bei der Verarbeitung von konstanten Werten an, z.B. beim Laden von Adressen.

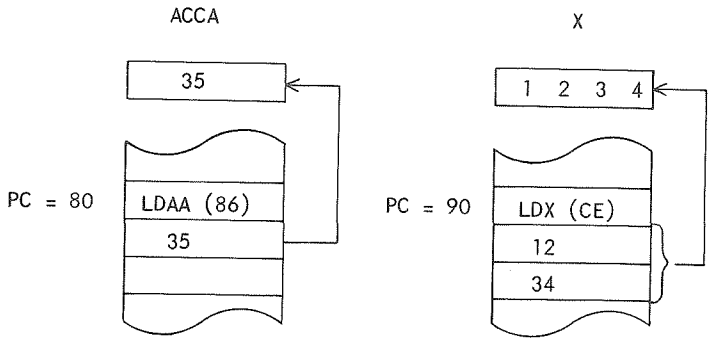


Bild 5.1 Immediate Adressierung

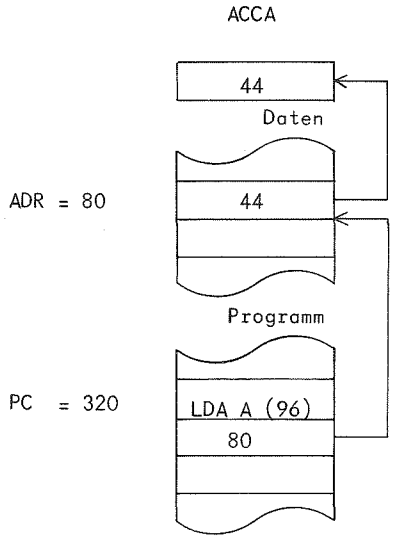
5.3.4. Direkte Adressierung

Befehle mit direktem Adressierungsmodus belegen immer 2 Bytes. Byte 1 gibt den Operationscode, Byte 2 den Adreßteil an. Im Adreßteil steht die Adresse eines Operanden in Speicherbereich von 0 - 255. Der 8 Bit Adreßteil wird also als positive Zahl aufgefaßt. Im TV-Computersystem entspricht dieser Speicherbereich den Seiten 0 bis 7.

Häufiger Gebrauch der direkten Adressierung wirkt sich günstig auf die Programmlänge aus. Es empfiehlt sich daher, bei längeren Programmen Daten und Hilfszellen an den Anfang des Speichers zu legen.

Für einige Befehle gibt es den direkten Adressierungsmodus nicht. In diesem Fall muß eine Adresse immer durch 16 Bit angegeben werden. Es sind dies die Befehle:

CLR, COM, NEG, DEC, INC, ROL, ROR, ASR, LSR, TST, JMP, JSR.

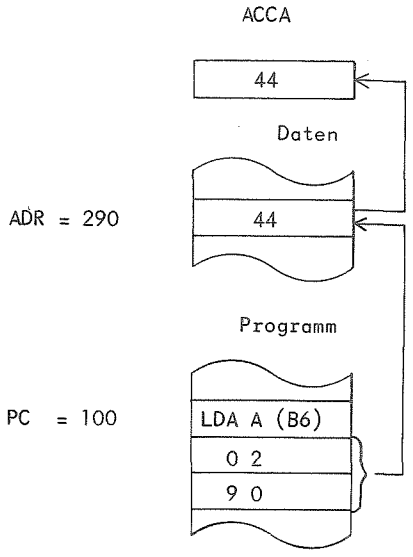


$0 \leq ADR \leq 255$

Bild 5.2
Direkte Adressierung

5.3.5. Extended Adressierung

Befehle in diesem Adressierungsmodus belegen immer 3 Bytes. Byte 1 enthält den Operationscode, die Bytes 2 und 3 nehmen eine 16-Bit-Adresse auf. In Byte 2 steht der höherwertige, in Byte 3 der niederwertige Anteil der Adresse. Damit sind 64 k Speicher adressierbar.



$0 \leq ADR \leq 65\,535$

Bild 5.3 Extended Adressierung

5.3.6. Indizierte Adressierung

Befehle im indizierten Adressierungsmodus belegen immer 2 Bytes. Byte 1 enthält den Operationscode, Byte 2 den Adreßteil.

Die effektive Operandenadresse wird jeweils dynamisch während der Programmausführung als Summe von Indexregister und Adreßteil ermittelt.

Mit X = aktueller Inhalt des Indexregisters
A = 8-Bit Adreßteil des Befehls

gilt für die Operandenadresse ADR die Beziehung

ADR = X + A

Der Adreßteil wird hierbei als positive Zahl im Bereich von 0 - 255 aufgefaßt. (Unterschied zur relativen Adressierung!)
Mithilfe der indizierenden Adressierung lassen sich durch Verändern des Indexregisters während der Programmausführung (Befehle LDX, INX, DEX, TSX) variable Adressen bilden.

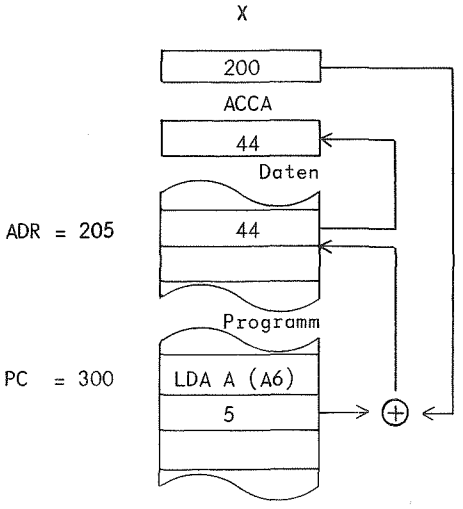


Bild 5.4
Indizierte Adressierung

5.3.7. Relative Adressierung

Bei der relativen Adressierung wird eine Speicherzelle relativ zum aktuellen Befehlszählerstand adressiert. Die relative Adressierung tritt nur bei Sprungbefehlen auf. Sie ist bei den bedingten Sprüngen die einzig mögliche Adressierungsart.

Die Befehlslänge beträgt immer 2 Bytes. Das erste Byte gibt den Operationscode, das zweite die Relativadresse an. Der Adreßteil wird dabei als ganze Zahl mit Vorzeichen, dargestellt im Zweierkomplement, aufgefaßt. Die Relativadresse liegt also im Bereich von - 128 bis + 127. Damit ist die Möglichkeit von Vorwärts- und Rückwärtssprüngen gegeben.

Mit PC = Adresse erstes Byte des Sprungbefehls
R = Relativadresse ($-128 \leq R \leq +127$)
ADR = Adresse des Sprungziels

muß folgende Beziehung gelten:

$$(PC + 2) - 128 \leq ADR \leq (PC + 2) + 127$$

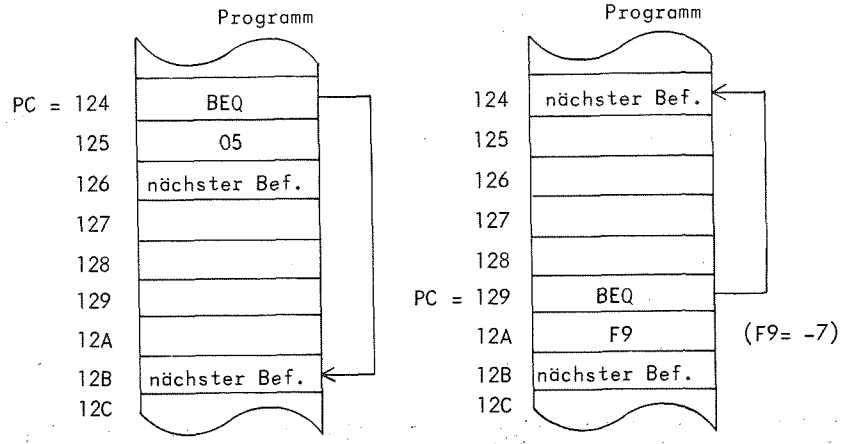
bzw.

$$PC - 126 \leq ADR \leq PC + 129$$

Für die Adresse des Sprungziels gilt:

$$ADR = (PC + 2) + R$$

Der nachfolgende Befehl hat also die Relativadresse 0, der Sprungbefehl selbst die Relativadresse - 2. Damit ist der dynamische Stop (3.4) durch BRA - 2 bzw. die Internform 20 FE erklärt. Bei Sprungzielen außerhalb des relativ adressierbaren Bereichs muß man unbedingte Sprungbefehle einsetzen. In Anhang C findet sich eine Tabelle aus der insbesondere die negativen Sprungweiten abgelesen werden können.



a) positive Relativadresse b) negative Relativadresse

Bild 5.5 Relative Adressierung

Falls die Sprungbedingung erfüllt ist, fährt das Programm

im Fall a) mit dem Befehl auf Adresse 12B,
im Fall b) auf Adresse 124 fort.

5.4. Transportbefehle

5.4.1. 8 Bit Transportbefehle

Die Akkumulatoren ACCA und ACCB sind die Rechenregister des Prozessors. Die Transportbefehle, die man weiter in Lade- und Speicherbefehle unterteilen kann, dienen zum Bereitstellen der Operanden für die weitere Verarbeitung bzw. zum Abspeichern von Ergebnissen im Speicher. Der Transport von einer Speicherzelle zu einer anderen ist nur über die Zwischenschaltung eines Registers möglich. Für diesen Befehlstyp sind die meisten Adressierungsarten (Ausnahme: relative Adressierung) erklärt.

LDAA : M ⟶ A (Load Accumulator)
LDAB : M ⟶ B

Der Inhalt der adressierten Speicherzelle, bzw. der Adreßteil beim Immediate-Modus, wird in den Akkumulator geladen. Die Bits N und Z des Bedingungsregisters werden je nach geladenem Wert gesetzt oder gelöscht, die Überlaufanzeige V wird auf 0 gesetzt. Die Bits H, I und C bleiben unverändert. Ein Ladebefehl überschreibt den bisherigen Inhalt des Registers, der Inhalt der angesprochenen Speicherzelle bleibt unverändert.

STAA : A ⟶ M (Store Accumulator)
STAB : B ⟶ M

Die Speicherbefehle überschreiben den bisherigen Inhalt des adressierten Speichers mit dem aktuellen Inhalt des Registers. Der Registerinhalt bleibt unverändert.

Zum Transport zwischen den Akkumulatoren dienen die Befehle

TAB : A ⟶ B (Transfer Accumulators)
TBA : B ⟶ A

Nach Ausführung dieser Befehle haben ACCA und ACCB den gleichen Inhalt.

Den Zugriff auf Daten im Stack (2.3.2) ermöglichen die Befehle PSH und PUL. Voraussetzung ist, daß das Stackregister am Programmanfang mit der Anfangsadresse des Stacks geladen wurde. Andernfalls kommt es bei PSH zu unkontrolliertem Abspeichern irgendwo im Speicher.

PSHA : A ⟶ M_{SP}, SP - 1 ⟶ SP (Push Data)
PSHB : B ⟶ M_{SP}, SP - 1 ⟶ SP

In der durch das Stackregister adressierten Zelle wird der Inhalt des Akkumulators abgespeichert. Anschließend vermindert sich der Inhalt des Stackregisters um eins. Es weist damit wieder auf die erste freie Stelle im Stack.

PULA : SP + 1 ⟶ SP, M_{SP} ⟶ A (Pull Data)
PULB : SP + 1 ⟶ SP, M_{SP} ⟶ B

Zuerst wird das Stackregister um eins erhöht. Es weist damit auf die oberste belegte Zelle, deren Inhalt anschließend in den Akkumulator geladen wird. Sie ist nach Ausführung von PUL die erste freie Zelle im Stack.

Das Bedingungsregister bleibt bei PSH und PUL im Unterschied zu den anderen Transportbefehlen unverändert!

PSH und PUL sind komplementäre Operationen, die bei korrekter Buchhaltung im Stack stets in paarweiser Zuordnung auftreten.

Man könnte sie mit den Klammern eines arithmetischen Ausdrucks vergleichen. Öffnende und schließende Klammern gehören paarweise zusammen. Bei diesem Vergleich entspricht PSH der öffnenden Klammer - es muß erst etwas abgespeichert werden, bevor es abgeholt werden kann - und PUL der schließenden Klammer.

Zwei weitere Transportbefehle gibt es zum Laden bzw. Abspeichern des Bedingungsregisters. Dies kann nur über den ACCA geschehen.

TAP : A ⟶ CCR (Transfer Accumulator A to
TPA : CCR ⟶ A Processor Condition Codes Register)

Zum Befehl TAP ist anzumerken, daß gleichgültig wie die Bits 6 und 7 im ACCA gesetzt sind, die entsprechenden Bits im Bedingungsregister immer mit 1 besetzt werden. Diesen zwei Binärstellen ist keine Bedeutung zugeordnet, sie sind stets mit 1 besetzt.

5.4.2. 16 Bit Transportbefehle

Bei den Transportbefehlen der 16-Bit-Register X und SP gilt für die Interpretation des Adreßteils allgemein, daß sich die Adresse auf das höherwertige Byte bezieht. Das adressierte Byte wird in das höherwertige, das darauffolgende Byte in das niederwertige Byte des Registers geladen. Analog verläuft das Abspeichern in zwei aufeinanderfolgende Bytes.

LDX : $M \longrightarrow X_H$, $(M + 1) \longrightarrow X_L$ (Load Index Register)
LDS : $M \longrightarrow SP_H$, $(M + 1) \longrightarrow SP_L$ (Load Stack Pointer)

STX : $X_H \longrightarrow M$, $X_L \longrightarrow (M + 1)$ (Store Index Register)
STS : $SP_H \longrightarrow M$, $SP_L \longrightarrow (M + 1)$ (Store Stack Pointer)

Auch für die beiden 16-Bit-Register gibt es den Transport zwischen den Registern.

TSX : $SP + 1 \longrightarrow X$ (Transfer Stack Pointer to Index Register)

Hier ist besonders zu beachten, daß der um eins erhöhte Inhalt des Stackregisters in das Indexregister transportiert wird. Das Stackregister bleibt unverändert.

Eine Begründung für die Addition von 1 ist, daß das Indexregister nach diesem Befehl auf das oberste belegte Byte des Stacks zeigt, auf dessen Inhalt man dann mit indizierter Adressierung zugreifen kann. Das Stackregister zeigt immer auf die erste freie Zelle. Befehl TSX setzt man beispielsweise in Unterprogrammen bei Parameterübergabe im Stack ein (6.6.2).

Für den Transport in der anderen Richtung gilt in Analogie, daß der um eines verminderte Inhalt des Indexregisters in das Stackregister geladen wird. Das Indexregister bleibt unverändert.

TXS : $X - 1 \longrightarrow SP$ (Transfer Index Register to Stack Pointer)

Beispiel:

Der Inhalt einer Zeile auf dem Bildschirm soll in die darunterliegende Zeile transportiert werden.

Damit der Einzelbefehlsablauf und die Veränderung des Datenbereichs gleichzeitig beobachtet werden können, wählen wir die Zeile 7 auf Seite 0, die nach Zeile 8 gespeichert werden soll. (7 und 8 beziehen sich auf die Tetraden am rechten Bildrand.) Auf Adressen bezogen lautet die Aufgabe:
Speichere den Inhalt der Zellen E und F nach 10 und 11.

Lösung 1: Die beiden Bytes werden nacheinander über Akkumulator A umgespeichert.

In der Spalte "Symbolische Adresse" haben wir hier die absoluten Adressen eingetragen.

Seite: 0

Transportiere Zeile 7 → Zeile 8 (auf Seite 0)

Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	Start	LDS	I	1F	8E	Stack liegt am Ende
1					00	von Seite 0
2					1F	
3		LDAA	D	E	96	Inhalt Adr. E → ACCA
4					0E	
5		STAA	D	10	97	ACCA → Adr. 10
6					10	
7		LDAA	D	F	96	Inhalt Adr. F → ACCA
8					0F	
9		STAA	D	11	97	ACCA → Adr. 11
A					11	
B	Stop	BRA	R	Stop	20	dynamischer Stop
C					FE	

Lösung 2: Umspeichern durch nur einen Lade- und Speicherbefehl über das Indexregister.

Für das Umspeichern größerer Bereiche ist diese Lösung in der Regel jedoch nicht anwendbar, da dann das Indexregister für die indizierte Adressierung benötigt wird.

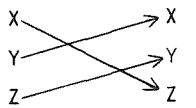
Seite 0

Transportiere Zeile 7 → Zeile 8 (auf Seite 0)

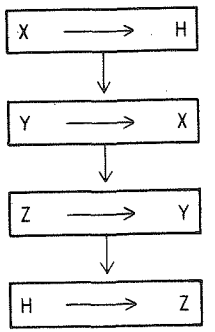
Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	Start	LDS	I	1F	8E	Stack liegt am Ende von
1					00	Seite 0
2					1F	
3		LDX	D	E	DE	Inhalt Adr. E, F → X
4					0E	
5		STX	D	10	DF	X → Adr. 10, 11
6					10	
7	Stop	BRA	R	Stop	20	dynamischer Stop
8					FE	

Beispiel:

Die Werte von X, Y und Z sollen zyklisch vertauscht werden. X, Y und Z seien die Adressen 12, 14 und 16 zugeordnet.



Wir zerlegen die Aufgabe in die Einzelschritte



Im Diagramm erscheint eine Hilfsgröße H, die den Ausgangswert von X aufnimmt und bis zum Abspeichern in Z aufbewahrt. Im Programm verwenden wir den ACCA als Zwischenspeicher für X, über ACCB läuft das Umspeichern von Y und Z.

Seite 0

Zyklisches Vertauschen

Bsp	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	Start	LDS	I	1F	8E	Stack liegt am Ende von
1					00	Seite 0
2					1F	
3		LDAA	D	X	96	X → ACCA
4					12	
5		LDAB	D	Y	D6	Y → ACCB
6					14	
7		STAB	D	X	D7	ACCB → X
8					12	
9		LDAB	D	Z	D6	Z → ACCB
A					16	
B		STAB	D	Y	D7	ACCB → Y
C					14	
D	.	STAA	D	Z	97	ACCA → Z
E					16	
F	Stop	BRA	R	Stop	20	dynamischer Stop
10					FE	
11						
12	X				11	
13						
14	Y				22	
15						
16	Z				33	

Probieren Sie diese kurzen Beispiele auf Seite 0 aus!
Startadresse = 0.

5.5. Setz- und Löschbefehle

Von den 6 Markierungsbits des Bedingungsregisters können drei - nämlich C, V und I - per Befehl direkt gesetzt oder gelöscht werden. H, Z und N sind nur indirekt durch Befehlsausführungen oder Befehl TAP (5.4.1) veränderbar.

CLC	:	0 ⟶ C	(Clear Carry)
SEC	:	1 ⟶ C	(Set Carry)
CLV	:	0 ⟶ V	(Clear Overflow)
SEV	:	1 ⟶ V	(Set Overflow)
CLI	:	0 ⟶ I	(Clear Interrupt Mask)
SEI	:	1 ⟶ I	(Set Interrupt Mask)

Anwendungen sind beispielsweise:

- o Definiertes Setzen von C vor Shiftbefehlen oder Addition mit Übertrag,
- o Löschen der Interruptmaske nach Programmstart, falls Interruptverarbeitung vorgesehen.

Ferner können die Akkumulatoren oder Speicherzellen mit Befehl CLR gelöscht werden.

CLR	:	0 ⟶ M	(Clear)

Befehl CLRA könnte beispielsweise auch durch LDAA 0 (immediate) ersetzt werden. Die Anwendung der CLR-Befehle ist kürzer und eleganter. Zum Löschen von Speicherzellen wird kein Register benötigt.

5.6. Shiftbefehle

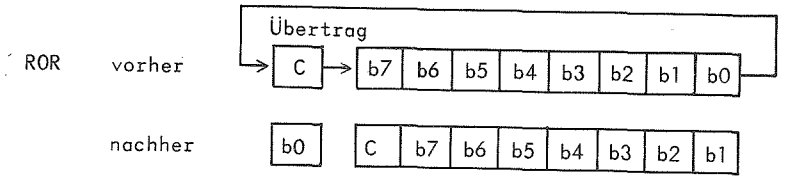
Es gibt 5 verschiedene Shiftbefehle:

- o Kreisshift rechts/links
- o Arithmetischer Rechts-/Linksshift
- o Logischer Rechtsshift.

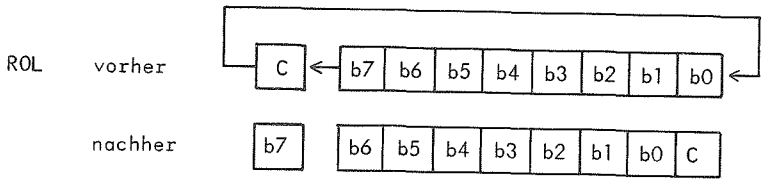
Sie können sich auf einen Akkumulator oder direkt auf den Speicher beziehen.

Man setzt Shiftbefehle bei der seriellen bitweisen Verarbeitung eines Wortes ein (siehe Beispielprogramm 7-Segmentanzeige). Außerdem für die Multiplikation und Division mit 2.

Kreisshift rechts: ROR (Rotate Right)



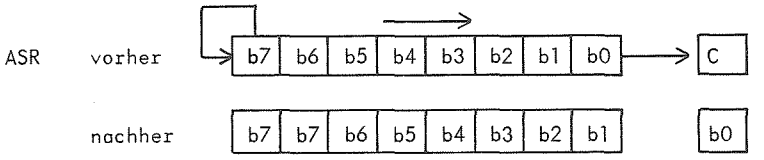
Kreisshift links: ROL (Rotate Left)



Beim Kreisshift ist immer die Übertragstelle C im Bedingungsregister beteiligt. Durch definiertes Setzen oder Löschen des Übertrags (SEC, CLC) vor dem Shiftbefehl kann man das "eingeshiftete" Bit bestimmen.

Nach dem Shiftbefehl kann man andererseits mit den Befehlen BCC bzw. BCL den Wert des "herausgeshifteten" Bits abfragen.

Arithmetischer Rechtsshift: ASR (Arithmetic Shift Right)

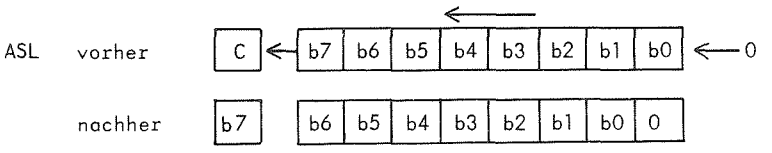


Beim arithmetischen Rechtsshift wird in die höchstwertige Binärstelle der ursprüngliche Wert wieder eingeschrieben. Damit führt dieser Befehl die Division durch 2 für positive und negative Zahlen im Zweierkomplement aus. In C steht anschließend das niederwertigste Bit, was gerade dem Rest bei der Division durch 2 entspricht.

Beispiel: 19 : 2 = 9 Rest 1

0001 0011 $\xrightarrow{ASR}$ 0000 1001, C = 1

Arithmetischer Linksshift: ASL (Arithmetic Shift Left)

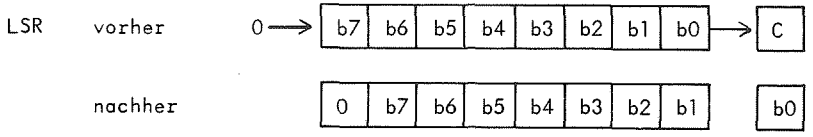


Befehl ASL multipliziert eine positive Zahl mit 2. Der Übertrag an der höchsten Binärstelle steht anschließend in C.

Beispiel: 19 · 2 = 38

0001 0011 —ASL→ 0010 0110

Logischer Rechtsshift (Logical Shift Right)



Der logische Rechtsshift unterscheidet sich vom arithmetischen dadurch, daß an der höchstwertigen Stelle immer eine Null eingeschrieben wird. Er führt die Division durch 2 für positive Zahlen aus, wobei in C wieder der Divisionsrest abzulesen ist.

5.7. Arithmetische Befehle

5.7.1. Zahlendarstellung

Der Mikroprozessor 6800 führt die binäre Addition von zwei 8-Bit-Zahlen in Zweierkomplementdarstellung aus. Der Befehlssatz zusammen mit der geeigneten Interpretation der Marken des Bedingungsresisters läßt jedoch arithmetische Operationen für 4 verschiedene Interpretationen der Zahlendarstellung zu.

1. Jedes Byte interpretiert man als ganz Zahl mit Vorzeichen im Zweierkomplement im Bereich von - 128 bis + 127:

$\pm 2^6$	2^5	2^4	2^3	2^2	2^1	2^0		
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	0	0	0	0	- 128 (Zweierkomplement)
1	1	1	1	1	1	1	1	- 1 "
0	0	0	0	0	0	0	0	0 "
0	0	0	0	0	0	0	1	1 "
0	1	1	1	1	1	1	1	+ 127 "

2. Jedes Byte interpretiert man als vorzeichenlose positive Dualzahl im Bereich von 0 bis 255:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	100
1	1	1	1	1	1	1	1	255

3. Jedes Byte enthält eine 4-Bit BCD Ziffer in den 4 niederwertigen Bits. Die 4 höherwertigen Bits sind Null. Diese Darstellung der Zahlen von 0 bis 9 bezeichnet man als ungepackte BCD-Zahlen.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		
0	0	0	0	0	0	0	0	0	(BCD)
0	0	0	0	0	1	0	1	5	"
0	0	0	0	1	0	0	1	9	"

4. Jedes Byte enthält zwei BCD Ziffern. Bei dieser Interpretation können in einem Byte die Zahlen 0 bis 99 dargestellt werden.

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0		
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		
0	0	0	0	0	0	0	0	0	(BCD)
0	0	1	0	0	1	1	1	27	"
1	0	0	1	1	0	0	1	99	"

5.7.2. Überlaufanzeige V

Bei arithmetischen Operationen kann es zum Über- oder Unterschreiten des darstellbaren Zahlenbereichs kommen. Dies zeigt Bit V des Bedingungsregisters an. Für das Setzen und Löschen von V liegt die Interpretation der Operanden als Zahlen im Zweierkomplement, also ein Zahlenbereich von -128 bis +127, zugrunde.

Einen Überlauf bei Interpretation der Operanden als positive Zahlen (0 bis 255) zeigt der Übertrag C an.

Beispielsweise ist V nach der Addition 120 + 95 gesetzt, da das Ergebnis 215 größer als die größte darstellbare Zahl ist.

Gleiches gilt für den negativen Zahlenbereich. Die Subtraktion - 100 - 50 setzt ebenfalls die Überlaufanzeige.

Ein Überlauf wird intern festgestellt, indem die Vorzeichenbits a_7 und b_7 der Operanden und das Vorzeichenbit e_7 des Ergebnisses untersucht werden. Wir diskutieren die verschiedenen Fälle für die Addition anhand Tabelle 5.6.

Nr.	a_7	b_7	e_7	V
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	0
7	1	1	0	1
8	1	1	1	0

Tabelle 5.6 Überlauf bei Addition

Wenn die Vorzeichen der Operanden verschieden sind, kann es keinen Überlauf geben. Durch die Addition einer positiven und einer negativen Zahl kann der Zahlenbereich nie über- oder unterschritten werden. V = 0 in Nr. 3, 4, 5 und 6.

Sind alle drei Vorzeichen gleich, so bleibt das Ergebnis im ursprünglichen Zahlenbereich. Kein Überlauf in Fall 1 und 8.

Übrig bleiben die beiden Fälle, in denen die Operanden das gleiche Vorzeichen haben und das Vorzeichen beim Ergebnis wechselt. Hier ist ein Überlauf eingetreten (Nr. 2 und 7).

Beispiel: 100 + 33

```
  0 1 1 0   0 1 0 0      A = 100
+ 0 0 1 0   0 0 0 1      B =  33
-----------------------
  1 0 0 0   0 1 0 1      E = -123 anstatt 133
                         V ist zur Überlaufanzeige
                         gesetzt!
```


5.7.3. Inkrementieren und Dekrementieren

Für die Addition und Subtraktion von 1 gibt es für alle Register Spezialbefehle. Desgleichen kann direkt im Speicher um eins erhöht oder vermindert werden. Diese Befehle sind für Zählzwecke besonders geeignet. Sie sind kürzer als die allgemeinen Additions- und Subtraktionsbefehle.

```
INC  :  M + 1 ⟶ M        (Increment)
DEC  :  M - 1 ⟶ M        (Decrement)
```

Für Indexregister und Stackregister sind dies die einzigen arithmetischen Operationen.

```
INX  :  X + 1 ⟶ X        (Increment Index Register)
DEX  :  X - 1 ⟶ X        (Decrement Index Register)

INS  :  SP + 1 ⟶ SP      (Increment Stack Pointer)
DES  :  SP - 1 ⟶ SP      (Decrement Stack Pointer)
```

Besteht die Aufgabe, das Indexregister um einen größeren Wert als 1 zu erhöhen, so kann dies bei kleineren Werten durch mehrere INX-Befehle (nur 1 Byte Speicherbedarf) oder allgemein durch eine einfache Schleife programmiert werden:

```
        LDAA   Zählwert
Schleife INX
        DECA
        BNE    Schleife
```

5.7.4. Addition und Subtraktion

Die Additionsbefehle sind 2-Operanden-Befehle. Es werden zwei 8-Bit-Dualzahlen nach den Regeln der dualen Addition verknüpft (siehe 4.2.3).

Befehl ADD addiert den Inhalt eines Akkumulators und den Inhalt einer Speicherzelle mit Ergebnis im betreffenden Akkumulator. Ein weiterer Befehl addiert die beiden Akkumulatoren A und B mit Ergebnis in ACCA.

Diese Befehle setzen oder löschen im Bedingungsregister die Stellen H, N, Z, V und C.

ADD : ACCX + M ⟶ ACCX (Add)
ABA : A + B ⟶ A (Add Accumulators)

Beispiel: ABA (Interpretation als positive Zahlen)

ACCA	0 1 1 0 1 1 0 0	108
ACCB	+ 0 0 0 1 0 1 0 1	+ 21
ACCA	1 0 0 0 0 0 0 1	129

C = 0
V = 1
H = 1
N = 1
Z = 0

Ein weiterer Additionsbefehl berücksichtigt das Übertragsbit C. Befehl ADC setzt man beispielsweise bei der Addition doppeltlanger Zahlen ein (siehe 5.7.5).

ADC : ACCX + M + C ⟶ ACCX (Add with Carry)

Beispiel: ADCA

ACCA	0 0 1 0 0 0 0 1	33
Operand in M	0 0 1 0 1 1 1 1	47
Übertrag C (vorher)	+ 1	+ 1
ACCA	0 1 0 1 0 0 0 1	81

C = 0
V = 0
H = 1
N = 0
Z = 0

Die den drei Additionsbefehlen entsprechenden Subtraktionsbefehle sind:

SUB : ACCX - M ⟶ ACCX (Subtract)
SBA : A - B ⟶ A (Subtract Accumulators)
SBC : ACCX - M - C ⟶ ACCX (Subtract with Carry)

Beispiel: SBA (Interpretation als Zweierkomplement)

ACCA	0 0 1 1 0 0 1 0	50
ACCB	- 0 1 1 0 0 1 0 0	- 100
ACCA	1 1 0 0 1 1 1 0	- 50

C = 1
V = 0
N = 1
Z = 0

Die Vorzeichenumkehr von Zahlen im Zweierkomplement führt der 1-Operanden-Befehl NEG aus. Er kann sich auf einen Akkumulator oder direkt auf den Speicher beziehen.

NEG : 0 - M ⟶ M (Negate)

Beispiel: NEGA

	0 0 0 0 0 0 0 0	
ACCA	- 0 0 1 1 0 0 1 0	50
ACCA	1 1 0 0 1 1 1 0	- 50
C = 1		
V = 0		
N = 1		
Z = 0		

Man beachte, daß -128 (1000 0000) durch die Bildung des Zweierkomplements unverändert bleibt.

5.7.5 Doppeltlange Addition und Subtraktion

Häufig reicht der in einem Byte darstellbare Zahlenbereich nicht aus. Man setzt dann zwei aufeinanderfolgende Bytes zu einer doppeltlangen Zahl zusammen. Die Addition und Subtraktion von zwei 16-Bit-Zahlen muß dann durch eine geeignete Befehlsfolge realisiert werden.

Beispiel: X + Y ⟶ Z

X, Y und Z seien doppeltlange Dualzahlen. Das höherwertige Byte werde durch X, Y und Z, das niederwertige durch X + 1, Y + 1 und Z + 1 adressiert. Die folgende Befehlsfolge führt die Operation aus:


```
LDAA   X + 1
LDAB   X
ADDA   Y + 1
ADCB   Y
STAA   Z + 1
STAB   Z
```

Wesentlich ist, daß zuerst das niederwertige Byte und anschließend unter Berücksichtigung eines eventuellen Übertrags das höherwertige Byte addiert wird.

Die doppeltlange Subtraktion X - Y ⟶ Z wird in ähnlicher Weise durch die nachstehende Befehlsfolge ausgeführt:

```
LDAA   X + 1
LDAB   X
SUBA   Y + 1
SUBC   Y
STAA   Z + 1
STAB   Z
```

5.7.6. Dezimalarithmetik

Der Prozessor verfügt über einen Spezialbefehl "Dezimalangleichung" DAA, der im Zusammenwirken mit den Additionsbefehlen und dem Bedingungsregister die Dezimalarithmetik ermöglicht. Eine korrekte Anwendung dieses Befehls ist sichergestellt, wenn

1) der Befehl DAA unmittelbar auf einen der Befehle ADD, ABA oder ADC folgt. Diese Befehle setzen bzw. löschen H und C in der erforderlichen Weise.

2) beide Operanden der Befehle ADD, ABA oder ADC Zahlen in

ACCA	0 1 1 0 0 1 1 1	67 in BCD-Darstellung
ACCB	+ 0 0 0 1 0 1 0 0	14 in BCD-Darstellung
ACCA	0 1 1 1 1 0 1 1	7B ist keine zulässige BCD-Zahl!
	C = 0	
	H = 0	

Befehl DAA addiert laut Tabelle 5.7 die BCD-Zahl 06 zu ACCA hinzu.

ACCA	0 1 1 1 1 0 1 1	7B
	+ 0 0 0 0 0 1 1 0	06
	1 0 0 0 0 0 0 1	81 Ergebnis in BCD-Darstellung

5.7.6.2. Dezimale Subtraktion

Es gibt keinen Befehl, der die Dezimalangleichung nach der binären Subtraktion von BCD-Zahlen erledigt. Die dezimale Subtraktion muß deshalb auf die Addition des Zehnerkomplements zurückgeführt werden.

Das Zehnerkomplement einer Dezimalzahl X ($0 \leq X \leq 99$) ist 100 - X.

Wir bilden es hier durch Berechnung von 99 - X + 1, da 100 in BCD-Darstellung nicht mehr in einem Byte darstellbar ist.

Die dezimale Subtraktion X - Y führen wir zurück auf
X + (99 - Y + 1) - 100.

```
LDAA  99      (immediate)
SUBA  Y       99 - Y ⟶ ACCA
SEC           1 ⟶ C
ADCA  X       ACCA + C + X ⟶ ACCA
DAA           Dezimalangleichung (- 100)
```

Die Subtraktion von 100 erfolgt durch DAA, indem der Übertrag C nach Ausführung von DAA unberücksichtigt bleibt. Ist das Ergebnis der Subtraktion positiv, so steht in ACCA der Wert in BCD-Darstellung. Ist es negativ, so steht in ACCA das Zehnerkomplement des Ergebnisses.

Beispielsweise: 33 - 50 = - 17. In ACCA steht 83.

5.7.7. Multiplikation und Division

Der Mikroprozessor 6800 hat keine Befehle für Multiplikation und Division. Diese Operationen müssen durch geeignete Befehlsfolgen realisiert werden.

In der Literatur /1/ ist ein Programm für die Multiplikation von zwei 16-Bit-Zahlen im Zweierkomplement sowie ein Divisionsprogramm mit 16-Bit-Dividend und 8-Bit-Divisor beschrieben.

Wir geben hier nur ein Unterprogramm an, das die Multiplikation von zwei positiven 8-Bit-Zahlen leistet. Das Ergebnis ist 16 Bits lang.

Die Multiplikation kann auf die Operationen Addition und Stellenverschiebung (Shift) zurückgeführt werden (siehe 4.2.3).

Um dieses Unterprogramm für den Einbau in verschiedene Hauptprogramme möglichst unabhängig zu machen, übergeben wir alle Parameter in den Registern, anstatt feste Speicherplätze für Multiplikand, Multiplikator und Ergebnis vorzusehen.

Parameter:
- ACCA = Multiplikator
- ACCB = Multiplikand
- X = Adresse Produkt (höherwertiges Byte)

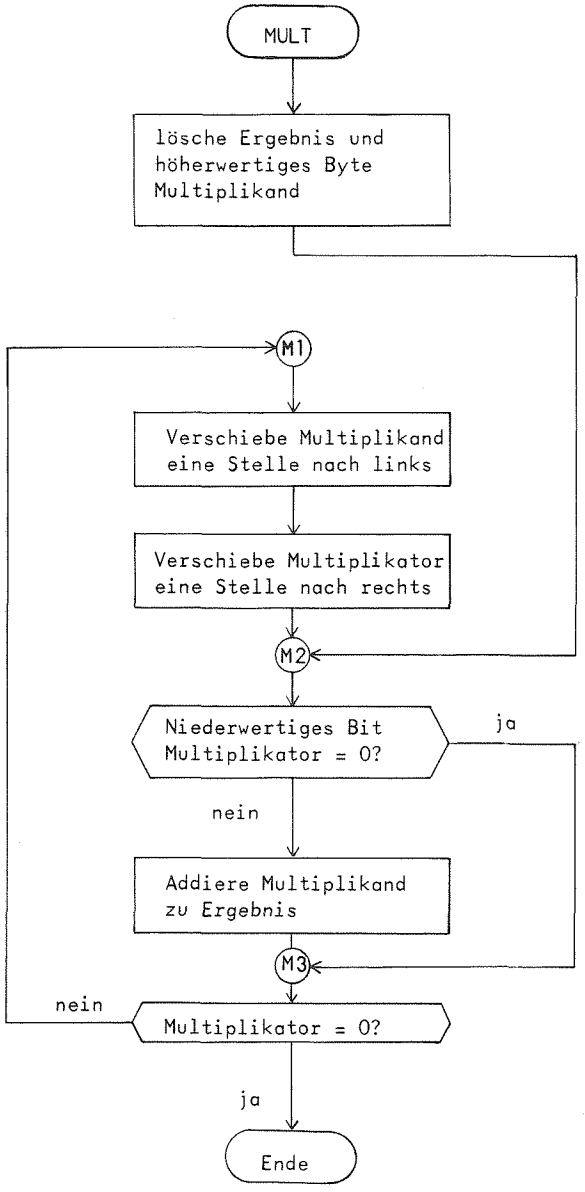


Bild 5.8 Multiplikationsunterprogramm

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Multiplikations-Unterprogramm (8x8 Bits)

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	MULT	STAB	D	Mult1	D7	Multiplikanden abspeichern
1	1					25	
2	2		CLR	E	Mult2	7F	Hilfszelle für höherwertigen Anteil des geshifteten Multiplikanden löschen
3	3					00	
4	4					24	die beiden Ergebnisbytes mit 0 vorbesetzen
5	5		CLR	X	0	6F	
6	6					00	
7	7		CLR	X	1	6F	
8	8					01	
9	9		BRA	R	M2	20	
10	A					06	
11	B	M1	ASL	E	Mult1	78	Shifte Multiplikanden 1 Stelle nach links (doppelt lang)
12	C					00	
13	D					25	
14	E		ROL	E	Mult2	79	
15	F					00	
16	10					24	Shifte Multiplikator 1 Stelle nach rechts falls C=0, keine Addition
17	11	M2	LSRA	R		44	
18	12		BCC	R	M3	24	
19	13					0D	
20	14		LDAB	D	Mult1	D6	Addiere niederwertiges Byte des geshifteten Multiplikanden zum niederwertigen Byte des Ergebnisses
21	15					25	
22	16		ADDB	X	1	EB	
23	17					01	
24	18		STAB	X		E7	
25	19				1	01	
26	1A		LDAB	D	Mult2	D6	Addiere höherwertige Bytes und berücksichtige Übertrag
27	1B					24	
28	1C		ADCB	X	0	E9	
29	1D					00	
30	1E		STAB	X	0	E7	
31	1F					00	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Aufgabe: Multiplikations-UP

Seite: 1

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0						
1	1	M3	TSTA	R	M1	4D	Multiplikator = 0 ?
2	2		BNE			26	nein, weitere Addition
3	3					E8	
4	4		RTS			39	Rücksprung aus UP
5	5	Mult2					Hilfszellen für Multiplikanden
6	6	Mult1					
7	7						
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

0375-TV

5.8. Logische Befehle

Es gibt Befehle für die vier logischen Operationen Negation, UND, ODER und exklusives ODER.

Negation

Die Negation ist ein Ein-Operandenbefehl. Alle Bits eines Akkumulators oder einer Speicherzelle werden invertiert.

COM : $\overline{M} \longrightarrow M$ (Complement)

	COM
0	1
1	0

Dieser Befehl darf nicht mit Befehl NEG verwechselt werden, der eine Zahl in Zweierkomplementdarstellung negiert (siehe 5.7.4). Die logische Negation bezeichnet man auch als Einerkomplement.

Die drei anderen logischen Befehle verknüpfen 2 Operanden. Der erste Operand muß in einem Akkumulator bereitgestellt sein, der nach der Befehlsausführung auch das Ergebnis enthält. Der zweite Operand wird aus dem Speicher gemäß Adressierungsmodus beschafft.

UND - Verknüpfung

AND : ACCX · M $\longrightarrow$ ACCX (Logical AND)

AND verknüpft die Operanden bitweise nach folgender Regel:

Op1	Op2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Man wendet Befehl AND beispielsweise an, um gewisse Bitpositionen - unabhängig vom vorherigen Wert - auf Null zu setzen, während andere Bits unverändert bleiben sollen. Den zweiten Operanden bezeichnet man in diesem Zusammenhang oft als Maske.

Beispiel: Die drei rechten Bits des Akkumulators A sollen auf Null gesetzt werden. Mit der Maske 1111 1000 und Befehl ANDA löst man diese Aufgabe.

Befehl BIT führt ebenfalls eine UND-Verknüpfung aus. In Unterschied zu AND setzt diese Operation nur N und Z im Bedingungsregister ohne die Operanden zu verändern. BIT eignet sich für Abfragen von Bitpositionen.

BIT : ACCX · M (Bit Test)

Beispiel: Es soll geprüft werden, ob in ACCA eine gerade oder ungerade Zahl steht. Mit unverändertem ACCA sind zwei verschiedene Programmzweige zu durchlaufen.

BITA mit Maske 0000 0001 setzt im Bedingungsregister Z = 1 bei gerader und Z = 0 bei ungerader Zahl.

```
            BITA  1     (immediate)
            BNE         ungerade
gerade      ---
```


ODER - Verknüpfung

ORA : ACCX + M ⟶ ACCX (Inclusive OR)

Befehl ORA verknüpft die Operanden bitweise gemäß:

Op1	Op2	ORA
0	0	0
0	1	1
1	0	1
1	1	1

Mit Befehl ORA kann man - durch die Maske gesteuert - gewisse Bits unabhängig vom vorherigen Wert auf 1 setzen.

Exklusives ODER

EOR : ACCX ⊕ M ⟶ ACCX (Exclusive OR)

Die bitweise Verknüpfung der beiden Operanden erfolgt gemäß:

Op1	Op2	EOR
0	0	0
0	1	1
1	0	1
1	1	0

Mit Hilfe von EOR und anschließender Negation (COM) kann man die logische Operation der Äquivalenz bilden.

5.9. Vergleichsbefehle

Die Vergleichsbefehle setzen entsprechend dem Vergleichsergebnis verschiedene Bits des Bedingungsregisters ohne die Operanden zu verändern.

CMP : ACCX - M (Compare)

Befehl CMP vergleicht der Inhalt eines Akkumulators mit dem Inhalt einer Speicherzelle indem intern die Differenz gebildet wird. Der Befehl setzt oder löscht die Bits N, Z, V und C des Bedingungsregisters.

CMP erlaubt gleichermaßen den numerischen Vergleich von positiven Zahlen oder von Zahlen in Zweierkomplementdarstellung. Wie die Werte interpretiert werden sollen, muß man erst bei der Auswahl des bedingten Sprungbefehls beachten. Siehe 5.10.2.

Zum Vergleich der beiden Akkumulatoren dient Befehl CBA.

CBA : A - B (Compare Accumulators)

TST : M - 0 (Test)

Mithilfe von TST kann der Inhalt einer Speicherzelle oder eines Akkumulators auf Null oder Minus abgefragt werden. Für die Abfrage von Werten im Speicher ist der Befehl günstig, da er N und Z setzt ohne daß der Wert dazu geladen werden muß. Für Werte in Akkumulatoren ist der Befehl interessant, wenn durch zwischenzeitliche Operationen das Bedingungsregister nicht mehr die erforderliche Information enthält.

CPX : X_L - (M + 1) (Compare Index Register)

X_H - M

CPX vergleicht den Wert des 16 Bit Indexregisters mit dem zweier aufeinanderfolgender Speicherzellen indem intern die Differenz gebildet wird.

Vorsicht: CPX verändert zwar N und V. Deren Wert ist aber in diesem Fall nicht für die Abfrage in bedingten Sprüngen gedacht. Nach CPX sind nur die Abfragen auf gleich/ungleich (BEQ, BNE) sinnvoll.

Der Grund liegt darin, daß intern die Subtraktion für das höher- und niederwertige Byte unabhängig voneinander ausgeführt wird. Es findet also keine Subtraktion einer doppelt langen Zahl statt, wo ein Übertrag vom niederwertigen Byte berücksichtigt werden müßte.

5.10. Sprungbefehle

Die Sprungbefehle sind Grundlage für einige wesentliche Fähigkeiten eines Rechners, wie das wiederholte Ausführen eines Programmteils und das Abarbeiten verschiedener Programmzweige in Abhängigkeit von Entscheidungen.

Das automatische Fortschalten des Befehlszählers nach jeder Befehlsausführung steuert den linearen Programmablauf. Sprungbefehle laden den Befehlszähler mit einer neuen Adresse, wodurch im nächsten Befehlszyklus die Programmausführung ab der betreffenden Adresse fortgesetzt wird.

5.10.1. Unbedingte Sprungbefehle

Die zwei unbedingten Sprungbefehle, die sich nur im Adressierungsmodus unterscheiden, laden den Befehlszähler mit der Adresse des Sprungziels. Die Programmausführung setzt in jedem Falle an dieser Stelle fort.

BRA : (PC + 2) + R ⟶ PC (Branch Always)
mit relativer Adressierung (5.3.7)

JMP : ADR ⟶ PC (Jump)
mit extended oder indizierter Adressierung
(5.3.5 bzw. 5.3.6)

Als spezieller Sprungbefehl sei noch Befehl NOP (No Operation) erwähnt. Die einzige Wirkung des Leerbefehls NOP ist das Fortschalten des Befehlszählers um eins, was als "Sprung um eins" aufgefaßt werden kann.

5.10.2. Bedingte Sprungbefehle

Der Befehlssatz umfaßt 14 bedingte Sprungbefehle. Es sind 7 Paare komplementärer Befehle. Mit ihnen können ein oder mehrere Bits des Bedingungsregisters abgefragt werden. Ist die Bedingung erfüllt, so wird der Sprung ausgeführt, das heißt, in den Befehlszähler wird die Adresse des Sprungziels geladen. Andernfalls fährt das Programm mit dem nächsten Befehl fort.

Die relative Adressierung, die bei diesen Befehlen die einzig mögliche Adressierungsart ist, ist in 5.3.7 genauer behandelt. Bedingte Sprünge treten zusammen mit Vergleichsbefehlen (5.9) auf oder sie werten unmittelbar das Ergebnis einer Operation aus.

Zusammenstellung der bedingten Sprungbefehle, die genau ein Bedingungsbit abfragen:

Befehl	Sprung, falls	
BPL	N = 0	(Branch if Plus)
BMI	N = 1	(Branch if Minus)
BNE	Z = 0	(Branch if Not Equal)
BEQ	Z = 1	(Branch if Equal)
BVC	V = 0	(Branch if Overflow Clear)
BVS	V = 1	(Branch if Overflow Set)
BCC	C = 0	(Branch if Carry Clear)
BCS	C = 1	(Branch if Carry Set)

Besonders zu beachten sind die Unterschiede zwischen den Befehlen BHI (Branch if Higher) und BGT (Branch if Greater Than) oder BLS (Branch if Less Than or Same) und BLE (Branch if Less Than or Equal). Im ersten Fall werden die Operanden eines Vergleichsbefehls als positive Zahlen (0 bis 255), im zweiten als Zahlen im Zweierkomplement (-128 bis +127) interpretiert.

Beispiel:

		positiv	Zweierkompl.
ACCA	1 0 0 0 0 1 1 1	135	-121
ACCB	0 0 0 0 1 0 0 0	8	8

Der Vergleich CBA bildet intern die Differenz A - B, wobei N = 0, Z = 0, V = 1 und C = 0 gesetzt wird.

Die Abfrage BHI testet "C + Z = 0 ?". Die Bedingung ist erfüllt, d.h. 135 > 8.

Die Abfrage BGT testet "Z + (N ⊕ V) = 0 ?". Die Bedingung ist wegen V = 1 nicht erfüllt, d.h. -121 ist nicht größer als 8.

Zusammenstellung der bedingten Sprungbefehle zum Vergleich von positiven Zahlen:

Befehl	arithmetische Bedingung	Abfrage Bedingungscode
BCS	ACCX < M	C = 1
BCC	ACCX ≥ M	C = 0
BLS	ACCX ≤ M	C + Z = 1
BHI	ACCX > M	C + Z = 0
BEQ	ACCX = M	Z = 1
BNE	ACCX ≠ M	Z = 0

Zusammenstellung der bedingten Sprungbefehle zum Vergleich von Zahlen in Zweierkomplementdarstellung (Positive und negative Zahlen):

Befehl	arithmetische Bedingung	Abfrage Bedingungscode
BLT	ACCX < M	N ⊕ V = 1
BGE	ACCX ≥ M	N ⊕ V = 0
BLE	ACCX ≤ M	Z + (N ⊕ V) = 1
BGT	ACCX > M	Z + (N ⊕ V) = 0
BEQ	ACCX = M	Z = 1
BNE	ACCX ≠ M	Z = 0

5.10.3. Unterprogrammsprung und Rücksprung

Der Unterprogrammaufruf ist ein spezieller (unbedingter) Sprungbefehl, der die Rückkehradresse - das ist die Adresse des nächsten Befehls - sicherstellt. Der Mikroprozessor 6800 stellt zwei Befehle zur Verfügung, die sich nur in der Adressierungsart unterscheiden:

BSR (Branch to Subroutine)
mit relativer Adressierung

JSR (Jump to Subroutine)
mit indizierter und extended Adressierung.

In jedem Fall wird die Rückkehradresse, die zwei Bytes umfaßt, in den Stack gespeichert und das Stackregister um zwei vermindert. Anschließend wird die Adresse des Unterprogramms gemäß Adressierungsmodus ermittelt und in den Befehlszähler geladen, siehe Bild 5.9.

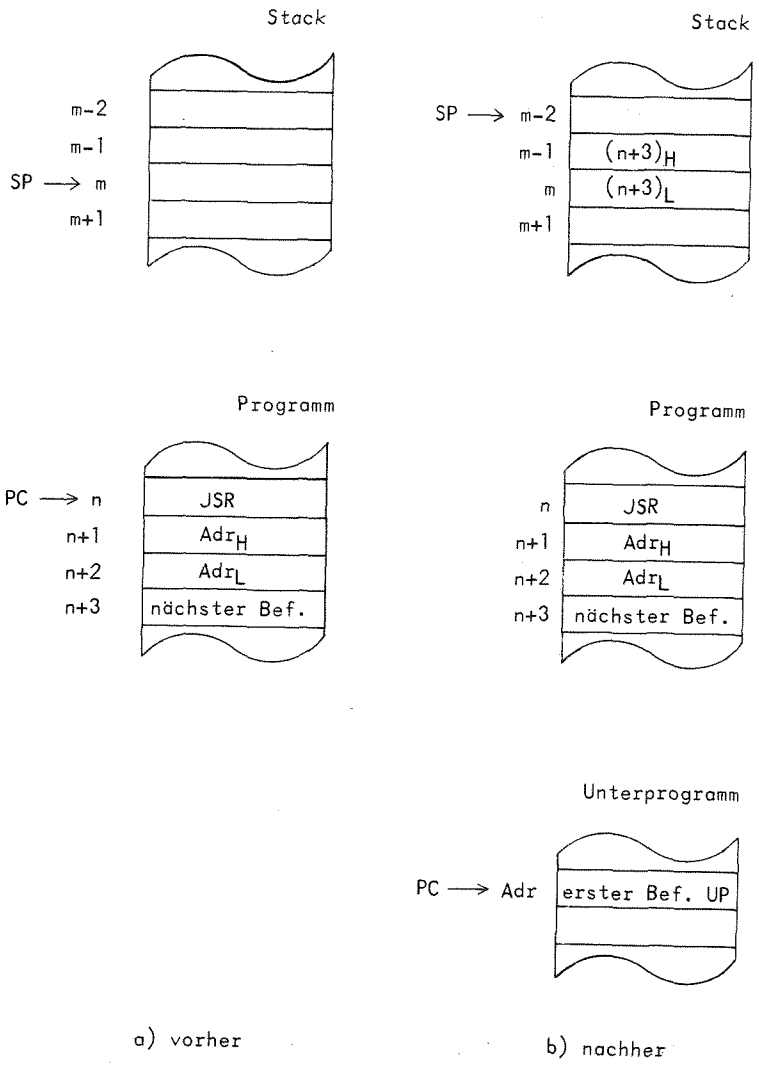


Bild 5.9 Wirkung von Befehl JSR (extended)

Der letzte Befehl eines Unterprogramms muß der Befehl RTS (Return from Subroutine) sein. Er lädt die im Stack gespeicherte Rückkehradresse in den Befehlszähler und erhöht das Stackregister entsprechend um zwei, siehe Bild 5.10.

Weitere Ausführungen zum Thema Unterprogramm stehen in Kapitel 6.6.

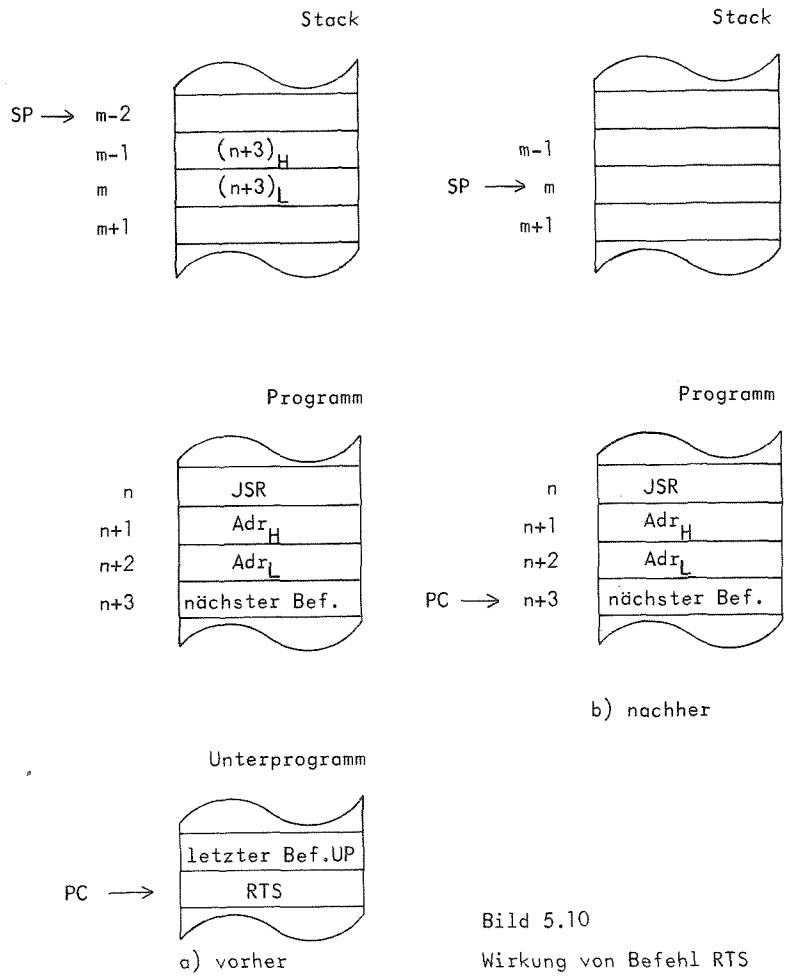


Bild 5.10
Wirkung von Befehl RTS

5.11. Sonderbefehle

5.11.1. Softwareinterrupt

Befehl SWI (Software Interrupt) veranlaßt das Abspeichern der Register auf dem Stack und die Ausführung einer Interrupt-Routine, deren Startadresse den Speicherzellen 3FA, 3FB (drittletzte Zeile Seite 1F) entnommen wird. Die Reihenfolge der Register auf dem Stack ist die gleiche wie beim Hardwareinterrupt, siehe Bild 3.5. Gleichzeitig setzt SWI die Interruptmaske I des Bedingungsregisters auf 1, so daß ein Hardwareinterrupt während der Interrupt-Routine nicht angenommen wird, es sei denn, I wird mit Befehl CLI gelöscht. Befehl SWI selbst wird vom Stand der Interruptmaske nicht beeinflußt, d. h. SWI wird auch bei I = 1 ausgeführt. Die Interrupt-Routine muß mit Befehl RTI enden, der die Register vom Stack zurück lädt.

Vorsicht: Die Ausführung von SWI im Einzelbefehlsmodus ist nicht möglich. Das gleichzeitige Auftreten von Software- und nichtmaskiertem Hardware-Interrupt (Einzelbefehl) wird vom M 6800 so behandelt, daß bei dieser Konstellation das Interruptprogramm für den maskierbaren Interrupt (Adressen 3F8, 3F9) gestartet wird.

5.11.2. Warten auf Interrupt

Befehl WAI (Wait for Interrupt) speichert die Register auf dem Stack ab (siehe 3.5) und versetzt den Prozessor anschliessend in einen Wartezustand. Ein Hardwareinterrupt beendet den Wartezustand und startet ein Interruptprogramm. WAI setzt man beispielsweise ein, um auf eine Benutzereingabe zu warten oder als Teststützpunkt.

Wichtig: Bei Einzelbefehlsausführung löst das hierzu eingesetzte nichtmaskierte Interruptsignal ebenfalls den Wartezustand auf. Da die zugehörige Routine nur aus den Befehlen NOP und RTI besteht, läuft man unmittelbar auf den nächsten Befehl nach WAI, was für einen WAI als Teststützpunkt auch wünschenswert ist. Soll hingegen ein Interruptprogramm ablaufen, so ist der Eintritt in dieses Programm nicht über Einzelbefehl möglich.

5.11.3. Rückkehr vom Interrupt

Jedes Interruptprogramm muß mit Befehl RTI (Return from Interrupt) enden. Er lädt die Register vom Stack zurück und erhöht das Stackregister entsprechend um 7. Die durch einen Interrupt auf 1 gesetzte Interruptmaske wird, falls sie vorher auf 0 war, hierdurch wieder auf Null gesetzt. Somit ist anschließend die Annahme eines Interruptsignals wieder erlaubt.

6. Programmiertechniken

Dieses Kapitel behandelt einige grundlegende Elemente der Programmierung und bringt gleichzeitig die Anwendung der Befehle des M 6800 in diesen verschiedenen Konstruktionen. Es zeigt, wie komplexere Funktionen aus dem gegebenen Befehlsvorrat aufgebaut werden können.

6.1. Programmablaufpläne

Ein Programmablaufplan (Flußdiagramm) ist eine graphische Beschreibungsform eines Programms, in der insbesondere komplexe Zusammenhänge, z. B. bei vielfachen Verzweigungen, anschaulich und übersichtlich darstellbar sind. Ein Programm kann durch mehrere Programmablaufpläne unterschiedlich detailliert beschrieben werden. In der Grobstruktur sind jeweils umfangreichere Teilaufgaben in einem Symbol zusammengefaßt. Ein solches Diagramm kann eine erste Übersicht über ein größeres Programm geben, seine Beschriftung ist verbal gehalten. Ein feiner strukturiertes Diagramm nimmt mehr Bezug auf die Programmierungsdetails, wobei einem Symbol meist einige Befehle im Programm entsprechen. Es empfiehlt sich, bei der Programmentwicklung und zur Dokumentation von Programmen mit Programmablaufplänen zu arbeiten. Im Buch finden sich zahlreiche Anwendungsbeispiele.

In der DIN 66001 sind die Sinnbilder festgelegt. Die für uns wichtigsten Symbole sind in Bild 6.1 aufgeführt.

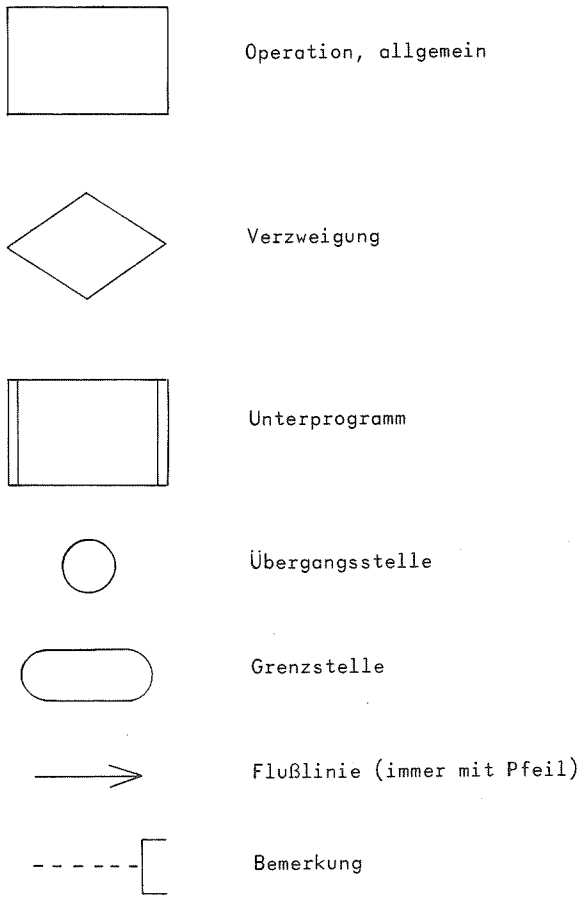


Bild 6.1 Sinnbilder für Programmablaufpläne

Für Verzweigungen verwenden wir ein von der Norm abweichendes Sinnbild: ein Rechteck mit spitzen Seitenbegrenzungen. Es ist für eine ausführliche Beschriftung günstiger als die Raute.

6.2. Bitmanipulationen

Für das Verändern einzelner Binärstellen eines Wortes sind die Operationen

- o Ausblenden
- o Einblenden
- o Zusammensetzen

grundlegend. Anwendungen finden sich in den Programmen 7.5 oder 7.6.4.

Beim Ausblenden sollen ein oder mehrere Bits gelöscht werden. Eine UND-Operation mit einer Maske mit 0 in den zu löschenden und 1 in den anderen Positionen, leistet das Gewünschte.

Beispiel: Alle Bits mit Ausnahme von b_4 und b_5 sind zu löschen.

ACCA	0 1 0 1 1 1 1 0	LDAA	Bitmuster
Maske	0 0 1 1 0 0 0 0	ANDA	Maske
ACCA	0 0 0 1 0 0 0 0		

Die Randbits können auch durch Shiftbefehle einfach verändert werden, beispielsweise Löschen von Bit 0 mit den Befehlen LSR und ASL.

Die Abfrage eines bestimmten Bits geschieht über das Ausblenden des restlichen Wortes und anschließender Abfrage auf Null. Eine Sonderstellung nimmt Bit 7 ein. Es ist unmittelbar über die Marke N des Bedingungsregisters mit Befehl BPL bzw. BMI abfragbar.

Eine ODER-Verknüpfung mit einer entsprechenden Maske führt das Setzen einzelner Bits aus.

Beispiel: Setze linkes Halbbyte auf 1.

ACCA	0 0 0 1 1 0 1 0	LDAA	Bitmuster
Maske	1 1 1 1 0 0 0 0	ORAA	Maske
ACCA	1 1 1 1 1 0 1 0		

Besteht die Aufgabe, das linke Halbbyte von M1 und das rechte Halbbyte von M2 zu einem neuen Wort M3 zusammenzusetzen, so löst dies das Programmstück:

```
LDAA    M1
ANDA I  F0          (Maske 1111 0000)
STAA    Hilfszelle
LDAA    M2
ANDA I  0F          (Maske 0000 1111)
ORAA    Hilfszelle
STAA    M3
```

6.3. Bedingungen

6.3.1. Einfache Bedingungen

Eine der wichtigsten Eigenschaften eines Rechners ist die Fähigkeit, während des Programmablaufs Entscheidungen auf Grund vorher errechneter Werte zu treffen. Das Programm wird in zwei verschiedenen Zweigen weitergeführt, je nachdem, ob eine Bedingung erfüllt ist oder nicht. Die Frage muß eindeutig mit ja oder nein beantwortbar sein. Das Symbol für Abfragen hat einen Eingang und zwei Ausgänge, die mit ja/nein unterschieden werden.

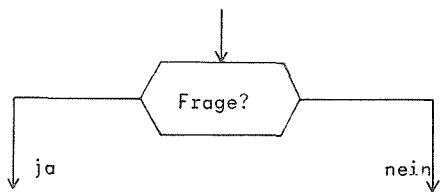
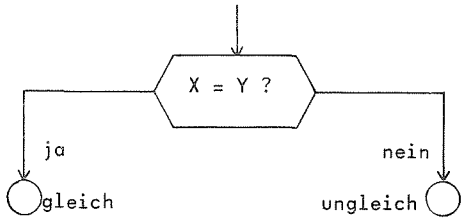


Bild 6.2 einfache Bedingung

Für den Vergleich von Zahlen kommen die Relationsoperationen $<, \leq, >, \geq, =, \neq$ (kleiner, kleiner oder gleich, größer, größer oder gleich, gleich, ungleich) in Betracht.

Andere Abfragen beziehen sich auf den Wert einzelner Binärstellen, wie z. B. die Bits des Bedingungsregisters. Den arithmetischen Vergleich realisieren Vergleichsbefehle zusammen mit bedingten Sprungbefehlen.

Beispiel: Abfrage von X und Y auf Gleichheit.




```
        LDAA   X
        CMPA   Y
        BNE    ungleich
gleich  ---
```


6.3.2. Zusammengesetzte Bedingungen

Eine zusammengesetzte Bedingung besteht aus Bedingungen, die durch logische Operatoren verbunden sind. Es handelt sich um Entscheidungen der Art "wenn sowohl die eine als auch die andere Bedingung erfüllt ist, dann" oder "wenn entweder die eine oder die andere oder beide Bedingungen erfüllt sind, dann". Das Element "zusammengesetzte Bedingung" hat ebenfalls einen Eingang und zwei Ausgänge. Es kann in weiteren Zusammenschaltungen für komplexere Abfragen wie eine einfache Bedingung eingebaut werden.

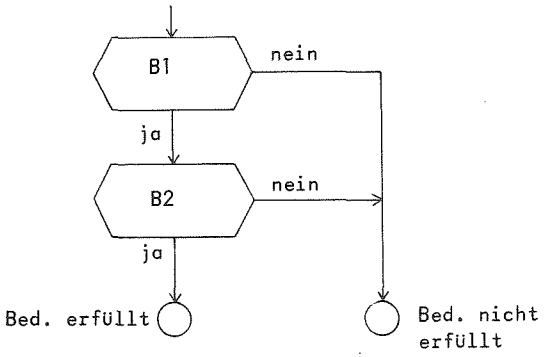


Bild 6.3 UND-Verknüpfung von zwei Bedingungen

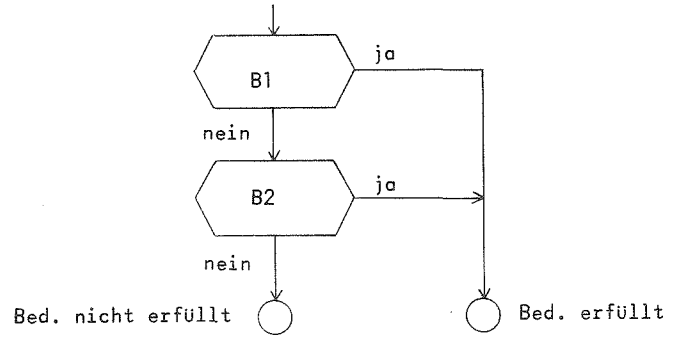


Bild 6.4 ODER-Verknüpfung von zwei Bedingungen

Ist bei UND die erste Antwort "nein", so liegt das Ergebnis bereits fest. Das gleiche gilt für ODER, falls die erste Frage mit "ja" beantwortet wird. Die zweite Bedingung braucht in diesen Fällen also nicht mehr untersucht werden.

Beim exklusiven Oder (Antivalenz) ist die Gesamtbedingung erfüllt, wenn die eine oder die andere Bedingung, nicht aber beide gleichzeitig erfüllt sind.

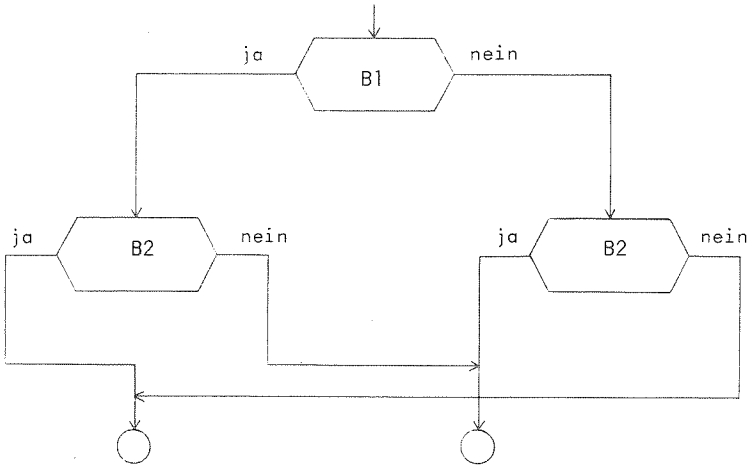


Bild 6.5 Exklusives Oder von zwei Bedingungen

Zuletzt geben wir noch ein Diagramm für die Äquivalenz-Verknüpfung von zwei Bedingungen an. Die Gesamtbedingung ist hier erfüllt, wenn beide Fragen mit ja oder beide mit nein beantwortet werden.

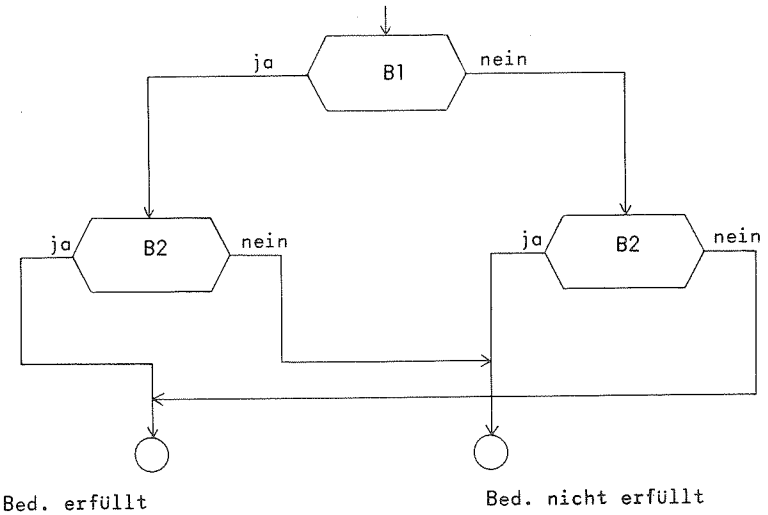


Bild 6.6 Äquivalenz von zwei Bedingungen

Beispiel: Programmierung der Abfrage X = 0 UND Y = 0.

```
                TST     X
                BNE     nicht erfüllt
                TST     Y
                BNE     nicht erfüllt
erfüllt         ___
```

6.4. Verteiler

Verteiler sind komplexere Programmverzweigungen, d. h. Verzweigungen nach mehr als zwei Stellen. Ein Verteiler ist immer mit einer Größe verbunden, die n verschiedene Werte annehmen kann. Wird der Verteiler angelaufen, so soll auf Grund des aktuellen Werts dieser Auswahlgröße eines von n Programmstücken angesprungen werden. Ohne auf programmierungstechnische Details einzugehen, stellt man einen Verteiler im Diagramm gemäß Bild 6.7 dar. Im allgemeinen ist für den Fall, daß die Auswahlgröße einen unzulässigen Wert enthält, ein Fehlerausgang vorzusehen.

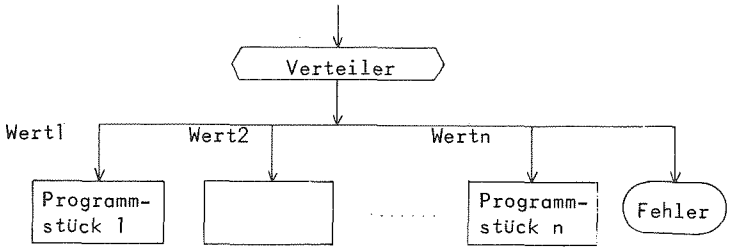


Bild 6.7 Struktur eines Verteilers

Bei beliebigen Werten der Auswahlgröße ist die Auswahl des zugehörigen Programmzweiges durch eine Folge von Abfragen zu realisieren. Bei der Anordnung in Bild 6.8 stehen alle Abfragen unmittelbar nacheinander. Dem Verteiler entspricht eine zusammenhängende Befehlsfolge, die Programmteile können sich in geordneter Reihenfolge anschließen. Diese Lösung ist der im Bild 6.9 dargestellten zumindest bei längeren Programmstücken vorzuziehen. Dort wird das Programmelement "Verteiler" über

weite Teile des Programms auseinandergerissen, worunter die Übersichtlichkeit leidet.

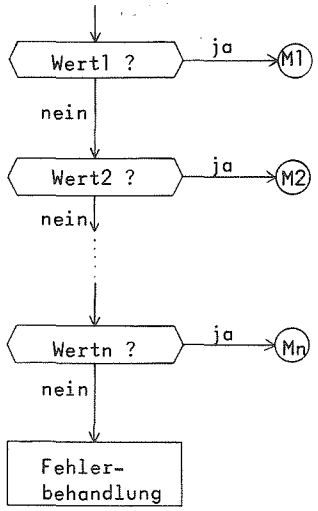


Bild 6.8 Verteiler als zusammenhängende Folge von Abfragen realisiert

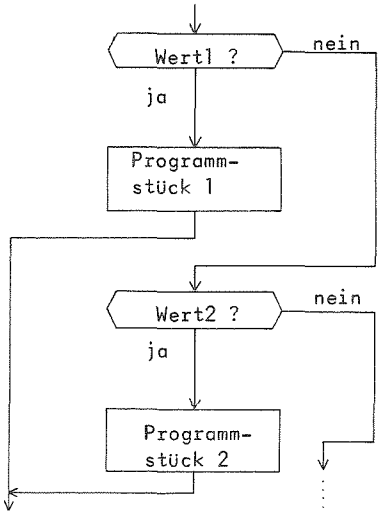


Bild 6.9 Verteiler ist über das Programm verstreut

Eine besonders günstige Lösung gibt es bei der Programmierung des M 6800, falls die Auswahlgröße Werte in einem zusammenhängenden Zahlenbereich in Zweierschritten annimmt, z. B. 0, 2, 4, 6, ... In diesem Fall ist der Verteiler über eine Adressentabelle realisierbar. In der Tabelle stehen der Reihe nach die Anfangsadressen der betreffenden Programmstücke (2 Bytes). Die Auswahlgröße dient als Index in dieser Tabelle, die so ermittelte Programmadresse, die ins Indexregister geladen wird, wiederum als Index für den nachfolgenden Sprungbefehl.

```
                LDX      Auswahlgröße
                LDX  X   Adressentabelle
                JMP  X     0
Adressentabelle Adresse  M0
                Adresse  M2
                -
                -
        M0      Programmstück 0
                -
```

Bei dieser Lösung muß die Adressentabelle im direkt adressierbaren Speicherbereich liegen. Ferner ist hier der Fehlerfall nicht berücksichtigt.

Eine weitere Möglichkeit bietet das Suchen des Werts in einer Tabelle, wobei ein Eintrag Wert und zugehörige Programmadresse umfaßt. Diese Lösung ist bei einer großen Zahl von Verzweigungen günstig.

Beispiel: Wir nehmen an, daß die Auswahlgröße ein Byte und die zugeordnete Adresse zwei Bytes belegt. Der aktuelle Wert wird in einer Schleife (siehe nächster Abschnitt) mit den Tabellenwerten verglichen. Bei positivem Vergleichsergebnis wird die Adresse ins Indexregister geladen und das betreffende Programmstück mit indizierter Adressierung angesprungen.

Tabellenanfang

Wert1	Adresse Programmstück 1
⋮	
Wertn	Adresse Programmstück n

Tabellenende

```
                LDAA       Auswahlgröße
                LDX        Adresse Tabellenanfang
Vergleiche      CMPA  X    0
                BEQ        Gefunden
                INX
                INX
                INX
                CPX        Adresse Tabellenende
                BNE        Vergleiche
Fehlerfall      ---
Gefunden        LDX   X    1
                JMP   X    0
```

6.5. Schleifen

6.5.1. Allgemeines

Ein wesentliches Programmelement sind Schleifen. Schleifen dienen zur wiederholten Ausführung eines Programmstücks. Vom Typ her unterscheidet man Zählschleifen und Bedingungsschleifen. Bei Zählschleifen ist die Anzahl der Schleifendurchläufe zu Beginn bekannt, bei Bedingungsschleifen nicht.

Allgemein baut sich eine Schleife aus vier Komponenten auf:

- o Initialisierung Schleifenkriterium
- o Bearbeitung (Schleifenrumpf)
- o Verändern Schleifenkriterium
- o Endetest.

Ist bekannt, daß es immer mindestens einen Schleifendurchlauf gibt, so kommt man zu folgendem Bild:

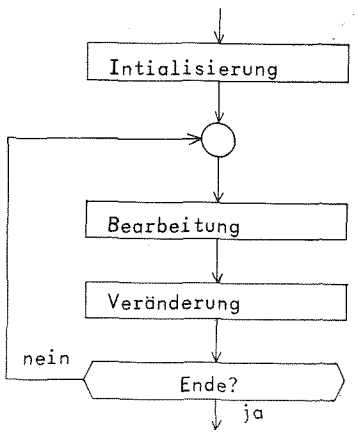


Bild 6.10 Struktur einer Schleife mit mindestens einem Durchlauf

Falls die Zahl der Durchläufe auch Null sein kann, muß der Endetest bereits vor der ersten Bearbeitung ausgeführt werden. Es gibt zwei Möglichkeiten für die Anordnung des Tests:

a) am Schleifende.
   Nach der Initialisierung folgt ein unbedingter Sprung auf die Endabfrage.

b) am Schleifenanfang.
   Am Schleifenende muß durch einen unbedingten Sprung zur Endabfrage zurückgekehrt werden.

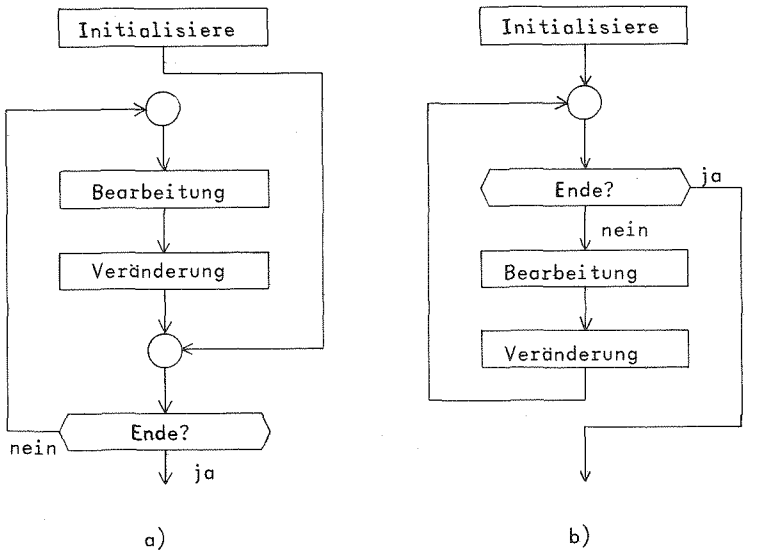


Bild 6.11 Struktur einer Schleife mit möglicherweise Null Durchläufen

6.5.2. Zählschleifen

Bei den Zählschleifen gibt es die beiden Varianten der Vorwärts- und der Rückwärtszählung. Wenn von der Problemstellung her beide Varianten in Betracht kommen, führt die Rückwärtszählung bei einer Endabfrage auf Null zu einer eleganteren Lösung als die Vorwärtszählung. Man kann den Vergleichsbefehl sparen.

6.5.2.1. Verzögerungsschleife

Die einfachste Form einer Zählschleife - die Komponente "Bearbeitung" entfällt hier ganz - setzen wir in den Beispielprogrammen (z. B. in 7.1.) als Verzögerungsschleife ein. Bei normalem Programmablauf wären Veränderungen auf dem Bildschirm wegen der hohen Verarbeitungsgeschwindigkeit kaum zu verfolgen. Eine einfache Zählschleife "beschäftigt" den Prozessor eine gewisse Zeit bis die eigentliche Verarbeitung weiterschreitet. Durch Variieren des Zählwertes sind verschiedene Verzögerungszeiten einstellbar. Für diesen Zweck reicht 1 Byte als Zähler nicht aus. Günstig ist die Rückwärtszählung im Indexregister.

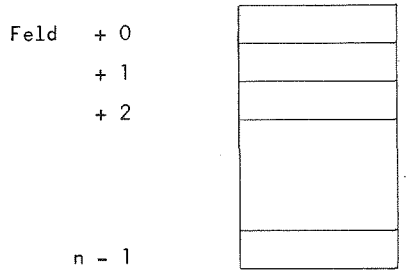
```
                LDX     Verzögerungsfaktor
Schleife        DEX
                BNE     Schleife
```

6.5.2.2 Zählschleifen bei Indizierung

Für die gleichartige Bearbeitung von zusammenhängenden Speicherbereichen bieten sich ebenfalls Zählschleifen an.

Meist ist hier der Zähler gleichzeitig Index zur fortlaufenden Adressierung der Speicherzellen des Bereichs.

Beispiel: Löschen des Speicherbereichs "Feld" von n aufeinander folgenden Zellen (n ≧ 1).



Lösung 1: Vorwärtszählen

```
                LDX     0
Schleife        CLR  X  Feld
                INX
                CPX     N
                BNE     Schleife
```

Lösung 2: Rückwärtszählen

```
                LDX     N
Schleife        CLR  X  Feld - 1
                DEX
                BNE     Schleife
```

Man beachte, daß hier die Adresse Feld - 1 stehen muß!
Diese beiden Lösungen sind nur richtig, wenn Feld im direkt adressierbaren Speicherbereich (Adressen 0 - 255) liegt.

Lösung 3: Beliebiger Speicherbereich, Vorwärtszählung

```
            LDX          Anfangsadresse Feld
Schleife    CLR    X     0
            INX
            CPX          Endadresse + 1
            BNE          Schleife
```

6.5.3. Bedingungsschleifen

Bei Bedingungsschleifen ist die Zahl der Durchläufe von vorneherein nicht bekannt. Sie wird durch die zu verarbeitenden Daten bestimmt. Ein Beispiel ist das Multiplikationsprogramm in 5.7.7.

Beispiel: Eine Tabelle Feld 1 variabler Länge soll in einen Bereich Feld2 umgespeichert werden. Das Tabellenende ist durch ein Wort mit Inhalt 0 gekennzeichnet, das noch mitzutransportieren ist.

Lösung 1:

```
            LDX          0
Schleife    LDAA   X     Feld1
            STAA   X     Feld2
            INX
            TSTA
            BNE          Schleife
```


Die Lösung gilt nur für direkt adressierbare Bereiche. Man beachte, daß ein Testbefehl nötig ist, da durch INX das Bedingungsregister verändert wurde. Lösung 2 umgeht dies durch Initialisieren des Indexregisters mit -1 (FFFF) und Erhöhen des Index vor dem Transport.

Lösung 2:

```
            LDX          -1
Schleife    INX
            LDAA   X     Feld1
            STAA   X     Feld2
            BNE          Schleife
```

6.6. Unterprogramme

6.6.1. Allgemeines

Es gibt zwei Gründe, eine bestimmte Befehlsfolge als Unterprogramm zu organisieren.

a) Die gleiche oder ähnliche Befehlsfolge kommt in einem Programm an mehreren Stellen vor. Das wiederholte Einfügen dieses (möglicherweise langen) Programmstücks umgeht die Formulierung als Unterprogramm.

b) Selbst wenn ein Programmstück, das eine in sich abgeschlossene Teilaufgabe erledigt, nur einmal benötigt wird, kann seine Organisation als Unterprogramm zweckmäßig sein. Dieses Vorgehen gliedert das Hauptprogramm und macht es kürzer und dadurch übersichtlicher.

Auch die Lösung von Standardaufgaben, die in verschiedenen Programmen einsetzbar sein sollen, stellt man in Form eines Unterprogramms zur Verfügung. Bei der Entwicklung eines Programms ist immer zu überlegen, ob es Teile gibt, die an verschiedenen Stellen in gleicher oder ähnlicher Weise vorkommen.

Ein Unterprogramm kann von einer beliebigen Stelle des Hauptprogramms aus aufgerufen werden. Ein spezieller Sprungbefehl (JSR, BSR) stellt die Rückkehradresse sicher, so daß nach Abarbeitung des Unterprogramms an der Aufrufstelle fortgefahren werden kann (siehe Bild 6.12).

Die Aufrufmöglichkeit von Unterprogrammen bleibt nicht auf das Hauptprogramm beschränkt. Ein Unterprogramm kann ein weiteres UP aufrufen und so fort. Man spricht dann von geschachtelten Unterprogrammen. Gäbe es nur eine feste Zelle zum Abspeichern der Rücksprungadresse, so wäre dies nicht möglich, da hier eine früher reservierte Adresse überschrieben würde. Das Konzept des Kellerspeichers (Stack) macht eine allgemeine Aufrufstruktur von UP möglich (siehe Bild 6.13).

Beispiel: Im Hauptprogramm werde an zwei Stellen das Unterprogramm UP aufgerufen. Die durchgezogene Linie gibt die zeitliche Abfolge der Befehle wieder.

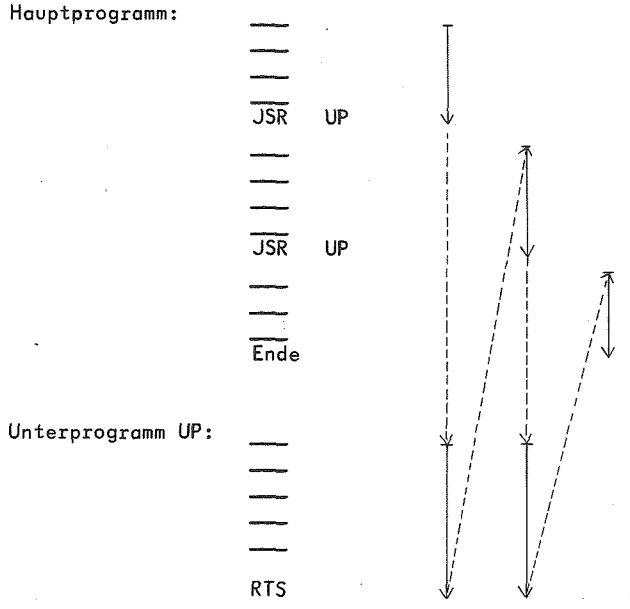


Bild 6.12 Programmablauf mit Unterprogramm

Bild 6.13 zeigt einen geschachtelten Unterprogrammaufruf von UP1 und UP2 und den Inhalt des Stacks in den verschiedenen Phasen.

1. Der Aufruf von UP1 im Hauptprogramm speichert die Adresse des Folgebefehls im Stack und vermindert das Stackregister.

2. Der Aufruf von UP2 im Unterprogramm UP1 speichert die Adresse des betreffenden Folgebefehls in gleicher Weise im Stack ab.

3. Der Rücksprung in UP2 entnimmt die oberste Adresse - die Fortsetzungsadresse in UP1 - und setzt den Programmablauf an dieser Stelle fort. Gleichzeitig erhöht sich das Stackregister.

4. Der Rücksprung in UP1 setzt die oberste Adresse - die Fortsetzung im Hauptprogramm - in den Befehlszähler und erhöht SP entsprechend.

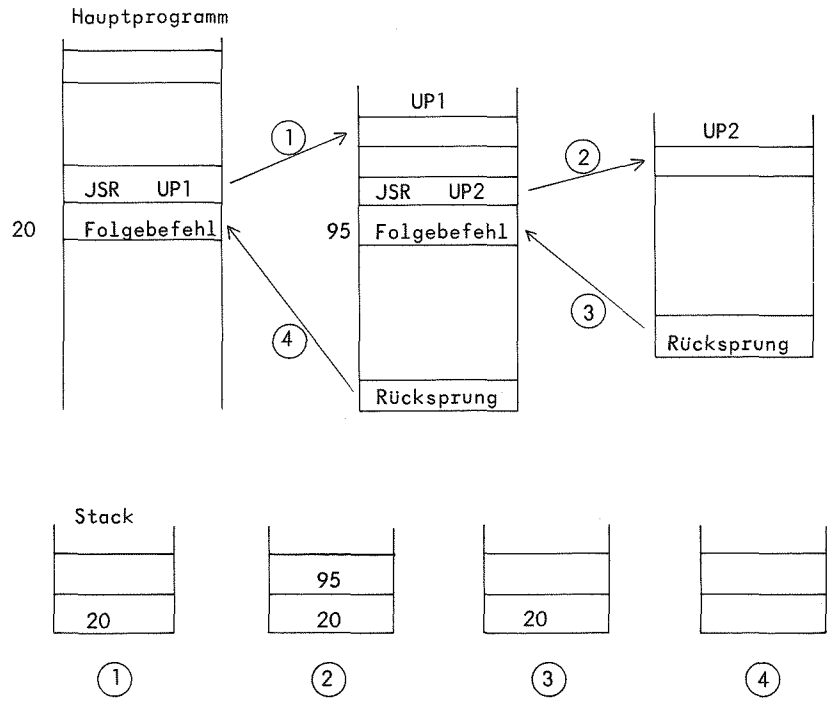


Bild 6.13 Beispiel für Speicherung der Rücksprungadressen bei geschachtelten Unterprogrammen

6.6.2. Parameterübergabe

Für Befehlsfolgen, die an verschiedenen Stellen in ähnlicher Form benötigt werden, beispielsweise bei gleichartiger Verarbeitung verschiedener Operanden, setzt man Unterprogramme mit Parametern ein. Parameter sind Größen, die sich von Aufruf zu Aufruf ändern können.

Wir diskutieren verschiedene Möglichkeiten der Parameterübergabe.

1. Parameter liegen in festen Speicherzellen.
   Das Hauptprogramm und das Unterprogramm kennen die absoluten Adressen. Das Hauptprogramm lädt diese Zellen vor dem Unterprogrammaufruf, das UP greift darauf zu und kann auch Ergebnisse dort ablegen. Dies ist die einfachste Möglichkeit der Parameterübergabe. Sie schränkt jedoch die allgemeine Verwendung eines Unterprogrammes ein und ist auch für rekursive Unterprogramme nicht geeignet.

2. Parameterübergabe in Registern.
   Die Unterprogramme sind allgemeiner einsetzbar, allerdings ist die Zahl der Parameter auf die Zahl der Register beschränkt. Diese Schwierigkeit umgeht die

3. Adressenübergabe in Registern.
   Die Anfangsadresse eines Speicherbereichs, in dem die Parameter in vereinbarter Reihenfolge gespeichert sind, wird zweckmäßigerweise im Indexregister - übergeben.
   Das UP greift mit indizierter Adressierung auf die Parameter zu.

4. Parameterübergabe im Stack
   Sie ist für rekursive Unterprogramme, für die eine Übergabe in Registern nicht möglich ist, die geeignete Form.
   Das UP lädt den Inhalt des Stackregisters in das Indexregister (TSX) und kann ebenfalls indiziert auf die Parameter zugreifen.
   Der Vollständigkeit halber erwähnen wir noch die

5. Parameterübergabe in Zellen nach dem Unterprogrammaufruf.
   Sie ist bei unserem Befehlssatz in der Programmierung relativ aufwendig, bietet keine weiteren Vorteile und führt außerdem zu einer Vermengung von Programm- und Datenbereich.

Programm 7.3 (Siebensegmentanzeige) ist ein Beispiel für die Parameterübergabe in Registern. In Programm 7.5 arbeiten wir mit der Parameterübergabe in festen Zellen.

6.6.3. Rekursive Unterprogramme

Unterprogramme, die sich unter gewissen Bedingungen selbst aufrufen, heißen rekursiv. Für einige Problemstellungen liefern rekursive Unterprogramme elegante Lösungen.

An ein rekursives Programm werden besondere Forderungen gestellt: es muß wiedereintrittsfähig (reentrant) sein. Kann ein UP, bevor es seine Aufgabe beendet hat, nochmals aufgerufen werden, um die gleiche Aufgabe - mit veränderten Parametern - auszuführen, so dürfen Daten oder Zwischenergebnisse des ersten Aufrufs durch spätere Aufrufe nicht überschrieben werden. Die Forderung ist also, daß alle veränderlichen Speicherzellen für jeden Aufruf getrennt bereitzustellen sind.

Das Hilfsmittel hierzu ist wieder der Stack. Auf dem Stack können Daten über eine Aufruffolge hinweg gerettet werden, siehe auch das Beispiel.

Beispiel: Berechnung der Fakultät.

$$n! = n.(n-1) \cdot \ldots \cdot 2 \cdot 1$$

Mit $0! = 1$

$$n! = n \cdot (n-1)!$$

ist eine rekursive Definition der Fakultät gegeben.

Diese Formeln sollen in ein Unterprogramm umgesetzt werden. Wir formulieren ein rekursives Unterprogramm FAK mit einem Eingangsparameter n und einem Ergebnis E, dem Wert von n!.

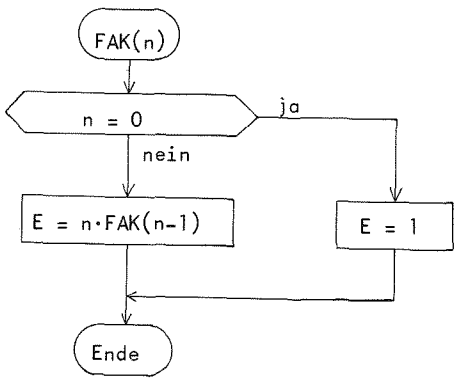
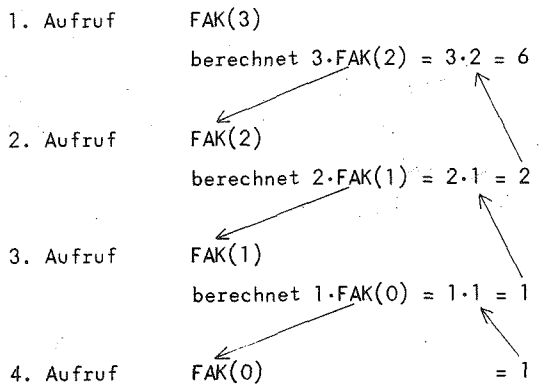


Bild 6.14 Unterprogramm zur Fakultätsberechnung

Der Aufruf von FAK(3) läuft folgendermaßen ab:



Das nachfolgende Programm besteht aus den drei Teilen

- o Hauptprogramm mit Aufruf von FAK (n)
- o rekursives Unterprogramm FAK
- o Multiplikationsunterprogramm MULT

Da wir mit 8 Bit Zahlen arbeiten, berechnet das Programm nur die Werte bis 5! richtig. Dies genügt aber, um das Prinzip der rekursiven Prozedur zu demonstrieren. Man beachte ferner, daß der jeweilige Wert von n auf dem Stack zwischengespeichert wird. Feste Speicherzellen oder Register kommen dafür nicht in Frage.

Der Stack und die Speicherplätze für n und FAK(n) werden auf die Seite 3 gelegt. Damit ist der Ablauf für verschiedene Werte gut beobachtbar.

Belegte Seiten: 0 - 3

Anzeigeseite: 3

Adresse n: 61, erste Zeile rechtes Byte auf Seite 3

Adresse FAK(n): 63, zweite Zeile rechtes Byte auf Seite 3

Startadresse: 0

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Fakultätsberechnung, Hauptprogramm

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	7F	8E	Stack liegt am Ende von Seite 3
1	1					00	
2	2					7F	
3	3		LDAA	D	N	96	Parameter n für UP FAK laden
4	4					61	
5	5		JSR	E	FAK	BD	
6	6					00	
7	7					20	
8	8		STAA	D	Ergebnis	97	Ergebnis FAK(n) abspeichern
9	9					63	
10	A	Stop	BRA	R	Stop	20	
11	B					FE	
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Fakultätsberechnung, UP FAK

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	FAK	TSTA			4D	n = 0 ?
1	1		BNE	R	Rechne	26	
2	2					03	
3	3		LDAA	I	1	86	ja, FAK(n) = 1
4	4					01	
5	5		RTS			39	Ende
6	6	Rechne	PSHA			36	Zwischenspeicherung von n im Stack
7	7		DECA			4A	n-1
8	8		JSR		FAK	BD	rekursiver Aufruf !
9	9					00	berechne FAK (n-1), Ergebnis steht
10	A					20	in ACCA
11	B		PULB			33	hole n aus Stack
12	C		LDX	I	Produkt	CE	Parameter für UP MULT
13	D					00	X = Adresse Ergebnis
14	E					36	ACCA = (n-1)!
15	F		JSR		MULT	BD	ACCB = n
16	10					00	
17	11					38	
18	12		LDAA	D	Produkt+1	96	nur niederwertiger Anteil des Produkts
19	13					37	wird weiterverarbeitet
20	14		RTS			39	
21	15						
22	16	Produkt					
23	17						
24	18	MULT	STAB	D	MULT1	D7	Multiplikations - UP
25	19					5D	X = Adresse für 16-Bit - Ergebnis
26	1A		CLR	E	MULT2	7F	ACCA = Multiplikator
27	1B					00	ACCB = Multiplikand
28	1C					5C	
29	1D		CLR	X	0	6F	weitere Erläuterungen siehe S. 7.7.
30	1E					00	
31	1F		CLR	X	1	6F	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: Fakultätsberechnung

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0					01	
1	1		BRA	R	M2	20	
2	2					06	
3	3	M1	ASL	E	MULT1	78	
4	4					00	
5	5					5D	
6	6		ROL	E	MULT2	79	
7	7					00	
8	8					5C	
9	9	M2	LSRA			44	
10	A		BCC	R	M3	24	
11	B					0D	
12	C		LDAB	D	MULT1	D6	
13	D					5D	
14	E		ADDB	X	1	EB	
15	F					01	
16	10		STAB	X	1	E7	
17	11					01	
18	12		LDAB	D	MULT2	D6	
19	13					5C	
20	14		ADCB	X	0	E9	
21	15					00	
22	16		STAB	X	0	E7	
23	17					00	
24	18		TSTA			4D	
25	19	M3	BNE	R	M1	26	
26	1A					E8	
27	1B		RTS			39	
28	1C	MULT2					
29	1D	MULT1					
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

6.7. Interruptverarbeitung

Dieser Abschnitt soll an einem Beispiel die Funktion der Interruptmaske demonstrieren. Die Interruptmaske, die beim Start eines Interruptprogramms automatisch gesetzt wird, sperrt die Annahme eines weiteren Interrupts. Wird sie durch Befehl CLI im Interruptprogramm gelöscht, so ist die Annahme eines weiteren Signals erlaubt. Das betreffende Interruptprogramm muß dann wiedereintrittsfähig sein. Dies ist der Fall, wenn der aktuelle Zustand des Programms durch die Registerinhalte vollständig beschrieben ist. Ein Interrupt speichert die Register auf dem Stack ab (im Unterschied zum Unterprogrammsprung, der nur die Rücksprungadresse abspeichert), wodurch der Programmzustand bis zur späteren Fortsetzung sichergestellt ist.

An einem einfachen Beispiel sind verschiedene Varianten der Interruptverarbeitung nachvollziehbar. Das Hauptprogramm besteht nur aus einigen Initialisierungsbefehlen und einem dynamischen Stop. Ein "langes" Interruptprogramm mit einer Zählschleife erlaubt das mehrfache Schalten der Interrupttaste während der Interruptverarbeitung. Probieren Sie folgende Fälle aus:

1) Befehl NOP in Adresse 8 sperrt die Annahme eines weiteren Interrupts.
2) Befehl CLI in Adresse 8 erlaubt weitere Interrupts. Mit jedem Interrupt füllt sich der Stack, der am Ende der Anzeigeseite liegt. Nach Abschluß eines Interruptprogramms ist die Fortsetzung mit dem alten Zählerstand des unterbrochenen Programms beobachtbar.
3) Einlegen von Halt, Interrupt und anschließender Fortstart zeigt, daß ein Interruptsignal im Haltzustand intern gespeichert wird und nach Aufheben des Halts die Interruptverarbeitung beginnt.

Belegte Seiten: 0 - 1 Programm siehe Seite 138.
Anzeigeseite: 1
Startadresse: 0
Interruptadresse: 8

6.8. Testpraktiken

Das Testen ist eine wichtige Phase in der Programmentwicklung. Der Programmtest soll das fehlerfreie Arbeiten des Programms nachweisen. Da Programme beim ersten Testlauf im allgemeinen nicht fehlerfrei sind, ist ein systematisches Vorgehen bei der Fehlersuche sehr wichtig.

Schon beim Programmentwurf wird über die leichte Testbarkeit und damit über den Zeitaufwand für die Testphase mitentschieden. Anhaltspunkte für einen testfreundlichen Programmaufbau sind:

- o Übersichtliche Gliederung des Programms durch Zusammenfassung von Teilaufgaben in Unterprogrammen und sinnvolle Namensgebung,
- o Vorsehen von Teststützpunkten durch Leerbefehle (NOP) vor bzw. nach wichtigen Abschnitten,
- o Zusätzliches Abspeichern von Zwischenergebnissen,
- o Freihalten einiger Bytes pro Seite für spätere Korrekturen.

Außerdem ist die Programmdokumentation stets auf dem neuesten Stand zu halten. Zur Dokumentation gehören Aufgabenstellung, Programmablaufpläne, kommentierte Befehlsliste, Speicherbelegungsplan und Bedienungsanleitung.

Es empfiehlt sich, vor dem ersten Start das Programm per Hand genau durchzuspielen. Dann ist das Programm mit Testdaten zu starten, deren Ergebnis bekannt ist. Nach dem Programmstart können drei Situationen eintreten: es erscheint

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Demonstration Interrupt - Varianten

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Haupt pr.	LDS	I	3F	8E	Stack liegt am Ende von Seite 1
1	1					00	
2	2					3F	
3	3		CLI			0E	Lösche Interruptmaske,
4	4	Stop	BRA	R	Stop	20	dynamischer Stop
5	5					FE	
6	6						
7	7						
8	8	Interrupt	NOP			02	// CLI - 0E erlaubt weitere Interrupts
9	9		LDX	I		CE	Zähler laden
10	A					FF	
11	B					FF	
12	C	Zähle	STX	D	20	DF	Zähler in erster Zeile Seite 1 abspeichern
13	D					20	
14	E		DEX			09	-1
15	F		BNE	R	Zähle	26	≠ 0, weiterzählen
16	10					FB	
17	11		RTI			3B	Ende Interruptprogramm
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

- das erwartete Ergebnis,
- ein falsches Ergebnis,
- nichts.

Im ersten Fall ist das Programm mit unterschiedlichen Daten wiederholt zu starten, um es in allen Verzweigungen auszutesten. In den beiden letzteren Fällen ist der Fehler zu suchen. Eventuell sind durch variieren der Eingangsdaten weitere Anhaltspunkte über die Fehlerursache zu gewinnen. Der nächste Abschnitt zählt einige häufige Programmierfehler auf. In der Auswirkung besonders unangenehm sind Fehler, die das Programm zerstören. Um dies festzustellen sollte im Fehlerfalle das Programm im Speicher nochmals kontrolliert werden.

Der Einzelbefehlsablauf mit Registeranzeige (siehe 3.5) erlaubt das lückenlose Verfolgen des Programmablaufs. Einzelbefehlsausführung ist ab Programmanfang oder ab einem Teststützpunkt mit Befehl WAI möglich. Auf diese Weise ist das stückweise Testen kritischer Programmteile einfach realisierbar.

Die Fehlerkorrektur durch Abändern einzelner Befehle ist unproblematisch. Schwieriger ist das Einfügen von Befehlen. Wir empfehlen, anfangs für Korrekturzwecke pro Seite einige Bytes freizuhalten. Ein logisch richtiges Programm kann ohne Schwierigkeiten in eine kompakte Form gebracht werden. Ist kein Platz für einzufügende Befehle vorhanden, so sollte das betreffende Programmstück neu programmiert und eingegeben werden.

Der Vollständigkeit halber sei noch die sogenannte Rucksackmethode erwähnt, obwohl wir sie wegen ihrer Unübersichtlichkeit nicht empfehlen. An der einzufügenden Stelle wird ein Befehl durch einen Sprung ans Programmende ersetzt. Den überschriebenen Befehl, die fehlenden Befehle und einen Rücksprung fügt man an das Ende des Programms an.

6.9. Hinweise auf häufige Fehler

1. Fehlerhafte Befehlslänge, Adreßteil zu kurz oder zu lang.

2. Relative Sprungweiten sind oft zu weit oder zu kurz.

3. Fehlerhafte Umsetzung von Externcode in den Interncode.

4. Fehlerhafte Start- oder Interruptadresse.

5. Einzelbefehlsausführung ohne vorgeschriebene Befehlsfolge für Interruptprogramm.

6. Stackregister nicht besetzt.

7. Bedingungsregister enthält nicht mehr die in der Abfrage gewünschte Information. Die Abfrage des Bedingungscodes sollte möglichst unmittelbar nach dem betreffenden Befehl erfolgen. Andernfalls ist genau zu prüfen, ob die dazwischenliegenden Befehle keine Änderung herbeiführen.

8. Fehlende Rücksprungbefehle bei Unterprogrammen oder Interruptprogrammen. Wird ein Unterprogramm nicht vorschriftsmäßig über den RTS, eine Interruptschleife nicht über den RTI verlassen, stimmt die Stackbuchhaltung nicht mehr. Das Programm läuft in andere Programm- oder Datenbereiche.

9. Bei zu hell eingestelltem Fernsehbild kann versehentliches Setzen von Nachbarfeldern zu Fehlern führen. Vor dem Programmstart ist das Programm nochmals zu kontrollieren.

10. Logische Fehler im Lösungsverfahren oder im Programmentwurf.

7. Beispielprogramme

Dieses Kapitel enthält eine Sammlung vollständiger Programme für das TV-Computersystem 6800. Es wurde versucht, in relativ kurzen Programmen die vielfältigen Möglichkeiten des Systems aufzuzeigen und Anregungen für eigene Arbeiten zu geben.

Die Programme können zunächst einfach eingegeben und ausprobiert werden, sie können aber auch Befehl für Befehl studiert und nachvollzogen werden. Dabei wird die Anwendung vieler behandelter Programmierungsdetails nochmals in einem größeren Zusammenhang vorgeführt.

Schreiben mehrere Programmierer Programme für die gleiche Aufgabe, so gibt es wahrscheinlich mehrere Lösungswege. Unsere Programme sind nur eine mögliche Lösung des jeweiligen Problems.

Jedes Programm ist durch ein Flußdiagramm, das den globalen Programmablauf ohne Berücksichtigung spezieller Programmierungsdetails wiedergibt, durch Kommentare in der Befehlsliste und weitere Erläuterungen dokumentiert. Im ersten Befehl wird jeweils das Stackregister geladen, so daß jedes Programm in Einzelbefehlsausführung ab Programmanfang ablaufen kann.

7.1. Optische Effekte

7.1.1. Blinkendes Bild

Das Programm invertiert die Anzeigeseite mit einstellbarer Wiederholfrequenz. Auf der Anzeigeseite kann ein beliebiges Muster eingeschrieben werden. Die Wiederholfrequenz ist durch eine

16-Bit-Zahl vorgebbar.

Belegte Seiten:	0, 1
Anzeigeseite:	1
Startadresse:	0
Wiederholfrequenz in:	1E - 1F, letzte Zeile Seite 0

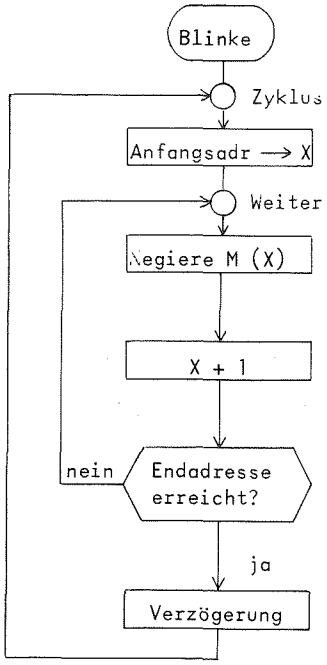


Bild 7.1 Ablaufdiagramm Programm "Blinkendes Bild"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07653) 444, 292
Fernschreiber 0772323 fmkg d

TV-Computersystem 6800

Aufgabe: Blinkendes Bild Seite: 0

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Blinke	LDS	I	3F5	8E 03	Stack liegt am Ende des Speichers
1	1					F5	
2	2	Zyklus	LDX	I	20	CE 00	Anfangsadr. Anzeigeseite (Seite 1) → X
3	3					20	
4	4		COM	X		00	Negiere
5	5					63	
6	6	Weiter	INX			00	X + 1
7	7		CPX	I	40	8C	Endadresse Seite 1 überschritten?
8	8					00	
9	9					40	
10	A		BNE	R	Weiter	26	Sprung, falls Adr. nicht erreicht
11	B					F8	
12	C		LDX	D	Zeit	DE	Verzögerungsschleife
13	D	Verzögerung				1E	Anfangswert Zähler laden
14	E		DEX			09	
15	F		BNE	R	Zähle	26	
16	10	Zähle				FD	
17	11		JMP	E	Zyklus	3E	Wiederhole Durchlauf
18	12					00	
19	13					03	
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D	Zeit				30	eine gefäßese Zahl verlangsamt, eine kleinere beschleunigt das Blinken.
30	1E					00	
31	1F						

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

7.1.2. Durchlaufendes Bild

Das Programm läßt das Bild auf der Anzeigeseite mit einstellbarer Geschwindigkeit von oben nach unten durchlaufen. Auf der Anzeigeseite kann ein beliebiges Muster vorgegeben werden. Die Geschwindigkeit ist durch eine 16-Bit-Zahl einstellbar.

Belegte Seiten:	0, 1 und 2
Anzeigeseite:	2
Startadresse:	0
Durchlaufzeit in:	20 - 21, erste Zeile Seite 1

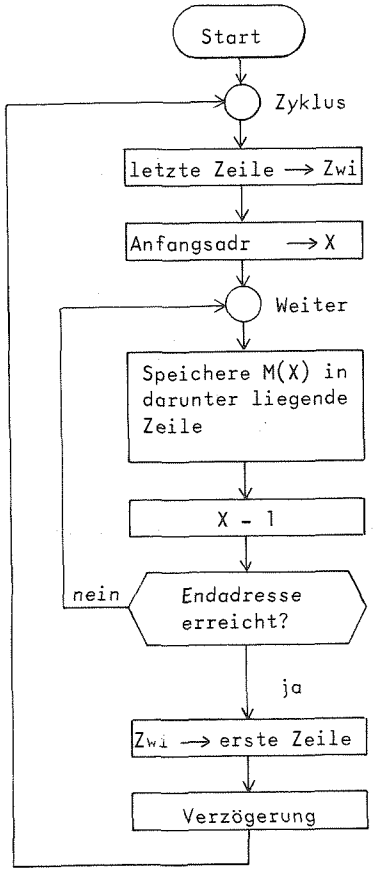


Bild 7.2 Ablaufdiagramm Programm "Durchlaufendes Bild"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Durchlaufendes Bild

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3	Zyklus	LDX	D	5E	DE	letzte Zeile der Anzeigeseite (Seite 2)
4	4					5E	zwischenspeichern
5	5		STX	D	Zwi	DF	
6	6					22	
7	7		LDX	I	5D	CE	Adresse vorletzte Zeile der Anzeigeseite
8	8					00	→ X
9	9					5D	
10	A	Weiter	LDAA	X	0	A6	durch X adressiertes Byte in darunter-
11	B					00	liegende Zeile speichern
12	C		STAA	X	2	A7	
13	D					02	
14	E		DEX			09	X-1
15	F		CPX	I	3F	8C	Anfangsadresse Anzeigeseite erreicht?
16	10					00	
17	11					3F	
18	12		BNE	R	Weiter	26	Sprung, wenn weiter zu transportieren ist
19	13					F6	
20	14		LDX	D	Zwi	DE	Zwischengespeicherte letzte Zeile wird
21	15					22	neue erste Zeile
22	16		STX	D	40	DF	
23	17					40	
24	18	Verzögerung	LDX	D	Zeit	DE	Verzögerungsschleife
25	19					20	lade Anfangswert Zähler
26	1A	Zähle	DEX			09	
27	1B		BNE	R	Zähle	26	
28	1C					FD	
29	1D		JMP		Zyklus	7E	
30	1E					00	
31	1F					03	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375.TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Durchlaufendes Bild

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Zeit				06	dieser Wert kann variiert werden
1	1					00	kleinere Zahl → schnellerer Durchlauf
2	2	Zwi				00	größere Zahl → langsamerer Durchlauf
3	3					00	Zwischenspeicher für letzte Zeile
4	4						
5	5						
6	6						
7	7						
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,

7.1.3. Bewegung zum Bildmittelpunkt

Das Programm bewegt den Inhalt der Anzeigeseite auf die Bildmitte zu. Es kann auf der Anzeigeseite ein beliebiges Muster vorgegeben werden, besonders günstig sind zum Mittelpunkt symmetrische Muster. Die Geschwindigkeit ist von der Größe einer 16-Bit-Zahl abhängig.

Belegte Seiten:	0 bis 4
Anzeigeseite:	3
Startadresse:	0
Geschwindigkeit in:	5E - 5F, Seite 2, letzte Zeile

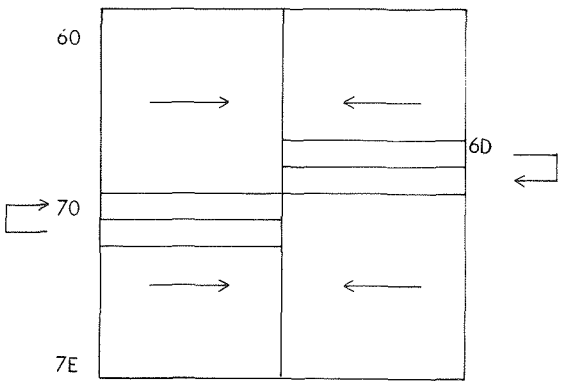


Bild 7.3 Wichtige Adressen auf der Anzeigeseite Seite 3

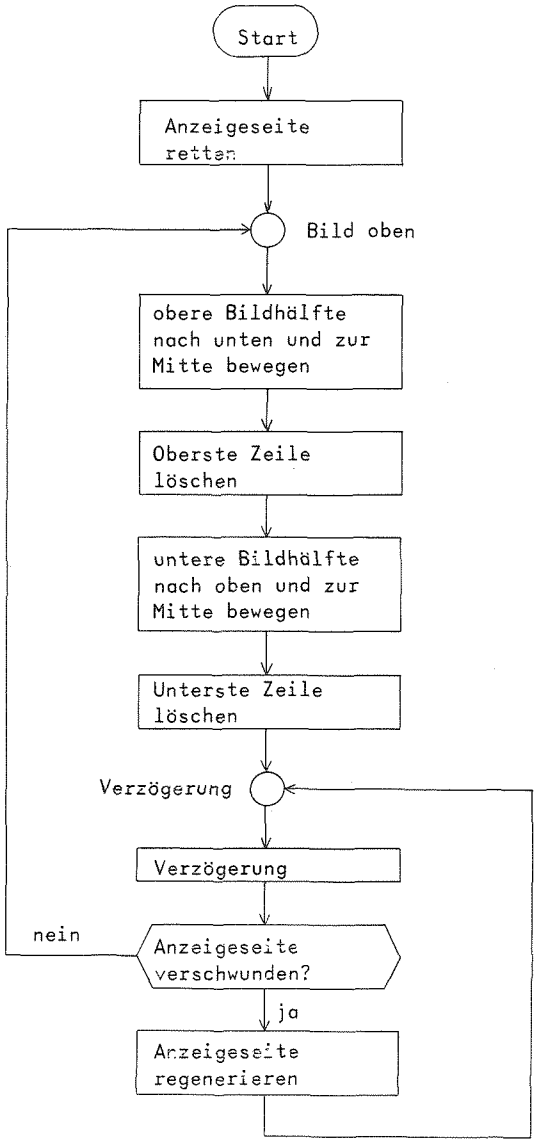


Bild 7.4 Ablaufdiagramm Programm "Bewegung zum Bildmittelpunkt"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Bewegung zum Mittelpunkt

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3FF	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					FF	
3	3		LDX	I	20	CE	Bild retten, Seite 3 → Seite 4 transp.
4	4					00	
5	5					20	
6	6	Transp1	LDAA	X	5F	A6	
7	7					5F	Anfandsadr. - 1 von Seite 3
8	8		STAA	X	7F	A7	
9	9					7F	Anfangsadr. - 1 von Seite 4
10	A		DEX			09	
11	B		BNE	R	Transp1	26	
12	C					F9	
13	D		LDAB	I	8	C6	ACCB: Zähler für Durchläufe, bis Seite 3
14	E					08	neu zu laden ist
15	F	Bild oben	LDX	I	6D	CE	obere Bildhälfte nach unten und zur
16	10					00	Mitte bewegen
17	11					6D	
18	12	Oben Forts.	LDAA	X	0	A6	rechtes Byte eine Stelle nach links
19	13					00	verschieben und in darunterliegende
20	14		ASLA			48	Zeile speichern
21	15		STAA	X	2	A7	
22	16					02	
23	17		DEX			09	
24	18		LDAA	X	0	A6	linkes Byte eine Stelle nach rechts
25	19					00	verschieben und in darunterliegende
26	1A		LSRA			44	Zeile speichern
27	1B		STAA	X	2	A7	
28	1C					02	
29	1D		DEX			09	
30	1E		NOP			02	
31	1F		NOP			02	

Seite = TV Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Bewegung zum Bildmittelpunkt

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		CPX	I	5F	8C	obere Bildhälfte abgearbeitet?
1	1					00	
2	2					5F	
3	3		BNE	R	Oben Forts.	26	
4	4					ED	
5	5		CLR	X	1	6F	Lösche oberste Zeile Seite 3
6	6					01	
7	7		CLR	X	2	6F	
8	8					02	
9	9	Bild unten	LDX	I	70	CE	untere Bildhälfte nach oben und zur
10	A					00	Mitte bewegen
11	B					70	
12	C	Unten Forts.	LDAA	X	2	A6	linkes Byte eine Stelle nach rechts
13	D					02	verschieben und in darüberliegende Zeile
14	E		LSRA			44	speichern
15	F		STAA	X	0	A7	
16	10					00	
17	11		INX			08	
18	12		LDAA	X	2	A6	rechtes Byte eine Stelle nach links
19	13					02	verschieben und in darüberliegende Zeile
20	14		ASLA			48	speichern
21	15		STAA	X	0	A7	
22	16					00	
23	17		INX			08	
24	18		CPX	I	7E	8C	untere Bildhälfte abgearbeitet?
25	19					00	
26	1A					7E	
27	1B		BNE	R	Unten Forts.	26	
28	1C					EF	
29	1D		CLR	X	0	6F	Lösche letzte Zeile Seite 3
30	1E					00	
31	1F		NOP			02	

Seite = TV Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (0 76 57) * 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: Bewegung zum Bildmittelpunkt

Bl	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		CLR	X	1	6F	
1	1					01	
2	2	Verzögerung	LDX	D	Zeit	DE	Verzögerungsschleife
3	3					5E	Lade Anfangswert Zähler
4	4	Zähle	DEX			09	
5	5		BNE	R	Zähle	26	
6	6					FD	
7	7		DECB			5A	Test, ob Bild neu zu laden ist
8	8		BEQ	R	Seite laden	27	
9	9					03	
10	A		JMP	E	Bild oben	7E	
11	B					00	
12	C					0F	
13	D	Seite laden	LDX	I	20	CE	Anzeigeseite regenerieren
14	E					00	Seite 4 → Seite 3 transportieren
15	F					20	
16	10	Transp2	LDAA	X	7F	A6	
17	11					7F	Anfangsadr. - 1 von Seite 4
18	12		STAA	X	5F	A7	
19	13					5F	Anfangsadr. - 1 von Seite 3
20	14		DEX			09	
21	15		BNE	R	Transp2	26	
22	16					F9	
23	17		LDAB	I		C6	ACCB: Zähler für Anzahl der Durchläufe,
24	18					09	bis Seite 3 neu zu laden ist
25	19		JMP	E	Verzögerung	7E	
26	1A					00	
27	1B					42	
28	1C						
29	1D						
30	1E	Zeit				0A	Verzögerungszeit kann variiert werden
31	1F					00	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.1.4. Zählseite

Das Programm zählt in der Anzeigeseite von oben nach unten fortlaufend mit einstellbarer Geschwindigkeit. Der Anfangswert des Zählers ist vorgebbar. Ist die letzte Stelle des Anfangswerts F, so enthalten die rechten vier Bits auf der Anzeigeseite die Werte 0 bis F in aufsteigender Reihenfolge.

Belegte Seiten: 0 - 2
Anzeigeseite: 2
Startadresse: 0
Adresse Zählzeit: 3C - 3D, vorletzte Zeile Seite 1
Adresse Anfangswert: 3E - 3F, letzte Zeile Seite 1

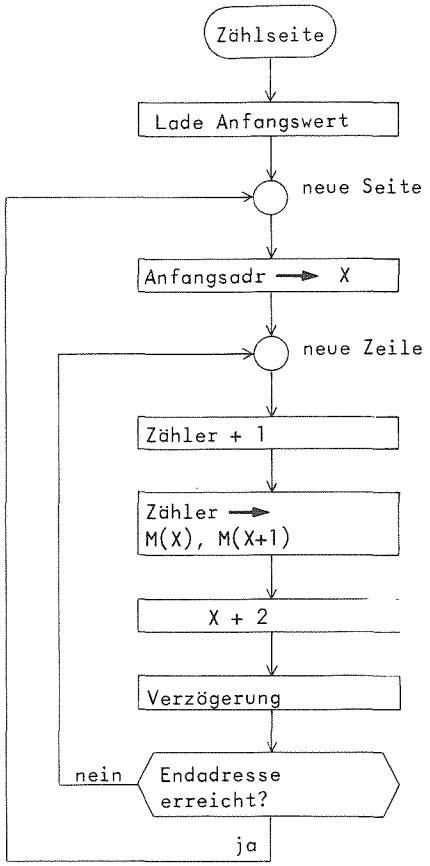


Bild 7.5 Ablaufdiagramm "Zählseite"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach-Hochschwarzwald 1
Fernruf Eisenbach (07657) 444,292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Aufgabe: Zählseite Seite: 0

Nr.	Byte	Marke	OPEX	A	Symb. Adr	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		LDAA	D	Anf.wert	96	AccA: höherwertiger Anteil des Zählers
4	4					3E	
5	5		LDAB	D	Anf.wert+1	D6	AccB: niederwertiger Anteil des Zählers
6	6					3F	
7	7	neue Seite	LDX	I	40	CE	Anfangsadr. Seite 2 → X
8	8					00	
9	9					40	
10	A	neue Zeile	INCB	R	kein Übertr.	5C	Zähler + 1
11	B		BNE			26	falls niederwertiges Byte in AccB = 0 ist
12	C					01	ein Übertrag zum höherwertigen Byte in
13	D		INCA			4C	AccA zu berücksichtigen
14	E	kein Übertr.	STAA	X	0	A7	Zähler abspeichern
15	F					00	
16	10		STAB	X	1	E7	
17	11					01	
18	12		INX			08	
19	13		INX			08	
20	14		STX	D	Adr	DF	aktuelle Abspeicheradresse zwischenspeichern
21	15					3A	X und in Verzögerungsschleife benötigt
22	16	Verzögerung	LDX	D	Zeit	DE	Verzögerungsschleife
23	17					3C	
24	18	Zähle	DEX			09	
25	19		BNE	R	Zähle	26	
26	1A					FD	
27	1B		LDX	D	Adr	DE	Abspeicheradresse laden
28	1C					3A	
29	1D		CPX	I		8C	Endadresse erreicht?
30	1E					00	
31	1F					20	

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Zählseite

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		BNE	R	neue Zeile	26	nein, auf Seite weiter zählen
1	1					E8	
2	2		JMP	E	neue Seite	7E	ja, am Seitenanfang wieder beginnen
3	3					00	
4	4					07	
5	5						
6	6						
7	7						
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A	Adr				00	Hilfzelle zum Zwischenspeichern der
27	1B					00	aktuellen Adresse
28	1C	Zeil				01	Verzögerungsfaktor, kann variiert werden
29	1D					00	
30	1E	Anf.wert				11	Anfangswert Zähler, kann variiert werden
31	1F					FF	

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV Bild
Byte = Nr. auf Seite
OPEX = externer Code

7.2. Umwandlung Dual ⟶ Dezimal

Das Programm wandelt eine positive 8 Bit Dualzahl in eine Dezimalzahl in BCD-Darstellung um.

Wir gehen vor wie bei der Umrechnung von Hand, wo man die Werte der Zweierpotenzen der mit 1 belegten Binärstellen addiert. Die hierzu benötigten Werte von 2^0, 2^1, ... , 2^7 sind in BCD-Darstellung in einer zum Programm gehörigen Tabelle gespeichert. Die größtmögliche Zahl ist 255. Für das Ergebnis müssen wir also 2 Bytes vorsehen. Die Addition der Zweierpotenz mit anschließender Dezimalangleichung wird für eine doppeltlange Zahl ausgeführt, d. h. zunächst Addition des niederwertigen Bytes und anschließend Addition des höherwertigen Bytes unter Berücksichtigung eines Übertrags. Die Additionsschleife endet, sobald der Rest der bitweise verarbeiteten Dualzahl Null wird. Dann liegt bereits das Ergebnis vor.

Belegte Seiten:	0 - 2
Anzeigeseite:	2
Startadresse:	0
Adresse Dualzahl:	41, erste Zeile Seite 2, rechtes Byte
Adresse Ergebnis:	42 - 43, zweite Zeile Seite 2

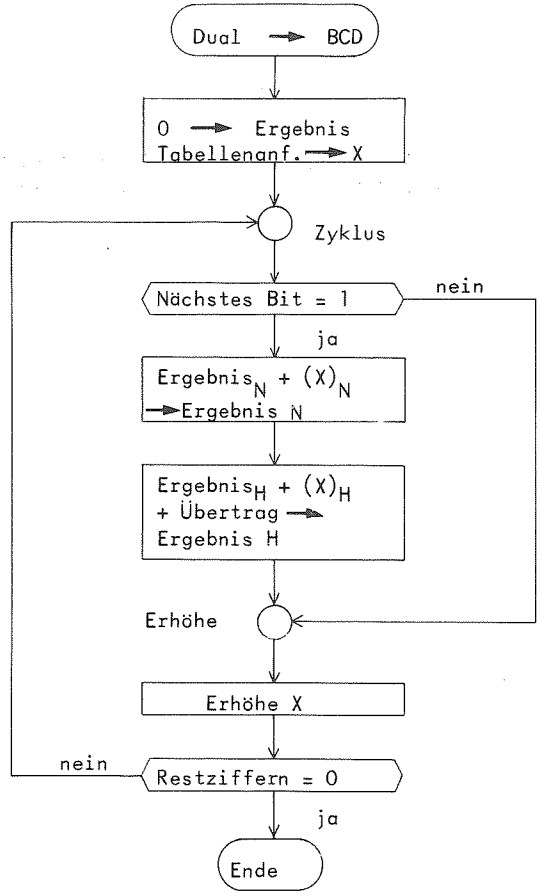


Bild 7.6 Ablaufdiagramm Programm "Dual → Dezimal"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach-Hochschwarzwald 1
Fernruf Eisenbach (0 76 57) * 444, 292
Fernschreiber 0772323 fmkg d

TV-Computersystem 6800 Seite 0

Aufgabe: Umwandlung Dual → BCD

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3FF	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	Lösche beide Bytes des Ergebnisses
3	3		CLR	E	Ergebnis	7F	
4	4					00	
5	5					43	
6	6		CLR	E	Ergebnis+1	7F	
7	7					00	
8	8					43	
9	9		LDX	I	Tabelle	CE	Anfangsadresse der Tabelle der 2ueier-
A	10					00	potenzen → X
B	11					26	
C	12		LDAB	D	Dualzahl	D6	ACCB nimmt Dualzahl für die bitweise
D	13					41	Verarbeitung auf
E	14	Zyklus	LSRB			54	nächstes Bit
F	15		BCC	R	Erhöhe X	24	= 0, nur Tabellenindex weiterschalten
10	16					0D	
11	17	Addiere	LDAA	D	Ergebnis+1	96	= 1, Tabellenwert zu Ergebnis
12	18					43	addieren,
13	19		ADDA	X	1	AB	wiederum 2ehner Bytes addieren
14	20					01	
15	21		DAA			19	Dezimalangleichung
16	22		STAA	D	Ergebnis+1	97	
17	23					43	
18	24		LDAA	D	Ergebnis	96	höher wertige Bytes mit Berücksichtigung
19	25					42	eines Übertrags addieren
1A	26		ADCA	X	0	A9	
1B	27					00	
1C	28		STAA	D	Ergebnis	97	
1D	29					42	
1E	30	Erhöhe X	INX			08	
1F	31		INX			08	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Umwandlung Dual → BCD

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		TSTB	R		5D	Restziffern = 0 ?
1	1	STOP	BEQ		STOP	27	ja, Stop
2	2					FE	
3	3		JMP	E	Zyklus	7E	nein, wiederhole Zyklus
4	4					00	
5	5					0E	
6	6	Tabelle				00	Tabelle von 2° bis 2⁷ im BCD -
7	7					01	Darstellung (2 Bytes pro Eintrag)
8	8					00	
9	9					02	
10	A					00	
11	B					04	
12	C					00	
13	D					08	
14	E					00	
15	F					16	
16	10					00	
17	11					32	
18	12					00	
19	13					64	
20	14					01	
21	15					28	
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

A = Adressenmodus
CODE = interner Code

Seite = TV-Bild
Byte = Nr auf Seite
OPEX = externer Code

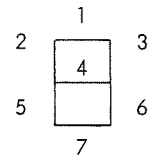
A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

7.3. Sieben-Segment-Anzeige

Aufgabe: Es sollen die Ziffern 0 - 9 nach dem Prinzip der 7-Segmentanzeige auf dem Bildschirm darstellbar sein unter Ausnutzung der graphischen Möglichkeiten bei der bitweisen Darstellung.

Die sieben Segmente numerieren wir wie folgt durch:

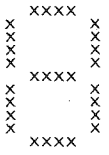


Für jede Ziffer können wir in einer Tabelle beschreiben welche Segmente hell (1) oder dunkel (0) zu steuern sind. Wir ergänzen die Tabelle um eine 8. Spalte, die immer mit Null belegt ist. Damit können wir die Anzeigevorschrift für eine Ziffer in einem Byte verschlüsseln.

Ziffer \ Segment	1	2	3	4	5	6	7	8	Verschlüsselungsbyte
0	1	1	1	0	1	1	1	0	EE
1	0	0	1	0	0	1	0	0	24
2	1	0	1	1	1	0	1	0	BA
3	1	0	1	1	0	1	1	0	B6
4	0	1	1	1	0	1	0	0	74
5	1	1	0	1	0	1	1	0	D6
6	1	1	0	1	1	1	1	0	DE
7	1	0	1	0	0	1	0	0	A4
8	1	1	1	1	1	1	1	0	FE
9	1	1	1	1	0	1	1	0	F6

Tabelle 7.7 Verschlüsselung der Ziffern 0 - 9 für die Siebensegmentanzeige.

Für die Darstellung auf dem Bildschirm wählten wir für ein Segment 4 Bitpositionen. Somit belegt eine Ziffer in der Höhe 11 und in der Breite 6 Felder. Die Breite ergänzen wir durch je ein Leerfeld rechts und links auf 1 Byte.



Um eine allgemeine Verwendbarkeit zu erreichen, entwerfen wir ein Unterprogramm "Anzeige" mit zwei Parametern

ACCA : anzuzeigende Ziffer zwischen 0 und 9
X : Adresse für Anzeige von Segment 1
(oberer Rand der Ziffer)

Durch den Parameter im Indexregister kann die Ziffer auf einer beliebigen Seite in beliebiger Höhe positioniert werden. Ferner ist durch Angabe einer geraden bzw. ungeraden Anfangsadresse die linke bzw. rechte Bytereihe auf dem Bildschirm auswählbar. Dieses Unterprogramm könnte in Spielprogrammen für die Anzeige von Ergebnissen eingesetzt werden.

Das Hauptprogramm besteht aus einer Zählschleife von 0 - 9 mit einem Aufruf von Unterprogramm Anzeige und anschließender vorgebbarer Verzögerung.

Belegte Seiten: 0 - 5
Startadresse: 0
Anzeigeseite: 5
Verzögerung in: 1E - 1F, letzte Zeile Seite 0

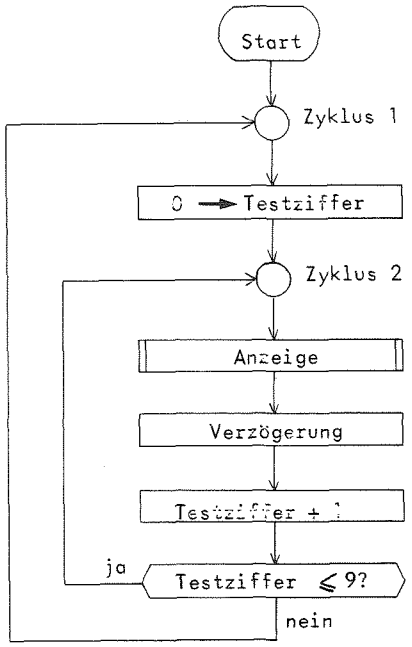


Bild 7.8 Ablaufdiagramm Hauptprogramm "Siebensegmentanzeige"

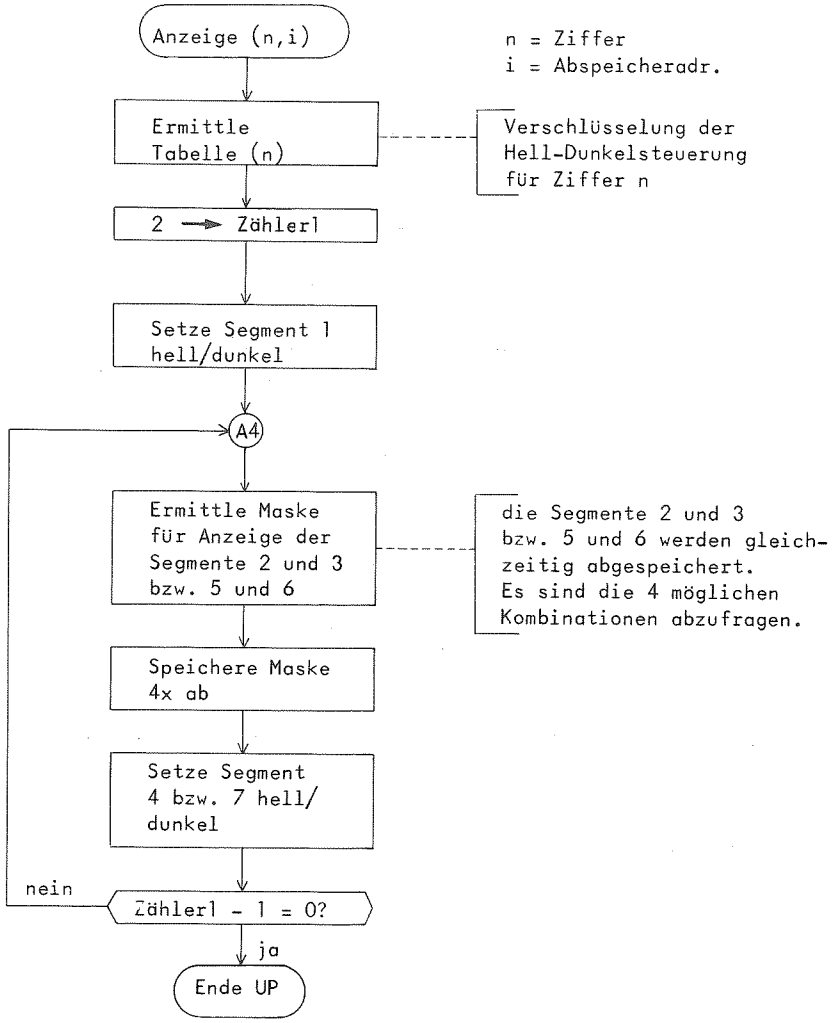


Bild 7.9 Ablaufdiagramm Unterprogramm "Anzeige"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444,292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: 7-Segmentanzeige, Hauptprogramm

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2	Zyklus 1	CLRA	D		F5	0 → Testziffer
3	3	Zyklus 2	STAA	D	Testziffer	97	Zwischenspeicherung der Testziffer
4	4		LDX	I	A6	1C	X = Adresse für Anzeige Segment 1
5	5					CE	mit Adresse A7 sind die rechte Byte-
6	6					00	reihe angewählt!
7	7		JSR	E	Anzeige	A6	Acc A : Testziffer
8	8					BD	
9	9					00	
A	10		LDX	D	Zeit	20	Verzögerungsschleife
B	11					DE	
C	12					1E	
D	13	Zähle	DEX			09	
E	14		BNE	R	Zähle	26	
F	15					FD	
10	16		LDAA	D	Testziffer	96	Erhöhe Testziffer
11	17					1C	
12	18		INCA			4C	
13	19		CMPA	I	9	81	≤ 9 ?
14	20					09	
15	21		BLS	R	Zyklus 2	23	ja, nächste Ziffer anzeigen
16	22					EC	
17	23		JMP	E	Zyklus 1	7E	Zyklus wieder mit 0 beginnen
18	24					00	
19	25					03	
1A	26						
1B	27						
1C	28	Testziffer				00	Hilfszelle
1D	29					00	
1E	30	Zeit				20	Verzögerungsfaktor, kann variiert werden
1F	31					00	

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: 7-Segmentanzeige, UP Anzeige

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Anzeige	STX	D	zwX	DF	Index zwischenspeichern
1	1					86	
2	2		LDX	I	Tabelle-1	CE	Anfangsadr.-1 mit Verschlüsselung der
3	3					00	Hell-Dunkelsteuerung
4	4					71	
5	5		INCA			4C	Testziffer+1 zur einfacheren Schleifen-
6	6	A1	INX			08	bildung
7	7		DECA			4A	Berechne Adresse des n-ten Tabelleneintrags
8	8		BNE	R	A1	26	
9	9					FC	
10	A		LDAA	X	0	A6	ACCA = Verschlüsselung zur Hell-/
11	B					00	Dunkelsteuerung
12	C		LDX	D	zwX	DE	Abspeicheradresse zurück laden
13	D					86	
14	E		LDAB	I	2	C6	2 → Zähler 1
15	F					02	
16	10		STAB	D	Zähler1	D7	
17	11					84	
18	12	A2	CLRB			5F	Vorbesetzung ACCB, falls Segment 1 dunkel
19	13		ASLA			48	nächstes Bit = Steuerung Anzeige
20	14		BCC	R	A3	24	Segment 1
21	15					02	
22	16		LDAB	D	Maske 1	D6	Segment 1 hell steuern.
23	17					80	
24	18	A3	STAB	X	0	E7	Setze Segment 1
25	19					00	
26	1A		INX			08	
27	1B		INX			08	
28	1C	A4	LDAB	I	4	C6	4 → Zähler 2, steuert Schleife zum
29	1D					04	Abspeichern der Punkte für Segmente
30	1E		STAB	D	Zähler 2	D7	2/3 bzw. 5/6
31	1F					85	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: 7-Segmentanzeige, UP Anzeige

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		CLRB			5F	Vorbesetzung dunkel
1	1		ASLA			48	nächstes Bit, Segment 2 bzw. 5
2	2		BCC	R	A6	24	
3	3					0B	
4	4		ASLA			48	nächstes Bit, Segment 3 bzw. 6
5	5		BCC	R	A5	24	
6	6					04	
7	7		LDAB	D	Maske 23	D6	beide Segmente hell
8	8					83	
9	9		BRA	R	A7	20	
10	A					09	
11	B	A5	LDAB	D	Maske 2	D6	Segment 2 bzw 5 hell, 3 bzw 6 dunkel
12	C					81	
13	D		BRA	R	A7	20	
14	E					05	
15	F	A6	ASLA			48	nächstes Bit, Segment 3 bzw 6
16	10		BCC	R	A7	24	
17	11					02	
18	12		LDAB	D	Maske 3	D6	Segment 2 bzw 5 dunkel, 3 bzw 6 hell
19	13					82	
20	14	A7	STAB	X	0	E7	Schleife, um Segmente 2/3 bzw 5/6
21	15					00	4x abzuspeichern
22	16		INX			08	
23	17		INX			08	
24	18		DEC	E	Zähler 2	7A	
25	19					00	
26	1A					85	
27	1B		BNE	R	A7	26	
28	1C					F7	
29	1D		CLRB			5F	Vorbesetzung dunkel
30	1E		ASLA			48	nächstes Bit, Segment 4/7
31	1F		NOP			02	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 3

Aufgabe: 7-Segmentanzeige, UP Anzeige

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		BCC	R	A8	24	
1	1					02	
2	2		LDAB	D	Maske 1	D6	Segment 4/7 hell
3	3					80	
4	4	A8	STAB	X	0	E7	Segment 4/7 setzen
5	5					00	
6	6		INX			08	
7	7		INX			08	
8	8		DEC	E	Zähler 1	7A	Zähler 1 - 1
9	9					00	
10	A					84	
11	B		BEQ	R	A9	27	= 0 Ende
12	C					03	
13	D		JMP	E	A4	7E	≠ 0 Anzeige von Segment 5-7
14	E					00	
15	F					3C	
16	10	A9	RTS			39	Rücksprung aus UP Anzeige
17	11					00	
18	12	Tabelle				EE	Verschlüsselung der Hell-Dunkelsteuerung der Ziffern 0-9
19	13					24	
20	14					BA	
21	15					B6	
22	16					74	
23	17					D6	
24	18					DE	
25	19					A4	
26	1A					FE	
27	1B					F6	
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 4

Aufgabe: 7-Segmentanzeige

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Maske 1				3C	
1	1	Maske 2				40	
2	2	Maske 3				02	
3	3	Maske 23				42	
4	4	Zähler 1				00	
5	5	Zähler 2				00	
6	6	Zwx				00	
7	7					00	
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.4. Sortierprogramm

Sortieren gehört zu den Standardaufgaben der Datenverarbeitung. Die Aufgabe besteht allgemein darin, eine Datenmenge, deren Elemente ein Ordnungskriterium besitzen, in eine geordnete Reihenfolge zu bringen. In der Literatur sind zahlreiche Verfahren einschließlich Aufwandsbetrachtungen beschrieben, damit wollen wir uns hier nicht weiter befassen. Ein programmtechnisch einfaches Verfahren soll das Prinzip demonstrieren. Wir wählten das "Sortieren durch Vertauschen" (siehe /3/).

Gegeben sei eine Liste von Zahlen, die in aufsteigende Reihenfolge gebracht werden sollen.

Im ersten Schritt werden das erste und das zweite Element verglichen. Ist das erste größer als das zweite, so werden die beiden vertauscht. Das Vergleichen und eventuelle Vertauschen wird mit dem zweiten und dritten Element fortgesetzt usw. bis schließlich das vorletzte mit dem letzten Element verglichen und eventuell vertauscht wird. Nach diesem ersten Durchlauf steht das größte Element auf dem letzten Listenplatz. Dem gleichen Ablauf unterzieht man im zweiten Durchgang die am Ende um eins verkürzte Liste. Danach steht das insgesamt zweitgrößte Element auf dem vorletzten Platz. Das beschriebene Verfahren setzt man jeweils mit einer um eins kürzeren Restliste fort, bis die Liste nur noch aus einem Element besteht. Dieses steht auf Grund des Verfahrens bereits auf der richtigen Position.

Das Verfahren wird an einem Beispiel mit 5 Zahlen durchgespielt. Die Vergleiche sind durch ☐ gekennzeichnet, die Restliste ist durch einen waagrechten Strich abgetrennt.

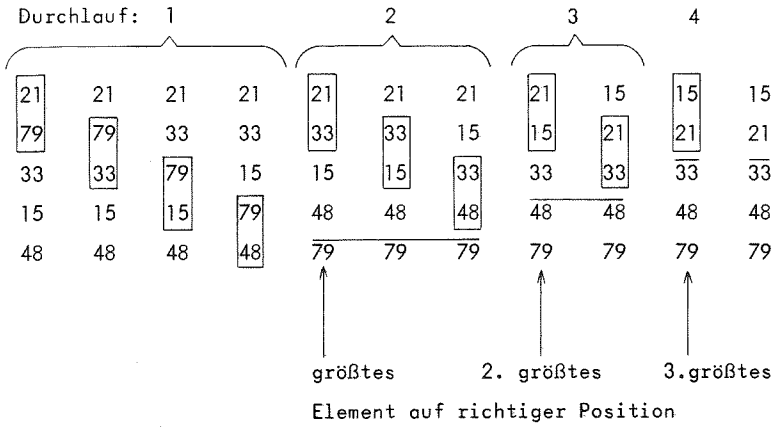


Bild 7.10 Beispiel zum Sortieren durch Vertauschen

Das Programm sortiert 16 Zahlen, die in der linken Bytereihe von Seite 2 stehen, in aufsteigender Reihenfolge. Zum besseren Verfolgen des Programmablaufs wird die unsortierte Liste zuerst in die rechte Bytereihe übertragen. Nach dem 1. Durchlauf hält das Programm bei einem WAI Befehl an. (Dieser kann durch NOP -02- überschrieben werden, dann läuft die Sortierung ohne Unterbrechung ab). Durch einen Interrupt wird der jeweils nächste Durchlauf gestartet. Als Interruptprogramm kann das normale Einzelbefehlsprogramm (siehe 3.5) genommen werden. Es ist dann unter Adresse 3F8, 3F9 (vierte Zeile von unten auf Seite 1F) ebenfalls die Adresse 3F6 einzutragen.

Belegte Seiten:	0 - 2
Anzeigeseite:	2
Startadresse:	0
Interruptadresse:	3F6 (mit 02 3B unter dieser Adresse)

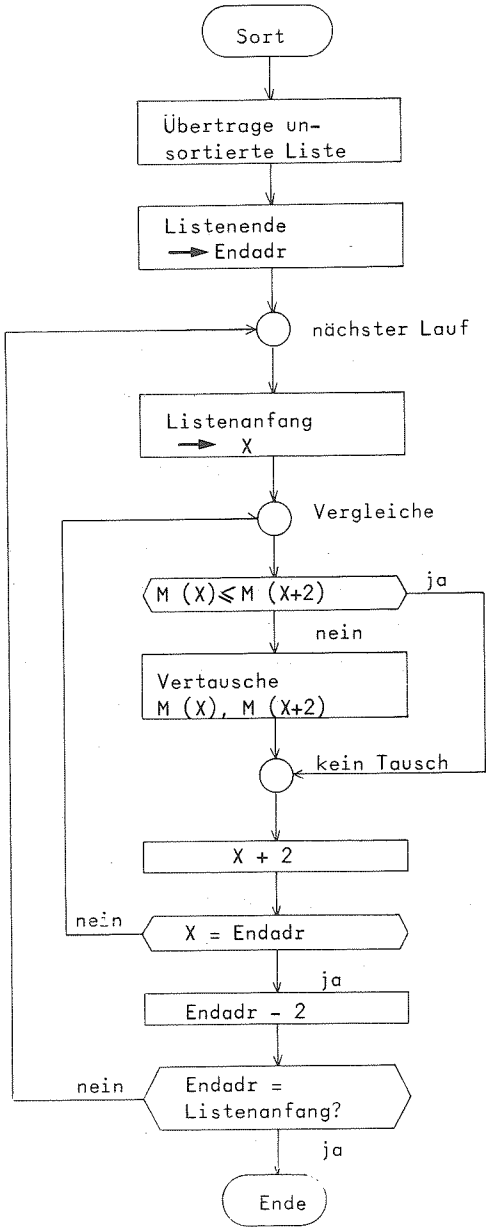


Bild 7.11 Ablaufdiagramm "Sortieren durch Vertauschen"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07655) 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Sortieren durch Vertauschen

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		CLI			0E	Lösche Unterrupmaske
4	4		LDX	I	20	CE	Übertrage unsortierte Liste in linker Byte-
5	5					00	reihe von Seite 2 in rechte Bytereihe von
6	6					20	Seite 2
7	7	Transport	LDAA	X	3E	A6	
8	8					3E	
9	9		STAA	X	3F	A7	
A	A					3F	
B	B		DEX			09	
C	C		DEX			09	
D	D		BNE	R	Transport	26	
E	E		LDAA	I	5E	F8	Adresse Listenende → Endadr
F	F					86	
10	10		STAA	D	Endadr +1	5E	
11	11					97	
12	12					3F	
13	13	nächster L.	LDX	I	40	CE	Anfangsadr. → X
14	14					00	
15	15					40	
16	16	Vergleiche	LDAA	X	0	A6	Vergleiche zwei aufeinander folgende
17	17					00	Worte und vertausche sie, falls erste
18	18		CMPA	X	2	A1	Zahl größer als zweite ist.
19	19					02	
1A	1A		BLS	R	kein Tausch	23	
1B	1B					06	
1C	1C	Tausch	LDAB	X	2	E6	
1D	1D					02	
1E	1E		STAA	X	2	A7	
1F	1F					02	

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Sortieren durch Vertauschen

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		STAB	X	0	E7	
1	1					00	
2	2	kein Tausch	INX			08	X+2
3	3		INX			08	
4	4		CPX	D	Endadr	9C	aktuelles Listenende erreicht?
5	5					3E	
6	6		BNE	R	Vergleiche	26	nein, setze Vergleich fort
7	7					EE	
8	8		DEC	E	Endadr +1	7A	ja, erniedrige die aktuelle Endadr.
9	9					00	
10	A					3F	
11	B		DEC	E	Endadr +1	7A	
12	C					00	
13	D					3F	
14	E	Warte	WAI			3E	Warten, Interrupt startet nächsten Durch-
15	F		LDX	I	40	CE	lauf. WAI kann durch NOP (02) ersetzt
16	10					00	werden
17	11					40	Aufangsadr = Endadr?
18	12		CPX	D	Endadr	9C	
19	13					3E	
20	14	Stop	BEQ	R	Stop	27	ja, Ende
21	15					FE	
22	16		JMP	E	nächster L.	7E	nein, nächster Durchlauf
23	17					00	
24	18					13	
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E	Endadr				00	Hilfszelle für aktuelle Endadresse
31	1F					00	(2 Bytes)

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.5. Demonstration der Mengenoperation Durchschnittsbildung

Das Programm bildet die Durchschnittsmenge von zwei Mengen mit je maximal 5 Elementen. Dabei werden die optischen Möglichkeiten bei bitweiser Darstellung ausgenützt. Zur Anzeige der Mengen wählten wir folgende Einteilung des Fernsehbildes:

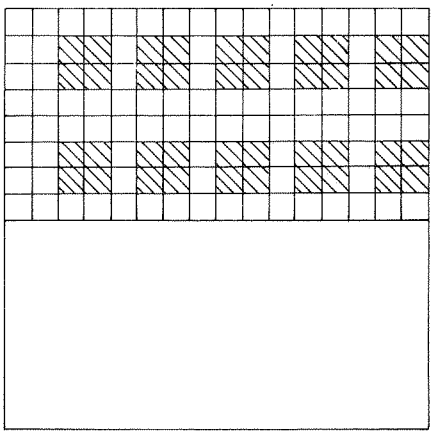
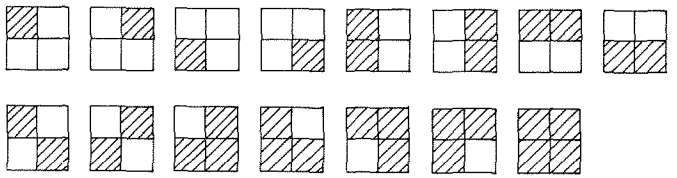


Bild 7.12 Aufteilung des Fernsehbildes bei Programm "Durchschnittsbildung"

In der linken Bytereihe stehen in zwei Reihen untereinander die Elemente der Menge 1 und der Menge 2. Auf die genaue Einhaltung dieser Einteilung ist bei der Eingabe besonders zu achten. In der rechten Bytereihe speichert das Programm die Durchschnittsmenge ab.

Ein Element wird durch 2 x 2 Bits dargestellt. Eine Elementposition, in der alle vier Bits Null sind, ist nicht belegt.

Es gibt damit folgende 15 Elemente:



Ein Programm, das die Vereinigungsmenge bildet, ist in ähnlicher Weise realisierbar.

Belegte Seiten:	0 - 4
Anzeigeseite:	4
Startadresse:	0

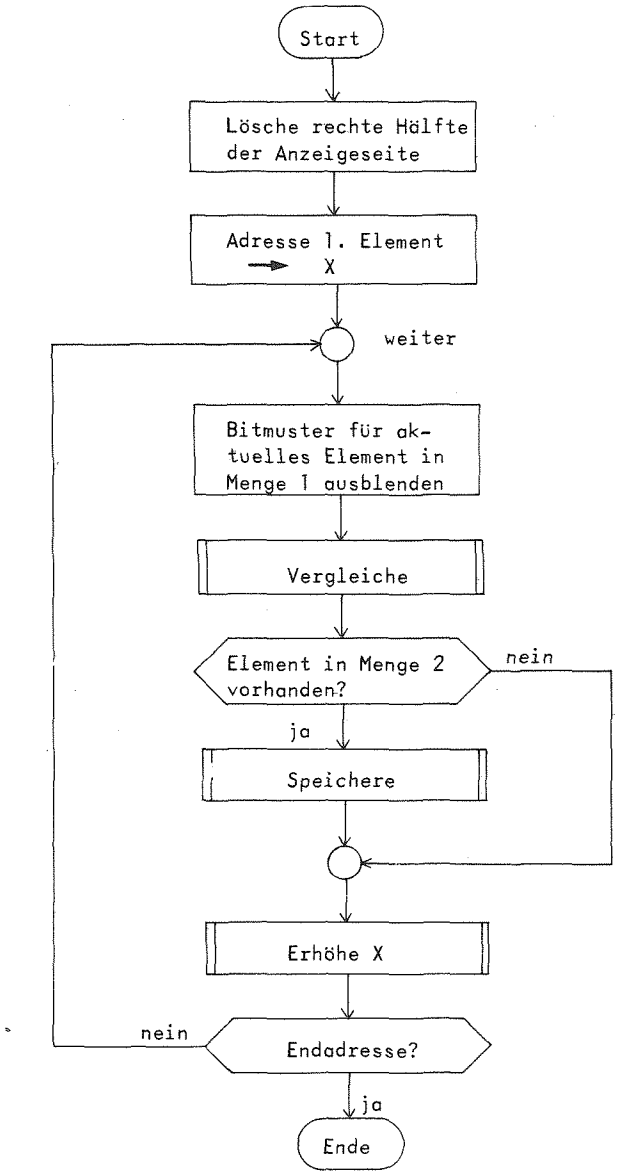


Bild 7.13 Ablaufdiagramm "Durchnittsbildung"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Durchschnittsbildung

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		LDX	I	20	CE	Lösche rechte Bytereihe der Anzeigeseite,
4	4					00	Seite 4
5	5					20	
6	6	Lösche	CLR	X	7F	6F	
7	7					7F	Anfangsadr. -1 Seite 4
8	8		DEX			09	
9	9		DEX			09	
10	A		BNE	R	Lösche	26	
11	B					FA	
12	C		LDX	I	80	CE	Anfangsadr Seite 4 → X
13	D					00	
14	E					80	
15	F		STX	D	XM3	DF	XM3: aktuelle Abspeicheradr. für Durch-
16	10					7C	schnittsmenge
17	11	Weiter	STX	D	XM1	DF	XM1: aktuelle Vergleichsadr. in Menge 1
18	12					7A	
19	13		LDAA	X	0	A6	Bitmuster des aktuellen Elements in
20	14					00	Menge 1 ausblenden und als Parameter
21	15		ANDA	I	60	84	für das Vergleichsprogramm in den
22	16					60	Zellen V1 und V2 bereitstellen
23	17		STAA	D	V1	97	
24	18					78	
25	19		LDAA	X	2	A6	
26	1A					02	
27	1B		ANDA	I	60	84	
28	1C					60	
29	1D		STAA	D	V2	97	
30	1E					79	
31	1F		NOP			02	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu = A oder B

0375.TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Durchschnittsbildung

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		ORAA	D	V1	9A	prüfe, ob leeres Element, dieses wird
1	1					78	nicht verglichen
2	2		BEQ	R	Erhöhe	27	
3	3					0B	
4	4		JSR	E	Vergleiche	BD	Suche aktuelles Element von Menge 1 in Menge 2
5	5					00	
6	6					3A	Ergebnisparameter:
7	7		TSTA			4D	ACCA = 0: Element in Menge 2 vorhanden
8	8		BNE	R	nicht Sp.	26	" ≠ 0: nicht vorhanden
9	9					03	
10	A		JSR	E	Speichere	BD	Speichere Element in Durchschnittsmenge
11	B					00	
12	C					60	
13	D	nicht Sp.	LDX	D	XM1	DE	Adr. in Menge 1 →X
14	E					7A	
15	F	Erhöhe	JSR	E	Erhöhe X	BD	X + 6
16	10					00	
17	11					70	
18	12		CPX	I		8C	Endadresse erreicht ?
19	13					00	
20	14					9E	
21	15	Stop	BEQ	R	Stop	27	ja, Stop
22	16					FE	
23	17		JMP	E	Weiter	7E	nein, nächster Vergleich
24	18					00	
25	19					11	
26	1A	Vergleiche	LDX	I	80	CE	Anfangsadr. Seite 4 → X
27	1B					00	
28	1C					80	
29	1D	VGL1	LDAA	X	0	A6	
30	1E					00	
31	1F		NOP			02	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf Eisenbach (07657) * 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: Durchschnittsbildung

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		ASLA			48	das Bitmuster eines Elements der
1	1		ASLA			48	Menge 2 muß in die gleiche Position
2	2		ASLA			48	wie Element der Menge 1 gebracht
3	3		ASLA			48	werden
4	4		CMPA	D	V1	91	vergleiche erstes Wort
5	5					78	
6	6		BNE	R	VGL2	26	≠, mit nächstem Element fortfahren
7	7					0C	
8	8		LDAA	X	2	A6	=, zweites Wort bearbeiten
9	9					02	
10	A		ASLA			48	
11	B		ASLA			48	
12	C		ASLA			48	
13	D		ASLA			48	
14	E		CMPA	D	V2	91	Vergleiche zweites Wort
15	F					79	
16	10		BNE	R	VGL2	26	≠, weitersuchen
17	11					02	
18	12		CLRA			4F	Element gefunden 0→ ACCA
19	13		RTS			39	Rücksprung UP Vergleiche
20	14	VGL2	JSR	E	Erhöhe X	BD	X + 6
21	15					00	
22	16					70	
23	17		CPX	I	9E	8C	Endadiesse erreicht?
24	18					00	
25	19					9E	
26	1A		BNE	R	VGL1	26	nein, Vergleich fortsetzen
27	1B					E1	ja, Element nicht gefunden
28	1C		INCA			4C	ACCA ≠0 setzen
29	1D		RTS			39	Rücksprung UP Vergleiche
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf Eisenbach (07657) * 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 3

Aufgabe: Durchschnittsbildung

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Speichere	LDX	D	XM3	DE	Abspeicher adr. → X
1	1					7C	
2	2		LDAA	D	V1	96	Bitmuster 1. Wort abspeichern
3	3					78	
4	4		STAA	X	1	A7	
5	5					01	
6	6		LDAA	D	V2	96	Bitmuster 2. Wort abspeichern
7	7					79	
8	8		STAA	X	3	A7	
9	9					03	
10	A		JSR	E	Erhöhe X	BD	X + 6
11	B					00	
12	C					70	
13	D		STX	D	XM3	DF	Adr. reservieren
14	E					7C	
15	F		RTS			39	Ende UP Speichere
16	10	Erhöhe X	INX			08	X + 6
17	11		INX			08	
18	12		INX			08	
19	13		INX			08	
20	14		INX			08	
21	15		INX			08	
22	16		RTS			39	Ende UP Erhöhe X
23	17						
24	18	V1				00	Bitmuster 1. Wort des aktuellen Elements
25	19	V2				00	" 2. Wort "
26	1A	XM1				00	aktuelle Adr. in Menge 1
27	1B					00	
28	1C	XM3				00	aktuelle Adr. in Durchschnittsmenge
29	1D					00	
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.6. Spielprogramme

7.6.1. Reaktionsspiel Kegeln

Auf der Anzeigeseite (bitweise Darstellung) sind im oberen Bildteil neun Felder in entsprechender Anordnung als "Kegel" hell gesetzt. In der letzten Zeile der Anzeigeseite läuft eine "Kugel"mit einstellbarer Geschwindigkeit dauernd von links nach rechts. Durch Betätigen der Interrupttaste läuft die Kugel nach oben und löscht die in der Bahn stehenden Kegel. Die Interrupttaste kann beliebig oft geschaltet werden, bis alle Kegel gelöscht sind. Ein erneuter Programmstart stellt die Anfangsbedingung wieder her.

Belegte Seiten:	0 - 4
Anzeigeseite:	4
Startadresse:	0
Interruptadresse:	39

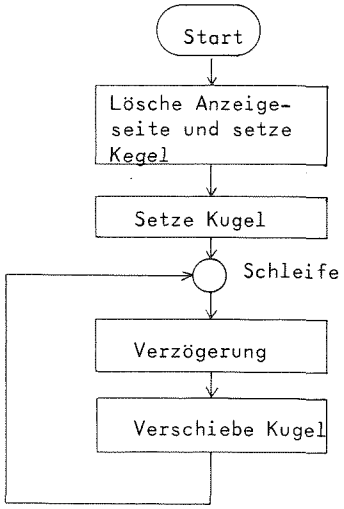


Bild 7.14 Ablaufdiagramm Hauptprogramm "Kegeln"

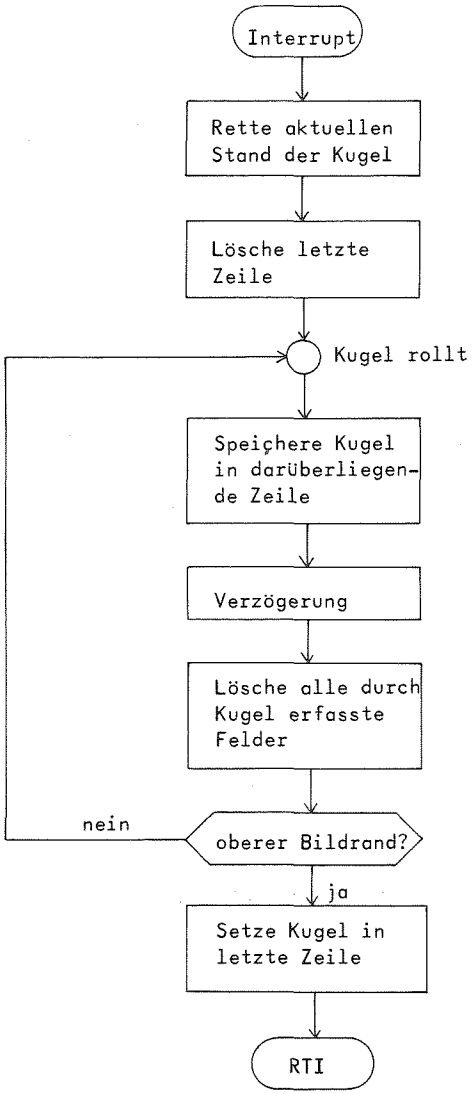


Bild 7.15 Ablaufdiagramm Interruptprogramm "Kegeln"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) * 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Kegeln

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		LDX	I	20	CE	Lösche Anzeigeseite, Seite 4
4	4					00	
5	5					20	
6	6	Lösche	CLR	X	7F	6F	
7	7					7F	Anfangsadr. - 1 Seite 4
8	8		DEX			09	
9	9		BNE	R	Lösche	26	
10	A					FB	
11	B		LDX	I	80	CE	Setzen der Kegel in den ersten 5
12	C					00	Zeilen der Anzeigeseite
13	D					80	Bitmuster für 1. und 5. Reihe
14	E		STX	D	80	DF	
15	F					80	1. Zeile
16	10		STX	D	88	DF	
17	11					88	5. Zeile
18	12		LDX	I	140	CE	Bitmuster für 2. und 4. Reihe
19	13					01	
20	14					40	
21	15		STX	D	82	DF	
22	16					82	2. Zeile
23	17		STX	D	86	DF	
24	18					86	4. Zeile
25	19		LDX	I	240	CE	Bitmuster für mittlere Reihe
26	1A					02	
27	1B					A0	
28	1C		STX	D	84	DF	
29	1D					84	3. Zeile
30	1E		LDX	D	Kugel	DE	Setze Kugel in letzte Zeile
31	1F					7A	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) * 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Kegeln

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		STX	D	9E	DF	
1	1					9E	letzte Zeile Seite 4
2	2		CLI			0E	lösche Interruptmaske
3	3	Schleife	LDX	D	Zeit1	DE	Verzögerungsschleife.
4	4					7C	
5	5	V21	DEX			09	
6	6		BNE	R	V21	26	
7	7					FD	
8	8		LDAA	D	9E	96	Verschiebe Kugel nach rechts (im Kreis)
9	9					9E	ACCA : linkes Byte
10	A		LDAB	D	9F	D6	ACC B : rechtes Byte
11	B					9F	
12	C		LSRA			44	
13	D		RORB			56	Übertrag wird übernommen!
14	E		BCC	R		24	falls C gesetzt, ist das Bit im
15	F					02	linken Byte einzublenden
16	10		ORAA	I	80	8A	
17	11					80	
18	12		STAA	D	9E	97	
19	13					9E	
20	14		STAB	D	9F	D7	
21	15					9F	
22	16		JMP	E	Schleife	7E	
23	17					00	
24	18					23	
25	19	Interrupt	LDX	D	9E	DE	Kugel (letzte Zeile Seite 4) zwischen-
26	1A					9E	speichern
27	1B		STX	D	Maske	DF	
28	1C					78	
29	1D		LDX	I	9C	CE	Adresse vorletzte Zeile Seite 4 → X
30	1E					00	
31	1F					9C	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: Kegeln

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		CLR	X	2	6F	Lösche letzte Zeile
1	1					02	
2	2		CLR	X	3	6F	
3	3					03	
4	4	Kugel rollt	LDAA	X	0	A6	das durch den aktuellen Stand der
5	5					00	Kugel gegebene Bitmuster in aktueller
6	6		ORAA	D	Maske	9A	Zeile einblenden
7	7					78	
8	8		STAA	X	0	A7	
9	9					00	
10	A		LDAA	X	1	A6	
11	B					01	
12	C		ORAA	D	Maske + 1	9A	
13	D					79	
14	E		STAA	X	1	A7	
15	F					01	
16	10		STX	D	ZWIX	DF	Abspeicheradresse zwischenspeichern, da
17	11					76	X in Verzögerungsschleife benötigt wird
18	12		LDX	D	Zeit2	DE	Verzögerungsschleife
19	13					7E	
20	14	V22	DEX			09	
21	15		BNE	R	V22	26	
22	16					FD	
23	17		LDX	D	ZWIX	DE	aktuelle Adresse wieder laden
24	18					76	
25	19		LDAA	D	Maske	96	die durch die Kugel "getroffenen" Kegel
26	1A					78	ausblenden
27	1B		COMA			43	
28	1C		ANDA	X	0	A4	
29	1D					00	
30	1E		STAA	X	0	A7	
31	1F					00	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 3

Aufgabe: Kegeln

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		LDAA	D	Maske + 1	96	
1	1					79	
2	2		COMA			43	
3	3		ANDA	X	1	A4	
4	4					01	
5	5		STAA	X	1	A7	
6	6					01	
7	7		DEX			09	
8	8		DEX			09	
9	9		CPX	I	7E	8C	oberer Bildrand erreicht?
10	A					00	
11	B					7E	
12	C		BNE	R	Kugel rollt	26	nein, Kugel rollt weiter nach oben
13	D					D6	
14	E		LDX	D	Maske	DE	Kugel in letzter Zeile Seite 4 wieder
15	F					78	zurückspeichern
16	10		STX	D	9E	DF	
17	11					9E	
18	12		RTI			3B	Ende Interruptprogramm Kegeln
19	13						
20	14						
21	15						
22	16	ZWIX				00	Hilfszelle für aktuelle Adresse
23	17					00	
24	18	Maske				00	Hilfszelle für aktuellen Stand der Kugel
25	19					00	bei Interrupt
26	1A	Kugel				F8	Anfangsbitmuster für Kugel
27	1B					00	
28	1C	Zeit1				01	Verzögerungszeit für horizontale Kugelbe-
29	1D					10	wegung, kann variiert werden
30	1E	Zeit2				01	Verzögerungszeit für vertikale Kugelbe-
31	1F					20	wegung, kann variiert werden

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.6.2. Zahlenspiel REVERSE

REVERSE ist ein Zahlenspiel für einen Spieler. Das TV-Computersystem übernimmt hier nur eine Anzeigefunktion.

Gegeben seien die Ziffern 1 bis 9 in beliebiger Reihenfolge. Die Aufgabe besteht darin, die Zahlenfolge durch eine möglichst geringe Anzahl von Umordnungen in eine geordnete, aufsteigende Reihenfolge zu bringen. Ein Umordnungsschritt bringt die Ziffern eines Anfangsstücks der Liste in die umgekehrte Reihenfolge. "Reverse k" bedeutet die Umkehrung der ersten k Zahlen.

Beispiel:

Anfangsfolge:	9	6	2	3	5	4	1	8	7
Reverse 9 :	7	8	1	4	5	3	2	6	9|
" 2 :	8	7|	1	4	5	3	2	6	9
" 8 :	6	2	3	5	4	1	7	8|	9
" 6 :	1	4	5	3	2	6|	7	8	9
" 3 :	5	4	1|	3	2	6	7	8	9
" 5 :	2	3	1	4	5|	6	7	8	9
" 2 :	3	2|	1	4	5	6	7	8	9
" 3 :	1	2	3|	4	5	6	7	8	9

Folgende Überlegung zeigt, daß jede beliebige Liste geordnet werden kann. Eine sichere Strategie ist: Suche das größte Element, seine Position sei k. Mit Reverse k wird es an die erste Stelle, mit Reverse n an die letzte Stelle gebracht. Analog bringt man die nächstgrößere Zahl in zwei Zügen auf die Position (n-1) usw. Zum Ordnen einer Liste von n Zahlen benötigt dieser Algorithmus (2n-3) Züge.

Häufig kann man jedoch vorhandene Teilordnungen ausnutzen oder Teilordnungen geschickter erzeugen und somit in wesentlich weniger Zügen zum Ziel kommen. Suchen Sie Strategien!

Nach einigen Initialisierungsbefehlen läuft das Programm auf den Befehl WAI, Warten auf Interrupt. Der Spieler trägt die Positionsnummer für die nächste Umkehrung ein und startet durch Betätigen der Interrupttaste das Programmstück, das die Umordnung vornimmt. Gleichzeitig werden die Spielzüge gezählt.

Belegte Seiten: 0 - 4

Anzeigeseite: 3

In der linken Bytereihe ist vor dem Start eine beliebige Zahlenreihe einzugeben. Die Länge der Liste ist frei, maximal 16, d. h. die ganze Anzeigeseite.

Adresse 61: (oberste Zeile rechts): Eingabe Position für nächsten Umordnungsschritt.

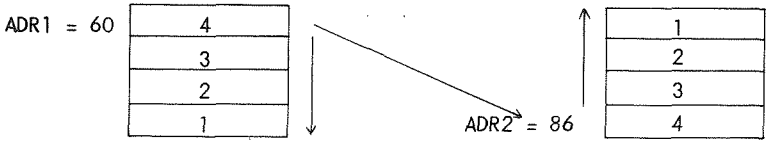
Adresse 63: Zähler für Anzahl der Spielzüge

Startadresse: 0

Interruptadresse: A

Das Umspeichern läuft über einen Hilfsbereich in Seite 4.

Beispiel für REVERSE 4:



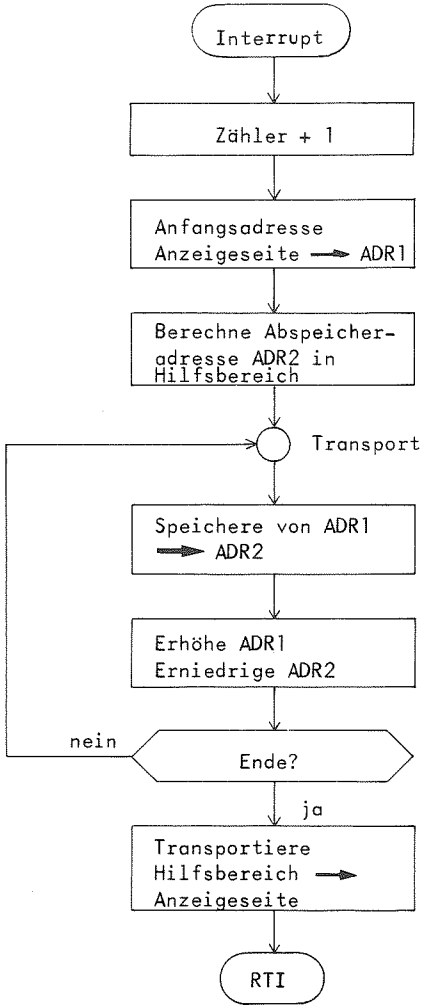


Bild 7.16 Ablaufdiagramm Interruptprogramm "REVERSE"

Franz MORAT KG
Elektro Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) * 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: REVERSE

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		CLI			0E	Lösche Interruptmaske
4	4		CLR	E	63	7F	Lösche Zähler
5	5					00	
6	6					63	
7	7	Warten	WAI			3E	Warteschleife, Interrupt startet
8	8		BRA	R	Warten	20	Umordnung der Zahlenreihe
9	9					FD	
10	A	Interrupt	INC	E	63	7C	Zähler + 1
11	B					00	
12	C					63	
13	D		LDX	I	60	CE	Anfangsadr. Seite 3 → ADR1
14	E					00	
15	F					60	
16	10		STX	D	ADR1	DF	
17	11					40	
18	12		LDAA	D	61	96	ACCA = REVERSE - Nr.
19	13					61	
20	14		BNE	R	UmSp.	26	= 0, kein Umspeichern
21	15					04	
22	16		RTI			3B	Ende Interruptprogramm
23	17	Umsp.	TAB			16	ACCB : REVERSE-Nr. als Zähler für Umsp.
24	18		ASLA			48	Berechne Abspeicheradresse ADR2
25	19		ADDA	I	7E	8B	Nr*2 + Anfangsadr. -2 von Seite 4
26	1A					7E	ergibt ADR2
27	1B					37	
28	1C		STAA	D	ADR2+1	43	
29	1D	Transport	LDX	D	ADR1	DE	Umspeichern von ADR1 → ADR2
30	1E					40	
31	1F		NOP			02	

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: REVERSE

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		LDAA	X	0	A6	
1	1					00	
2	2		INX			08	ADR1 +2
3	3		INX			08	
4	4		STX	D	ADR1	DF	
5	5					40	
6	6		LDX	D	ADR2	DE	
7	7					42	
8	8		STAA	X	0	A7	
9	9					00	
10	A		DEX			09	ADR2 - 2
11	B		DEX			09	
12	C		STX	D	ADR2	DF	
13	D					42	
14	E		DECB			5A	Zähler - 1
15	F		BNE	R	Transport	26	≠ 0, Fortsetzung Umspeichern
16	10					EC	
17	11		LDX	I	0	CE	= 0, Transportiere umgeordnete Reihe
18	12					00	von Seite 4 → Seite 3 zurück
19	13					00	
20	14	Rücktransp	LDAA	X	80	A6	
21	15					80	Anfangsadr. Seite 4
22	16		STAA	X	60	A7	
23	17					60	Anfangsadr. Seite 3
24	18		INX			08	
25	19		INX			08	
26	1A		DEC	E	61	7A	REVERSE-Nr. wird auf 0 zurückgezählt
27	1B					00	
28	1C					61	
29	1D		BNE	R	Rücktransp	26	≠ 0, Rücktransport fortsetzen
30	1E					F5	
31	1F		RTI			3B	Ende Interruptprogramm REVERSE

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: REVERSE

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	ADR1				00	Hilfszelle für aktuelle Abholadresse Seite 3
1	1					00	
2	2	ADR2				00	Hilfszelle für aktuelle Abspeicheradresse Seite 4
3	3					00	
4	4						
5	5						
6	6						
7	7						
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.6.3. NIM-Spiel

Das Spiel:

*Gegeben sind mehrere Haufen mit einer beliebigen Zahl von Spiel*steinen. Zwei Spieler nehmen abwechselnd Steine weg nach den Regeln:

1. Es dürfen nur von einem Haufen Steine entfernt werden.
2. Es muß mindestens ein Stein und es dürfen höchstens alle Steine des gewählten Haufens genommen werden.
3. Sieger ist, wer den letzten Zug macht.

Aus den Regeln folgt, daß es immer einen Gewinner gibt, es gibt kein Unentschieden.

Analyse des Spiels:

Es gibt sogenannte Gewinnstellungen, die man auch sichere Positionen nennt. Hinterläßt ein Spieler nach seinem Zug eine solche Stellung, so gewinnt er sicher, wenn er auch in den folgen*den Zügen entsprechend spielt. Ein Spieler, der eine sichere* Position vorfindet, kann nie ebenfalls eine sichere Stellung hinterlassen. Er verliert also, *wenn der Gegenspieler keinen* Fehler macht.

Beispiel:	H1	H2	H3
Ausgangssituation sei	2	3	2
1. Spieler 1 nimmt 1 von H1	1	3	2
2. Spieler 2 nimmt 3 von H2	1	0	2
3. Spieler 1 nimmt 1 von H3	1	0	1
4. Spieler 2 nimmt 1 von H1	0	0	1
5. Spieler 1 gewinnt!			

Gewinnstellungen lassen sich leicht charakterisieren, wenn man die Anzahl der Steine je Haufen als Dualzahlen schreibt. Addiert man die Dualziffern jeder Stelle ohne Berücksichtigung eines Übertrags, so liegt eine Gewinnstellung vor, wenn die so gebildete Summe gleich Null ist. Ist eine Summe ungleich Null, so kann sie immer durch Wegnehmen der geeigneten Zahl von Steinen zu Null gemacht werden.

Im obigen Beispiel war die Anfangssituation

0010	(2)
0011	(3)
0010	(2)
0011	(Summe ohne Überträge)

Durch Wegnehmen von einem Stein von H1 ergibt sich:

0001
0011
0010
0000

Spieler 1 hinterläßt eine Gewinnstellung, sein Sieg ist damit vorprogrammiert. Es hätte auch andere Züge gegeben, die zur Summe Null führen, z. B. Wegnehmen aller 3 Steine von H2.

Es ist zu überlegen, nach welchem Kriterium bei der Gewinn*strategie der Haufen und die Zahl der wegzunehmenden Steine* ermittelt wird. Man bestimmt in der Summe das höchstwertige Bit das mit 1 belegt ist. Da die Summe Null werden soll, muß ein Haufen gewählt werden, in dessen Anzahl dieses Bit ebenfalls gesetzt ist. Im Beispiel ist das bei allen 3 Haufen der Fall, allgemein mindestens bei einem. Wir wählten ohne weitere Überlegung den ersten Haufen.

Wieviele Steine wegzunehmen sind probieren wir aus, indem wir zunächst alle Steine wegnehmen und die Summe neu berechnen. Ist sie nicht Null, so addieren wir solange wieder 1 hinzu bis die Summe Null wird.

Im nächsten Zug kann Spieler 2 - gleichgültig wie er zieht - keine sichere Position hinterlassen. Im 3. Zug verfolgt Spieler 1 weiter die Gewinnstrategie.

Aus	0001	wird	0001
	0000		0000
	0010		0001
	0010		0000

Damit hat Spieler 2 keine Chance mehr auf den Gewinn. Betrachten wir noch die Anfangssituation 1, 2, 3. Sie ist eine sichere Position, da

0001
0010
0011
0000

Kennt Spieler 2 die Gewinnstrategie, so hat hier Spieler 1 keine Chance. Es hängt also von der Anfangssituation ab, ob es günstiger ist das Spiel zu beginnen oder nicht.

Das Programm

Es war das Ziel, ein möglichst kurzes Programm zu schreiben in dem der Rechner gegen den Menschen NIM spielt. Wir schränkten die allgemeine Form ein auf: 3 Haufen mit je maximal 9 Steinen in der Anfangsstellung (genauer: maximal 15 Steine bei hexadezimaler Zifferanzeige). Das Programm verfolgt, wenn möglich, die Gewinnstrategie, andernfalls wird in einem "Verlegenheitszug" vom ersten nicht leeren Haufen ein Stein weggenommen.

Es gibt keine Sicherheitsvorkehrungen gegen Mogeln. Der Spieler gibt seinen Zug ein, indem er die neue Anzahl Steine an der betreffenden Stelle einträgt. Der Rechner wartet nach einigen Initialisierungsbefehlen auf einen Interrupt. Das Betätigen der Interrupttaste startet das Spielprogramm, der Rechner führt seinen Zug aus und kehrt wieder in die Warteposition zurück.

Belegte Seiten:	0 - 3
Anzeigeseite:	3
	Die ersten 3 Bytes auf Seite 3 (Adressen 60, 61, 62) nehmen die Zahlen, die die Spielsteine repräsentieren, auf.
Startadresse:	0
Interruptadresse:	7

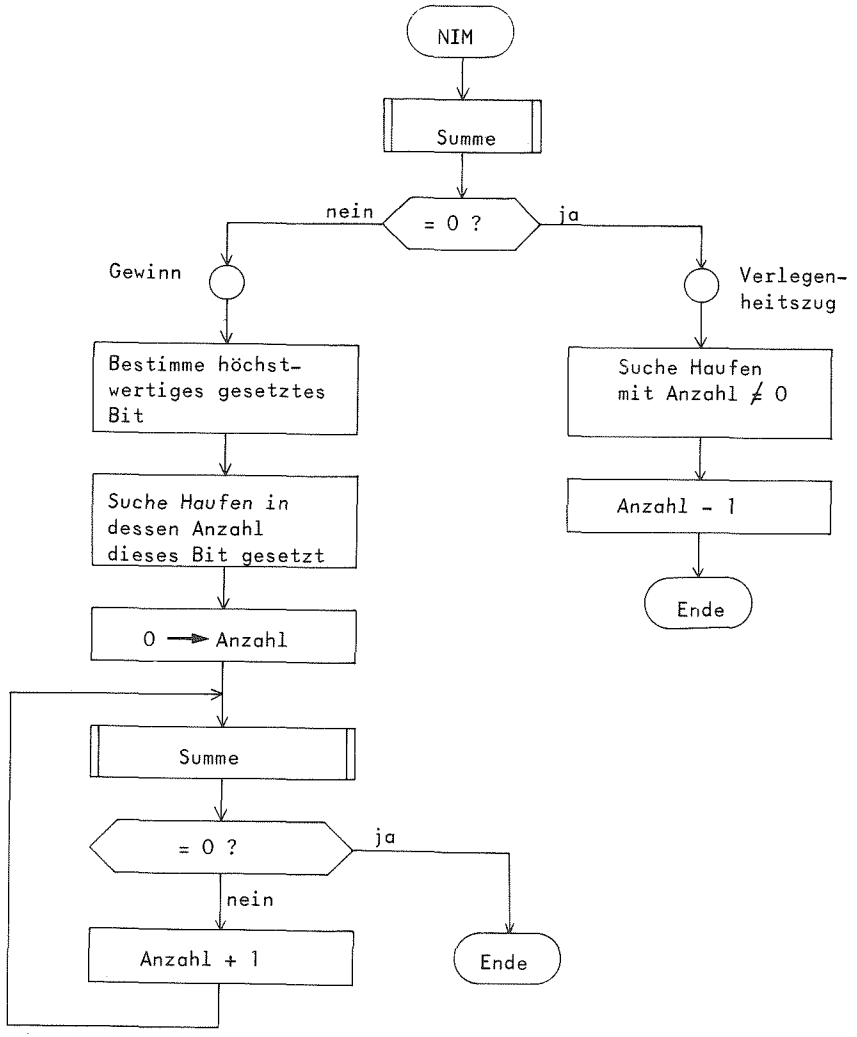


Bild 7.17 Ablaufdiagramm Programm "NIM"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07 722 323 fmkg d

TV-Computersystem 6800

Aufgabe: NIM Seite: 0

Nr	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		CLI			0E	Lösche Interruptmaske
4	4	Warte	WAI			3E	Warte schleife
5	5		BRA		Warte	20	
6	6		JSR	E	Summe	FD	das Interruptprogramm ist das Spielprog.
7	7	Interrupt				BD	Spaltensumme der vorliegenden Steine
8	8					00	ermitteln
9	9		BNE	R	Gewinn	43	≠ 0 , Gewinnstrategie verfolgen
10	A					26	
11	B		JMP	E	Verlegenh.	03	= 0 , Verlegenheitszug ausführen
12	C					7E	
13	D					00	
14	E		LDAB	I	8	38	
15	F	M1				C6	
16	10		STAB	D	Maske	08	bestimme höchstes Bit der Summe, das
17	11					D7	gesetzt ist
18	12		TAB			52	Aufgangsmarke sehen.
19	13		ANDA	D	Maske	16	
20	14	M2				94	Summe in AccA nach AccB retten
21	15					52	
22	16		BNE	R	M3	26	≠ 0 Bit gesetzt
23	17					08	
24	18		LSR	E	Maske	74	nächste Stelle abfragen
25	19					00	
26	1A					52	
27	1B		TBA			17	
28	1C		BRA	R	M2	20	
29	1D					F6	
30	1E		NOP			02	
31	1F		NOP			02	

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

A = Adressenmodus
CODE = interner Code

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: NIM

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	M3	LDX	I		CE	bestimme Haufen aus dem Steine entfernt
1	1					00	werden
2	2					60	Adr. Haufen 1
3	3		TAB			16	Maske höchstes Bit in ACCB retten
4	4	M4	ANDA	X	0	A4	
5	5					00	
6	6		BNE	R	M5	26	≠0, gefunden
7	7					04	
8	8		INX			08	=0, Vergleich fortsetzen
9	9		TBA			17	
10	A		BRA	R	M4	20	
11	B					F8	
12	C	M5	CLR	X	0	6F	X = Adr. des Haufens aus dem Steine
13	D					00	entfernt werden, setze Anzahl = 0
14	E	M6	JSR	E	Summe	BD	und prüfe Summe
15	F					00	
16	10					49	
17	11		BEQ	R	Ende	27	=0, Gewinnstellung erreicht, Ende
18	12					04	
19	13		INC	X	0	6C	≠0, Anzahl +1
20	14					00	
21	15		BRA	R	M6	20	erneuter Test ob Summe = 0
22	16					F7	
23	17	Ende	RTI			3B	Ende Interruptprogramm
24	18	Verlegenh	LDX	I	60	CE	Suche Haufen ≠ 0 Steine und entferne
25	19					00	einen Stein
26	1A					60	
27	1B	V1	TST	X	0	6D	
28	1C					00	
29	1D		BNE	R	V2	26	
30	1E					07	
31	1F		INX			08	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: NIM

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		CPX	I	63	8C	alle Haufen leer ?
1	1					00	
2	2					63	
3	3		BNE	R	V1	26	
4	4					F6	
5	5		RTI			3B	Rücksprung, kein Zug möglich, alles 0
6	6	V2	DEC	X	0	6A	Anzahl - 1
7	7					00	
8	8		RTI			3B	Rücksprung, 1 Stein entfernt
9	9	Summe	LDAA	D	60	96	die Summe ohne Berücksichtigung von
10	A					60	Überträgen ist durch Exklusives Oder
11	B		EORA	D	61	98	realisierbar
12	C					61	
13	D		EORA	D	62	98	
14	E					62	
15	F		RTS			39	Ende UP Summe
16	10						
17	11						
18	12	Maske				00	Hilfszelle
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

7.6.4. Lebensspiel (LIFE)

Das Spiel:

Conway, ein Mathematiker, entwickelte das Spiel, um auf einfache Weise Lebensvorgänge zu simulieren. Es wurde in /8/ erstmals beschrieben.

Auf der Spielseite sind beliebige Muster vorgebbar. Jedes gesetzte Feld gilt dabei als Lebewesen. Das Spiel besteht im Wechsel der Generationen, wobei das Muster sich bei einem Generationswechsel nach folgenden Regeln ändert:

- Es überlebt, wer auf den acht Nachbarfeldern zwei oder drei Nachbarn hat.
- Es stirbt, wer vier oder mehr Nachbarn hat (an Überbevölkerung) und wer nur einen oder keinen Nachbarn hat (an Vereinsamung).
- Jedes leere Feld mit genau drei Nachbarn wird neu besetzt (Geburt).

Eine neue Generation wird gleichzeitig für das gesamte Spielfeld ermittelt. Auf diese zweite Generation werden im nächsten Spielzug wieder die obigen Regeln angewendet usw. Die auf diese Weise entstehenden Muster sind schwer vorauszusagen und entwickeln mitunter ein reizvolles Eigenleben.

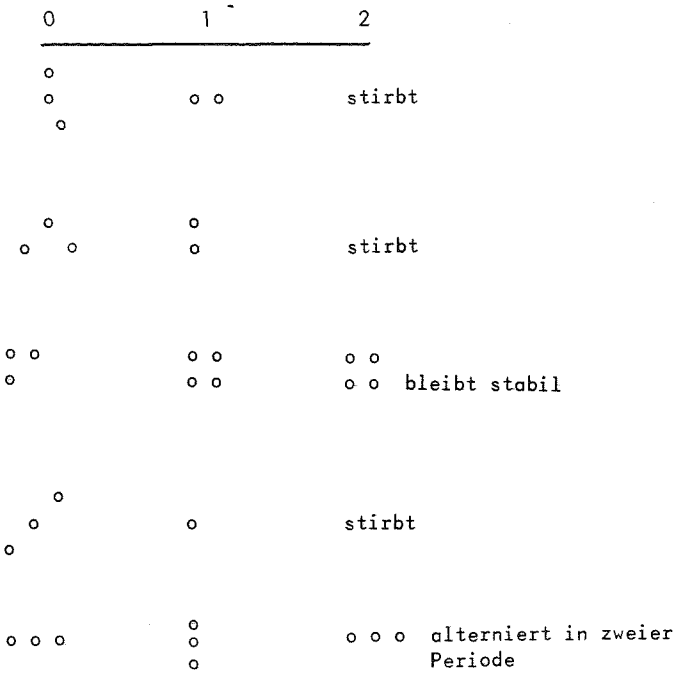
Conway wählte die Regeln nach folgenden Forderungen:

- Es sollte kein Ausgangsmuster geben, für das es einen einfachen Beweis gibt, daß die Bevölkerung unbegrenzt wächst.

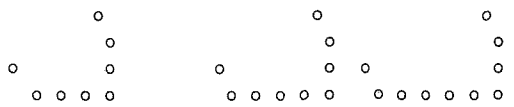
- Es sollte unbegrenzt wachsende Ausgangsmuster geben.
- Es sollte einfache Ausgangsmuster geben, die nach einer Wachstumsperiode wieder aussterben oder in festbleibende, stabile bzw. oszillierende Phasen eintreten.

Unabhängig von der Größe des Spielfeldes kann früher oder später das Randfeld als Störgröße in Erscheinung treten. Es verhindert weitere Ausbreitung und zerstört Symmetrien. Im folgenden werden einige typische Beispiele beschrieben.

1. Das Schicksal von 5 Tripeln

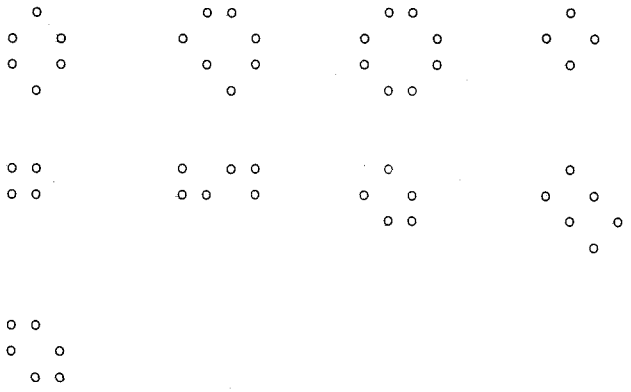


2. Die Figuren bewegen sich über die Spielseite

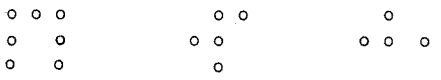


Raumschiffe verschiedener Größen

3. Folgende Figuren bleiben stabil



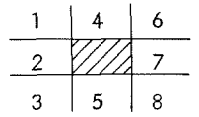
4. Figuren mit längeren Entwicklungsphasen sind:



Suchen Sie weitere Muster!

Das Programm

Auf der Anzeigeseite stehen 14 x 14 Felder zur Verfügung, die das Programm nach den Spielregeln in jeder Generation setzt bzw. löscht. Die Randfelder werden nur abgefragt, selbst aber nicht gesetzt. Da die Ermittlung der nächsten Generation für alle Felder gleichzeitig ablaufen muß, speichert das Programm die neue Belegung zunächst in einem Hilfsbereich (Seite 6) ab, der nach jedem Durchlauf in die Anzeigeseite übertragen wird.



Die Abfrage der acht Nachbarfelder ist durch Ausblenden mit geeigneten Masken realisiert. Die Wortgrenze in der Mitte der Zeile erschwert die Umgebungsabfrage und erfordert eine Sonderbehandlung. Neben den Ablaufdiagrammen für das Hauptprogramm und das wichtige Unterprogramm "Generation" seien die Aufgaben der Unterprogramme hier nochmals kurz beschrieben:

UP Generation:

verarbeitet Bit 6 bis Bit 1 eines Bytes vollständig. Von Bit 0 zählt es noch die Nachbarn 1 - 5, dann erfolgt der Rücksprung. Die Sonderbehandlung an der Wortgrenze erledigt das Hauptprogramm. Ein zweiter Einsprung "Gen. Fortsetzung" setzt die Verarbeitung von Bit 7 des rechten Bytes mit der Zählung der Nachbarfelder 4 - 8 und anschließend mit der Bearbeitung von Bit 6 bis Bit 1 fort. Die Nachbarfelder 1 - 3 von Bit 7 wurden in der Sonderbehandlung im Hauptprogramm bereits gezählt.

UP Erzeuge Masken:

erhält in ACCB als Parameter die Maske des zu verarbeitenden Feldes. Es erzeugt die Masken für den linken und rechten Nachbarn und löscht noch den Zähler.

UP U-Abfrage 3:

erhält in ACCB eine Maske. Es zählt die Nachbarn in den drei durch die Maske bezeichneten übereinanderliegenden Feldern. Der Index, der auf dem aktuellen Wort steht, wird vorübergehend erniedrigt bzw. erhöht. Dieses UP dient zur Abfrage der Felder 1 - 3 bzw. 6 - 8.

UP U-Abfrage 2:

ist ein Teil von U-Abfrage 3. Es zählt nur die darüber- und darunterliegenden Nachbarn und dient zur Abfrage von Feld 4 und 5.

UP Zähle:

erhält in ACCB eine Maske, in X steht die aktuelle Adresse. Es erhöht den Zähler, falls das durch die Maske bezeichnete Bit gesetzt ist.

UP Setze:

fragt den Zähler ab und setzt das bearbeitete Feld nach den Spielregeln.

Der Rechenzeitbedarf für die Ermittlung einer neuen Generation ist so groß, daß keine Verzögerungsschleife nötig ist. Die Generationen wechseln ca. alle 1 - 2 Sekunden. Durch eine kleine Modifikation des Programms - Befehl WAI in Adresse 3C anstelle von NOP - ist das Erzeugen einer neuen Generation auch per Interrupt steuerbar.

Belegte Seiten:	0 - 7
Anzeigeseite:	7
Startadresse:	0
(Interruptadresse:	3F6, mit 02 3B unter dieser Adresse)

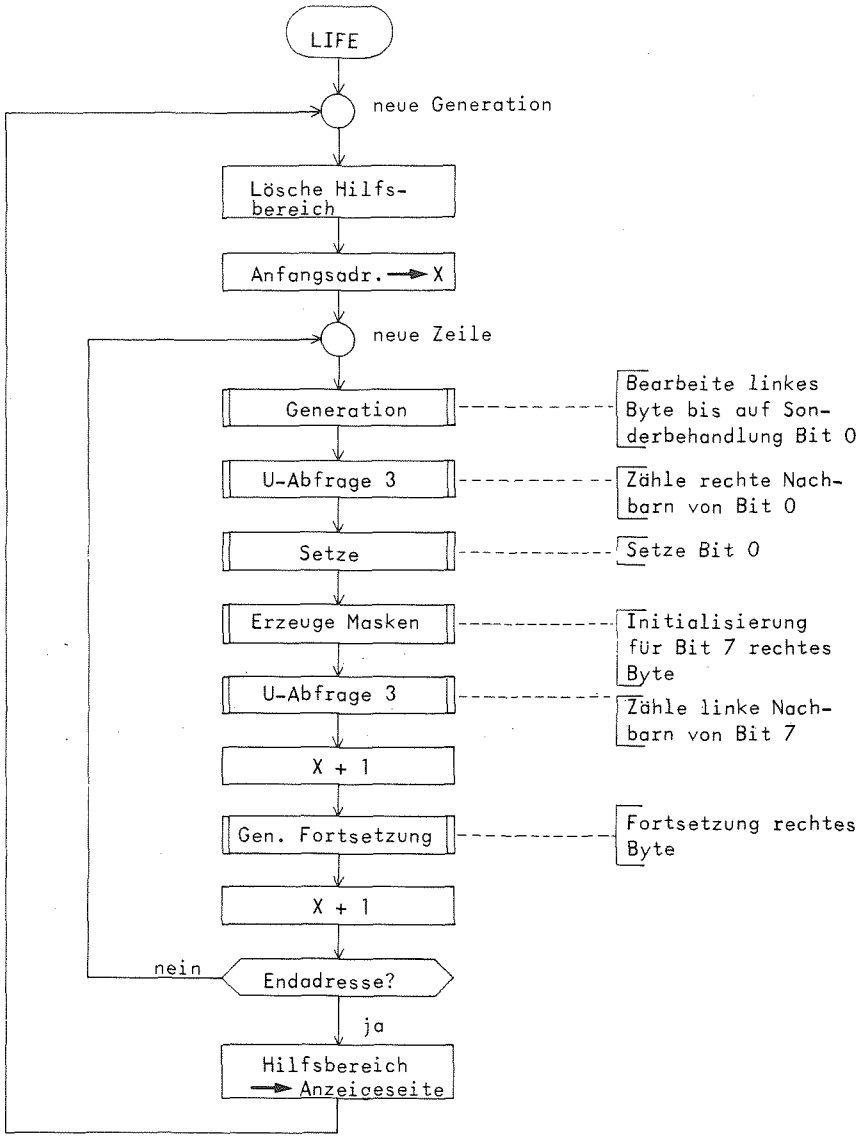


Bild 7.18 Ablaufdiagramm Programm "Lebensspiel"

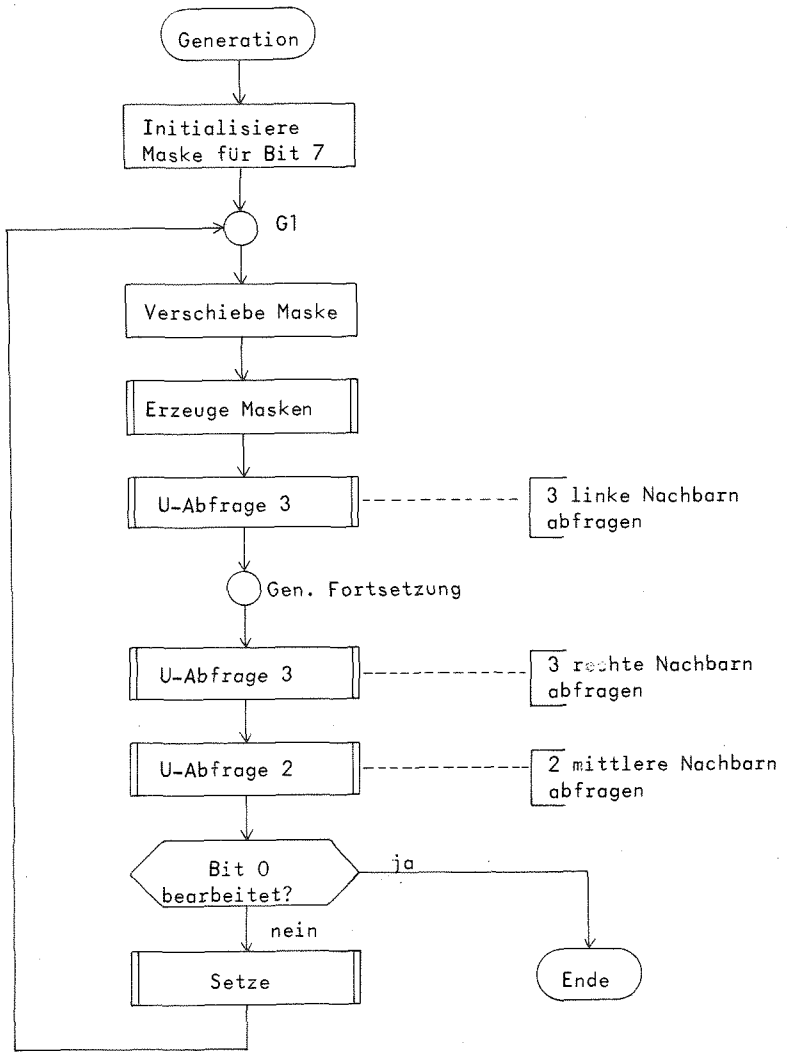


Bild 7.19 Ablaufdiagramm Unterprogramm "Generation"

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 0

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Start	LDS	I	3F5	8E	Stack liegt am Ende des Speichers
1	1					03	
2	2					F5	
3	3		CLI			0E	Lösche Interruptmaske
4	4	neue Gen.	LDX	I	20	CE	Lösche Hilfsbereich (Seite 6)
5	5					00	
6	6					20	
7	7	Lösche	CLR	X	BF	6F	
8	8					BF	Anfangsadr. - 1 Seite 6
9	9		DEX			09	
10	A		BNE	R	Lösche	26	
11	B					FB	
12	C		LDX	I	C2	CE	Anfangsadr. → X
13	D					00	
14	E					C2	Adresse 2. Zeile Seite 6
15	F	neue Zeile	JSR	E	Generation	BD	Bearbeite Bit 6 - Bit 0 im linken Byte
16	10					00	einer Zeile ohne rechte Nachbarn von
17	11					40	Bit 0
18	12		INX			08	rechte 3 Nachbarn von Bit 0 abfragen
19	13		LDAB	I	80	C6	ACC B: Maske
20	14					80	
21	15		JSR	E	U-Abfrage 3	BD	
22	16					00	
23	17					70	
24	18		DEX			09	
25	19		JSR	E	Setze	BD	Setze Bit 0
26	1A					00	
27	1B					8B	
28	1C		LDAB	I	80	C6	Maske für Bit 7 rechtes Byte
29	1D					80	
30	1E		NOP			02	
31	1F		NOP			02	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375 TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 1

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		JSR	E	Erzeuge M.	BD	
1	1					00	
2	2					62	
3	3		LDAB	I	01	C6	Maske für linken Nachbarn Bit 7
4	4					01	
5	5		JSR	E	U-Abfrage 3	BD	zähle die drei linken Nachbarn von Bit 7
6	6					00	
7	7					70	
8	8		INX			08	Fortsetzung im rechten Byte
9	9		JSR	E	Gen. Forts.	BD	
10	A					00	
11	B					4B	
12	C		INX			08	
13	D		CPX	I	DE	8C	Endadresse erreicht ?
14	E					00	
15	F					DE	Adr. letzte Zeile Seite 6
16	10		BNE	R	neue Zeile	26	nein, nächste Zeile bearbeiten
17	11					DD	
18	12		LDX	I	20	CE	Transportiere Seite 6 → Anzeigeseite
19	13					00	(Seite 7)
20	14					20	
21	15	Transp	LDAA	X	BF	A6	
22	16					BF	Anfangsadr. - 1 Seite 6
23	17		STAA	X	DF	A7	
24	18					DF	Anfangsadr. - 1 Seite 7
25	19		DEX			09	
26	1A		BNE	R	Transp	26	
27	1B					F9	
28	1C		NOP			02	hier kann WAI eingesetzt werden, falls
29	1D		JMP	E	neue Gen.	7E	neue Generation per Interrupt gewünscht
30	1E					00	
31	1F					04	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07 722323
fmkg d

TV-Computersystem 6800

Seite: 2

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0	Generation	LDAB	I	80	C6	
1	1					80	ACC B : Maske für Bit 7
2	2	G1	LSRB			54	Maske für aktuelles Bit
3	3		JSR	E	Erzeuge M	BD	leite hieraus Masken für rechten / linken
4	4					00	Nachbarn ab
5	5					62	
6	6		LDAB	D	Maske L	D6	die drei linken Nachbarn abfragen
7	7					A5	ACC B = Maske
8	8		JSR	E	U-Abfrage3	BD	
9	9					00	
10	A					70	
11	B	Gen. Forts.	LDAB	D	Maske R	D6	die drei rechten Nachbarn abfragen
12	C					A7	ACC B = Maske
13	D		JSR	E	U-Abfrage 3	BD	
14	E					00	
15	F					70	
16	10		LDAB	D	Maske M	D6	die zwei mittleren Nachbarn abfragen
17	11					A6	ACC B = Maske
18	12		JSR	E	U-Abfrage 2	BD	
19	13					00	
20	14					73	
21	15		CMPB	I	1	C1	Maske M = 1 ?
22	16					01	
23	17		BEQ	R	GE	27	ja, Bit 0 bearbeitet, Ende UP Generation
24	18					08	nein, Fortsetzung
25	19		JSR	E	Setze	BD	aktuelles Bit je nach Anzahl der Nach-
26	1A					00	barn Setzen / Löschen
27	1B					8B	
28	1C		LDAB	D	Maske M	D6	
29	1D					A6	
30	1E		JMP	E	G1	7E	
31	1F					00	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07 722323
fmkg d

TV-Computersystem 6800

Seite: 3

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0					42	
1	1	GE	RTS			39	Ende UP Generation
2	2	Erzeuge M	STAB	D	Maske M	D7	Erzeuge Masken
3	3					A6	ACC B = Maske MitHe
4	4		TBA			17	
5	5		ASLA			48	Maske Links
6	6		STAA	D	Maske L	97	
7	7					A5	
8	8		TBA			17	
9	9		LSRA			44	Maske rechts
10	A		STAA	D	Maske R	97	
11	B					A7	
12	C		CLR	E	Zähler	7F	Lösche Zähler
13	D					00	
14	E					A4	
15	F		RTS			39	Ende UP Erzeuge Masken
16	10	U-Abfrage 3	JSR	E	Zähle	BD	ACC B = Maske
17	11					00	Abfrage in aktueller Zeile
18	12					82	
19	13	U-Abfrage 2	DEX			09	Abfrage in darüber liegender Zeile
20	14		DEX			09	
21	15		JSR	E	Zähle	BD	
22	16					00	
23	17					82	
24	18		INX			08	Abfrage in darunterliegender Zeile
25	19		INX			08	
26	1A		INX			08	
27	1B		INX			08	
28	1C		JSR	E	Zähle	BD	
29	1D					00	
30	1E					82	
31	1F		DEX			09	alten Stand von X herstellen

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 4

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		DEX			09	
1	1		RTS			39	Ende UP U-Abfrage 3 / U-Abfrage 2
2	2	Zähle	TBA			17	ACCB: Maske, X = aktuelle Adresse
3	3		ANDA	X	20	A4	in Seite 6. Das betreffende Bit in Seite 7
4	4					20	ist abzufragen.
5	5		BEQ	R	ZE	27	
6	6					03	
7	7		INC	E	Zähler	7C	≠ 0, Erhöhe Anzahl der Nachbarn
8	8					00	
9	9					A4	
10	A	ZE	RTS			39	Ende UP Zähle
11	B	Setze	LDAA	D	Zähler	96	ACCA = Anzahl der Nachbarn
12	C					A4	
13	D		CMPA	I	2	81	
14	E					02	
15	F		BNE	R	S1	26	
16	10					06	
17	11		LDAA	X	20	A6	2 Nachbarn, das aktuelle Bit ist aus der
18	12					20	Anzeigeseite zu übernehmen
19	13		ANDA	D	Maske M	94	
20	14					A6	
21	15		BRA	R	S2	20	
22	16					06	
23	17	S1	CMPA	I	3	81	
24	18					03	
25	19		BNE	R	S3	26	Anzahl ≠ 2, 3, Feld bleibt gelöscht
26	1A					07	
27	1B		LDAA	D	Maske M	96	Anzahl = 3, Feld setzen
28	1C					A6	
29	1D	S2	ORAA	X	0	AA	Bit einblenden
30	1E					00	
31	1F		NOP			02	

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

0375-TV

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(0 76 57) * 444, 292

Fernschreiber
07 722 323
fmkg d

TV-Computersystem 6800

Seite: 5

Aufgabe: Lebensspiel

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		STAA	X	0	A7	
1	1					00	
2	2	S3	RTS			39	Ende UP Setze
3	3						
4	4	Zähler				00	Hilfszellen
5	5	Maske L				00	"
6	6	Maske M				00	"
7	7	Maske R				00	"
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

Seite = TV-Bild
Byte = Nr. auf Seite
OPEX = externer Code

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

8. Peripherie

Inhalt

8.1 Kassettenrekorderanschluß

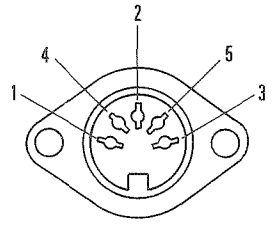
8.1.1. Einführung

Mit dem Kassettenanschluß bietet das TV-Computersystem die Möglichkeit der Ein- und Ausgabe über handelsübliche Kassettenrecorder. Die externe Speicherung von Programmen auf Kassetten vereinfacht das Arbeiten mit dem System erheblich. Die Kassetten-Ein- Ausgabe von beliebigen Speicherbereichen erspart die wiederholte Eingabe von Programmen bzw. Daten per Lichtgriffel.

In der Programmentwicklungsphase wird man den jeweils aktuellen Stand oder auch mehrere Versionen eines Programms auf einem Arbeitsband aufzeichnen. Ausgetestete Programme können auf einem Bibliotheksband zur Verfügung stehen.

8.1.1.1. Anschlußbedingungen

Das Aufzeichnungsverfahren stellt keine besonderen Anforderungen an die Kassettenrecorder. Als Anhaltspunkt sei erwähnt, daß das System mit einem preiswerten Gerät, das von der Stiftung Warentest in der Zeitschrift Test 7/76 mit "zufriedenstellend" bewertet wurde, einwandfrei funktioniert.



Mab5S

pin 2: Masse
pin 1: Eingang
pin 3: Ausgang

Bild 8.1.1.1.

8.1.1.2. Arbeitsweise der Kassetten-E/A

Die Ein- und Ausgabe wird per Mikroprozessor durch zwei in einem Festwertspeicher - 1/2 K PROM im Adressbereich C000 bis C1FF - gespeicherte Programme gesteuert. Alle variablen Daten dieser Programme liegen auf Seite 1F im Adressbereich von 3E0 bis 3F5. Es sind dies vom Benutzer bereitzustellende Parameter, Hilfszellen der Programme und der Stack. Der übrige Speicher ist für den Benutzer frei verfügbar.

Das Programm darf während des Ausgabe- bzw. Eingabevorgangs nicht unterbrochen werden, da das Erzeugen bzw. Erkennen der verschiedenen Impulsfrequenzen zeitkritische Vorgänge sind. Im Normalbetrieb rechnet der Prozessor pro Bildwiederholung nur in der Zeit zwischen unterem Bildende und Bildwechsel während der Bildschirmanzeige des Speichers wird er gestoppt. Aus den genannten Gründen muß also die Speicherabbildung auf dem Fernsehschirm während der Datenübertragung zum bzw. vom Kassettengerät unterbrochen werden. Der Bildschirm bleibt bis auf die Steuerfelder am rechten Rand in dieser Zeit dunkel.

Durch einen Start des Kassetten-Ausgabeprogramms wird ein beliebig langer, zusammenhängender Speicherbereich, ergänzt um einige organisatorische Daten, auf Band geschrieben. Ein Ausgabevorgang erzeugt auf dem Band einen variabel langen, zusammenhängenden Datenblock, der durch eine Identitätsnummer zwischen 00 und FF gekennzeichnet ist. Zum Erkennen von Übertragungsfehlern wird pro Byte ein Paritätsbit und pro Block eine Längsprüfsumme mit ausgegeben. Auf ein Band können nacheinander mehrere Blöcke geschrieben werden. Eine spezielle Synchronisation zum Erkennen eines Blockanfangs erlaubt es, zwischen Datenblöcken Sprache (oder Musik) aufzuzeichnen. Wir empfehlen, vor jedem Datenblock seine wichtigsten Angaben auf Band zu sprechen, so daß das Band gleichzeitig ein "verständliches" Inhaltsverzeichnis enthält.

Bei der Eingabe wird der durch die vorgegebene Identitätsnummer gekennzeichnete Datenblock auf Band gesucht und - falls vorhanden - in den Speicher eingelesen. Dabei muß das Band nicht unmittelbar vor dem gesuchten Datenblock stehen, da Blöcke mit ungleicher Identnummer (und Sprache) beim Suchvorgang überlesen werden. Das Eingabeprogramm nimmt eine Paritykontrolle vor und meldet gegebenenfalls einen Übertragungsfehler.

8.1.1.3. Angaben zur Zeitabschätzung

Vor jedem Datenblock wird ein Vorspann ausgegeben, der zur Synchronisierung und zur Identifizierung des Blocks dient. Für die Ausgabe des Vorspanns sind ca. 7,8 sec. zu veranschlagen. Die Ausgabe eines Bytes dauert ca. 75 msec. Damit ergeben sich folgende Übertragungszeiten:

1 TV-Seite	(32 Bytes)	ca. 10 sec
1/2 K	(512 Bytes)	ca. 46 sec
1 K	(1024 Bytes)	ca. 85 sec.

8.1.1.4. Erläuterung der Parameter

Identitätsnummer

Die Identitätsnummer ist im Bereich von 00 bis FF wählbar. Der Benutzer sollte darauf achten, daß eine Nummer pro Band nur einmal vergeben wird.

Anfangsadresse

Die Anfangsadresse (2 Bytes) ist bei der Ausgabe stets erforderlich, bei der Eingabe nur bei verschieblichem Format.

Endadresse + 1

Die um eins erhöhte Endadresse ist nur als Ausgabeparameter erforderlich.

Modus

Dieser Parameter unterscheidet die zwei Fälle:

Modus = 0: absolutes Programm

Modus ≠ 0: verschiebliches Programm.

Absolut bedeutet, daß ein Programm (z. B. durch Adressen) an einen festen Speicherbereich gebunden ist. Bei der Ausgabe eines absolut gekennzeichneten Speicherbereichs wird die Anfangsadresse mit auf Band geschrieben. Bei der Eingabe wird der Block in den ursprünglichen Bereich geladen, die Angabe einer Anfangsadresse als Parameter erübrigt sich.

Der Inhalt eines verschieblich gekennzeichneten Bereichs kann an jede beliebige Stelle eingelesen werden. Hierzu ist bei der Eingabe die Anfangsadresse als Parameter bereitzustellen.

Die Modusangabe wird vom Ein- und Ausgabeprogramm abgefragt. Dabei können widersprüchliche Angaben auftreten. Die vier möglichen Kombinationen sind:

Ausgabemodus	Eingabemodus	Bemerkung
absolut	absolut	-
absolut	verschieblich	siehe 1)
verschieblich	verschieblich	-
verschieblich	absolut	siehe 2)

1) In diesem Fall geben wir bei der Eingabe dem Modus verschieblich den Vorrang. Das heißt, die auf Band gespeicherte Anfangsadresse wird ignoriert, der Block wird ab der in den Zellen 3E2, 3E3 gespeicherten Anfangsadresse abgelegt.

2) Dieser Fall führt zu einer Fehlermeldung (Rückmeldung FE) und zum Abbruch des Eingabeprogramms. Da bei verschieblichem Ausgabemodus keine Adresse aufgezeichnet wird und beim absoluten Eingabemodus keine Adresse bereitzustellen ist, kann der Block nicht geladen werden.

8.1.2. Bedienung des Kassettenausgabeprogramms

Die Startadresse des Ausgabeprogramms ist C000. Das Programm erwartet vier Parameter:

- Identitätsnummer
- Anfangsadresse
- Endadresse + 1
- Modusangabe

Adresse		
3E0		Identnr.
3E2	Anfangsadresse	
3E4	Endadresse + 1	
3E6	Modus	
.		
.		
.		
3FE	Startadresse	

Bild 8.1.2.1. Parameter des Kassettenausgabeprogramms auf Seite 1F

Bei der Ausgabe eines Speicherbereichs ist in folgenden Schritten vorzugehen:

- Eintragen der Ausgabeparameter auf Seite 1F (siehe Bild 8.1.2.1.)
  - Adresse 3E1: Identitätsnummer
  - Adresse 3E2, 3E3: Anfangsadresse (höherwertiges Byte in 3E2, niederwertiges Byte in 3E3)
  - Adresse 3E4, 3E5: Endadresse + 1
  - Adresse 3E6: Modus

- Eintragen der Startadresse C000 in der letzten Zeile von Seite 1F.

- Vorbereiten und Anschließen des Kassettengeräts, Band zurückspulen oder nach letztem Datenblock positionieren, Lautstärkeregler etwa in mittlerer Position einstellen.

- Start des Kassettengeräts im Aufnahmemodus

- Nach einigen Sekunden Start des Mikroprozessors

Unmittelbar nach dem Start des Ausgabeprogramms wird, wie bereits erwähnt, die Speicheranzeige dunkel. Das Wiederaufleuchten des Bildschirms zeigt das Ende des Übertragungsvorgangs an.

- Nach einigen Sekunden Band stoppen.
- Mikroprozessor stoppen.

Für jedes Band sollte ein Inhaltsverzeichnis geführt und bei jeder Ausgabe auf den aktuellen Stand gebracht werden.

Beispiel:

Ein absolutes Programm, das die Seiten 1 bis 5 belegt, soll unter der Identitätsnummer 33 ausgegeben werden. Bild 8.1.2.2. zeigt die Parameter auf Seite 1F. Man beachte vor allem, daß die um eins erhöhte Endadresse einzutragen ist.

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1
Fernruf Eisenbach (07657) 444, 292
Fernschreiber 07722323 fmkg d

TV-Computersystem 6800

Aufgabe: Beispiel: Kassettenausgabe Seite: 1F

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	3E0					00	Indentitätsnummer
1	1					33	
2	2					00	Anfangsadresse des auszugebenden
3	3					20	Programms
4	4					00	Endadresse +1 des auszugebenden
5	5					C0	Programms
6	6					00	Modus: absolutes Programm
7	7						
8	8						
9	9						
10	A						
11	B						
12	C						
13	D						
14	E						
15	F						
16	3F0						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E					C0	Startadresse des Kassetten-
31	1F					00	ausgabeprogramms

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Bild 8.1.2.2.

0375-TV

8.1.3. Bedienung des Kassetteneingabeprogramms

Die Startadresse des Eingabeprogramms ist C100. Parameter sind:

- Indentitätsnummer
- Modus
- Anfangsadresse, falls Modus = verschieblich.

Das Programm liefert eine Rückmeldung unter Adresse 3E0.

Adresse	
3E0	Rückm. | Identnr.
3E2	(Anfangsadresse)
3E4	
3E6	Modus |
.	
.	
.	
3FE	Startadresse

Bild 8.1.3.2. Parameter des Kassetten-Eingabeprogramms

Bei der Kassetten-Eingabe ist in folgenden Schritten vorzugehen:

- Eintragen der Eingabeparameter auf Seite 1F (siehe Bild 8.1.3.1.)
  - Adresse 3E1: Identitätsnummer
  - Adresse 3E6: Modus (=0 bedeutet absolut, ≠0 verschieblich)
  - falls Modus ≠0:
    Adresse 3E2, 3E3: Anfangsadresse.
- Eintragen der Startadresse C100 unter letzten Zeile von Seite 1F.
- Vorbereiten und Anschließen des Kassettengerätes. Das Band entweder an den Anfang zurückspulen und Block suchen lassen oder

Band vor dem betreffenden Block anhalten. Lautstärkeregler in mittlerer Position einstellen.

- Start des Mikroprozessors

Start des Kassettenrekorders in Wiedergabemodus.

Wie bei der Ausgabe ist der Bildschirm auch während des Eingabevorgangs dunkel. Das Wiederaufleuchten des Bildschirms zeigt das Ende der Übertragung an. Wird die vorgegebene Identitätsnummer nicht gefunden, läuft das Band immer weiter und auch der Bildschirm bleibt dunkel. Um diese Situation zu erkennen, ist das Abschätzen der Eingabezeit nützlich.

- Band und Mikroprozessor stoppen.

Das Eingabeprogramm liefert im Byte 3E0 eine Rückmeldung, die drei verschiedene Werte annehmen kann:

Rückmeldung = 00: Gesuchter Datenblock wurde fehlerfrei in den Speicher eingelesen.

" = FF: Es trat ein Übertragungsfehler (Parityfehler) auf, die Übertragung wurde an der betreffenden Stelle abgebrochen und damit nicht ordnungsgemäß abgeschlossen. Die Eingabe sollte nochmals versucht werden.

" = FE: Benutzerfehler durch widersprüchliche Modusangaben bei Ein- und Ausgabe, siehe 8.1.1.4.

8.2. BUS-Anschluß

Die Signale am BUS haben TTL-Pegel. Die Adressenleistungen sind intern über 7417 gepuffert. Der Datenbus sollte extern über LS-TTL gepuffert werden. Der Takt Ø2 ist ebenfalls, ohne längere Verbindung, zu puffern. Wird eine Adresse außerhalb des internen 1 k Byte-Speichers vom Mikroprozessor angewählt, schaltet das Gerät den internen Speicher ab.

Das Signal gibt die Zeit an, in der der Mikroprozessor auf den Speicher des TV-Systems zugreifen kann. Dies sind alle 18 µsec. je 2 µsec. Während der 18 µsec. greift die Anzeigeelektronik auf dem Speicher zu. In dieser Zeit befindet sich der 6800 im Haltzustand.

8.3 Relaisanschluß

Über die Anschlüsse 4 und 5 der 5-poligen Tonabnehmerbuchse (Mab5S) ist jeweils ein Relais per Programm ansteuerbar. Wir empfehlen und liefern R-Relais der Fa. National (165 Ohm) mit einem Umschaltkontakt. Die Wicklung wird unmittelbar zwischen die Anschlußstifte und + 5V geschaltet.

Die Relais sind über das Datenwort FF00 Bit 0 und/oder Bit 1 anzusprechen. Über Bit 2 läßt sich die Speicherausgabe unterdrükken. Bit 7 enthält das Datenbit der Kassetten-Ein- und Ausgabe.

Das Relais Bit 0 wird durch das Kassettenprogramm angesteuert und ist daher bei der Kassetten-Ein- und Ausgabe nicht frei verfügbar.

Franz MORAT KG
Elektro-Feinmechanik und Maschinenbau
7821 Eisenbach/Hochschwarzwald 1

Fernruf
Eisenbach
(07657) * 444, 292

Fernschreiber
07722323
fmkg d

TV-Computersystem 6800

Seite: 0

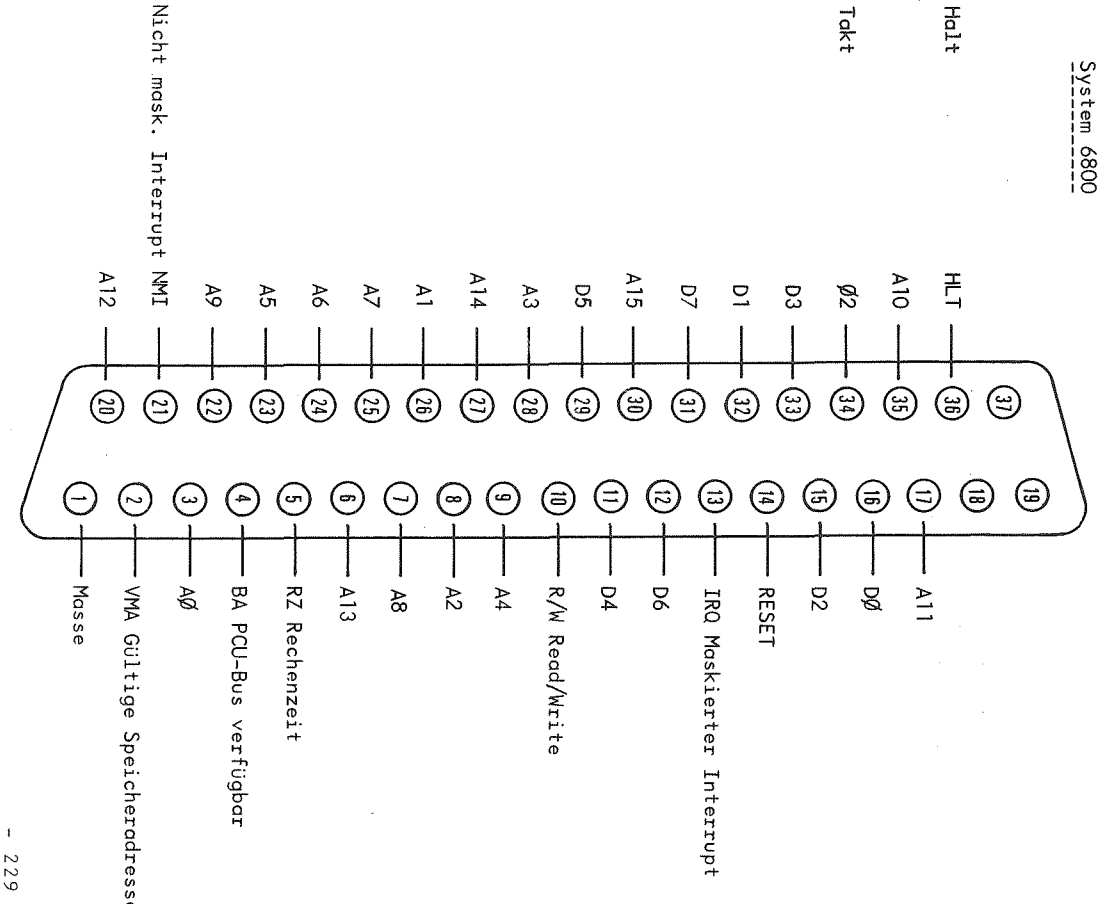
Aufgabe: Ansprechen der Relais-Anschlüsse I und II

Nr.	Byte	Marke	OPEX	A	Symb. Adr.	CODE	Kommentar
0	0		LDAA	I		86	
1	1					01	(02 bei R-Anschluß II)
2	2		STAA	E	FF00	B7	Ansprechen des Datenworts FF00
3	3					FF	
4	4					00	
5	5		BRA	R		20	Stop
6	6					FE	
7	7						
8	8						
9	9						
10	A	Wenn die Relais-Anschlüsse gelöscht werden sollen so ist der befehl LDAA durch CLRA					
11	B	zu ersetzen.(Der gleiche Effekt wird durch Ersetzen des Wertes 01 bzw. 02 in Byte 01					
12	C	durch den Wert 00 erhalten.)					
13	D						
14	E						
15	F						
16	10						
17	11						
18	12						
19	13						
20	14						
21	15						
22	16						
23	17						
24	18						
25	19						
26	1A						
27	1B						
28	1C						
29	1D						
30	1E						
31	1F						

A = Adressenmodus
CODE = interner Code

A: Immediate = I, Implied = M,
Extended = E, Relativ = R,
Index = X, Accu. = A oder B

Bild 8.3.



Literatur

/1/ Motorola: M 6800 Microprocessor Applications Manual. 1975.

/2/ Motorola: M 6800 Microprocessor Programming Manual. 1975.

/3/ Wettstein, H.: Systemprogrammierung. C. Hanser-Verlag München 1972.

/4/ Franke, K.: Das Programmieren von Kleinrechnern im Maschinencode. C. Hanser-Verlag München 1974.

/5/ People's Computer Company.: What to Do After You Hit Return or P.C.C. 's First Book of Computer Games. Menlo Park, California 1975.

/6/ DIN 44300: Informationsverarbeitung, Begriffe. Beuth-Verlag Berlin.

/7/ DIN 66001: Sinnbilder für Datenfluß- und Programmablaufpläne. Beuth-Verlag Berlin.

/8/ Gardner, M.: Mathematical Games. Scientific American. October, November 1970

ANHANG

Inhalt Seite

Anhang A: Alphabetische Befehlsübersicht

ABA	Addiere Akkumulatoren	(Add Accumulators)
ADC	Addiere mit Übertrag	(Add with Carry)
ADD	Addiere	(Add)
AND	Logisches UND	(Logical And)
ASL	Arithmetischer Links-Shift	(Arithmetic Shift Left)
ASR	Arithmetischer Rechts-Shift	(Arithmetic Shift Right)
BCC	Sprung wenn kein Übertrag	(Branch if Carry Clear)
BCS	Sprung wenn Übertrag	(Branch if Carry Set)
BEQ	Sprung wenn Null	(Branch if Equal to Zero)
BGE	Sprung wenn größer gleich Null	(Branch if Greater or Equal Zero)
BGT	Sprung wenn größer Null	(Branch if Greater than Zero)
BHI	Sprung wenn größer	(Branch if Higher)
BIT	Bit Test	(Bit Test)
BLE	Sprung wenn kleiner gleich	(Branch if Less or Equal)
BLS	Sprung wenn kleiner gleich	(Branch if Lower or Same)
BLT	Sprung wenn kleiner Null	(Branch if Less than Zero)
BMI	Sprung wenn Minus	(Branch if Minus)
BNE	Sprung wenn nicht Null	(Branch if Not Equal to Zero)
BPL	Sprung wenn Plus	(Branch if Plus)
BRA	Unbedingter Sprung (relativ)	(Branch Always)
BSR	Unterprogrammsprung(relativ)	(Branch to Subroutine)
BVC	Sprung wenn kein Überlauf	(Branch if Overflow Clear)
BVS	Sprung wenn Überlauf	(Branch if Overflow Set)
CBA	Vergleiche Akkumulatoren	(Compare Accumulators)
CLC	Lösche Übertrag	(Clear Carry)
CLI	Lösche Interrupt-Maske	(Clear Interrupt Mask)
CLR	Lösche	(Clear)
CLV	Lösche Überlauf	(Clear Overflow)
CMP	Vergleiche	(Compare)
COM	Logische Negation	(Complement)
CPX	Vergleiche Indexregister	(Compare Index Register)
DAA	Dezimalangleichung	(Decimal Adjust)
DEC	Vermindere (-1)	(Decrement)
DES	Vermindere Stack-Register (-1)	(Decrement Stack Pointer)
DEX	Vermindere Index-Register (-1)	(Decrement Index Register)
EOR	Exklusives ODER	(Exclusive Or)
INC	Erhöhe (+1)	(Increment)
INS	Erhöhe Stack-Register (+1)	(Increment Stack Pointer)
INX	Erhöhe Index-Register (+1)	(Increment Index Register)
JMP	Sprung	(Jump)
JSR	Unterprogrammsprung	(Jump to Subroutine)

LDA	Lade Akkumulator	(Load Accumulator)
LDS	Lade Stack-Zähler	(Load Stack Pointer)
LDX	Lade Index-Register	(Load Index Register)
LSR	Logischer Rechtsshift	(Logical Shift Right)
NEG	Arithmetische Negation	(Negate)
NOP	Keine Operation	(No Operation)
ORA	Inklusives ODER	(Inclusive OR Accumulator)
PSH	Transport von Akku. nach Stack	(Push Data)
PUL	Transport von Stack nach Akku.	(Pull Data)
ROL	Kreisshift links	(Rotate Left)
ROR	Kreisshift rechts	(Rotate Right)
RTI	Rücksprung von Interrupt	(Return from Interrupt)
RTS	Rücksprung von Unterprogramm	(Return from Subroutine)
SBA	Subtraktion Akkumulatoren	(Subtract Accumulators)
SBC	Subtraktion mit Übertrag	(Subtract with Carry)
SEC	Setze Übertrag	(Set Carry)
SEI	Setze Interrupt-Maske	(Set Interrupt Mask)
SEV	Setze Überlauf	(Set Overflow)
STA	Speichere Akkumulator	(Store Accumulator)
STS	Speichere Stack Register	(Store Stack Register)
STX	Speichere Indexregister	(Store Index Register)
SUB	Subtrahiere	(Subtract)
SWI	Software-Interrupt	(Software Interrupt)
TAB	Transport Akkumulatoren	(Transfer Accumulators)
TAP	Transport Akku. nach Bed.-Reg.	(Transf. Accu. to Cond.Reg.)
TBA	Transport Akkumulatoren	(Transfer Accumulators)
TPA	Transport Bed.-Reg. nach Akku.	(Transfer Cond. R. to Accu)
TST	Test	(Test)
TSX	Transport Stack-Register nach Index	(Transfer Stack to Index)
TXS	Transport Index nach Stack-Register	(Transfer Index to Stack)
WAI	Warten auf Interrupt	(Wait for Interrupt)

Anhang B: Befehlsliste

Bedeutung der verwendeten Zeichen und Abkürzungen

OP	Operationscode
B	Byte-Anzahl pro Befehl
ACCA, A	Akkumulator A
ACCB, B	Akkumulator B
ACCX	Akkumulator A oder B
SP	Stackregister
X	Indexregister
PC	Befehlszähler
H	höherwertiges Byte } als Zusatz zu SP, X oder PC
L	niederwertiges Byte } als Zusatz zu SP, X oder PC
M	Speicherwort
M_{SP}	Oberstes Speicherwort im Stack
CCR	Bedingungsregister (Condition-Codes-Register)
H	Halbübertrag von Bit 3 (für dezimales Rechnen)
I	Interruptmaske
N	Negativ
Z	Null
V	Überlauf
C	Übertrag von Bit 7
R	wird gelöscht (Reset)
S	wird gesetzt (Set)
/	wird gesetzt, wenn Bedingung erfüllt, andernfalls gelöscht
o	gleibt unverändert
00	Byte = Null
→	von - nach
·	Logisches UND
+	Logisches ODER
⊕	Exklusives ODER
-	Logische Negation

Sonderfälle im Bedingungsregister

1	(V)	Test:	Ergebnis = 10000000?
2	(C)	"	Ergebnis = 00000000?
3	(C)	"	höherwertiges BCD-Zeichen größer neun? (wird nicht gelöscht, wenn es vorher gesetzt war)
4	(V)	"	Operand = 10000000 vor der Ausführung? 80_{16}
5	(V)	"	Operand = 01111111 vor der Ausführung? $3F_{16}$
6	(V)	"	wird auf das Ergebnis von N ⊕ C nach Ausführung des Shifts gesetzt
7	(N)	"	Vorzeichenbit des höherwertigen Bytes des Ergebnisses = 1?
8	(V)	"	Überlauf des Zweierkomplements bei der Subtraktion des niederwertigen Bytes?
9	(N)	"	Ergebnis kleiner Null? (Bit 15 = 1)
10	(alle)	"	lädt Bedingungsregister vom Stack
11	(I)	"	wird durch Interrupt gesetzt. Falls vorher gesetzt, kann nur ein nichtmaskierter Interrupt den Wartezustand beenden.
12	(alle)	"	alle Bits werden entsprechend dem Inhalt von Akkumulator A gesetzt.

Transportbefehle

Ext. code	Bezeichnung	Wirkung	Immed OP	B	Direkt OP	B	Index OP	B	Extend OP	B	Inher. OP	B	Bedingungsreg. 5 4 3 2 1 0 H I N Z V C
LDAA	Lade Akku A	M→A	86	2	96	2	A6	2	B6	3			o o / / R o
LDAB	Lade Akku B	M→B	C6	2	D6	2	E6	2	F6	3			o o / / R o
LDS	Lade Stack	M→SPH, M+1→SPL	8E	3	9E	2	AE	2	BE	3			o o 9 / R o
LDX	Lade Index	M→XH, M+1→XL	CE	3	DE	2	EE	2	FE	3			o o 9 / R o
PSHA	von Akku A nach Stack	A→MSP, SP-1→SP									36	1	o o o o o o
PSHB	" " B " "	B→MSP, SP-1→SP									37	1	o o o o o o
PULA	vom Stack nach Akku A	SP+1→SP, MSP→A									32	1	o o o o o o
PULB	" " " " B	SP+1→SP, MSP→B									33	1	o o o o o o
STAA	Speichere aus Akku A	A→M			97	2	A7	2	B7	3			o o / / R o
STAB	" " " B	B→M			D7	2	E7	2	F7	3			o o / / R o
STS	" " Stack	SPH→M, SPL→(M+1)			9F	2	AF	2	BF	3			o o 9 / R o
STX	" " Index	XH→M, XL→(M+1)			DF	2	EF	2	FF	3			o o 9 / R o
TAB	Trans. Akku A nach B	A→B									16	1	o o / / R o
TAP	" Akku nach Bed.R.	A→CCR									06	1	----12----
TBA	" Akku B nach A	B→A									17	1	o o / / R o
TPA	" Bed. R. nach AkkuA	CCR→A									07	1	o o o o o o
TSX	" Stack nach Index	SP+1→X									30	1	o o o o o o
TXS	" Index nach Stack	X-1→SP									35	1	o o o o o o

Setz- Lösch- und Shift-Befehle

Ext. code	Bezeichnung	Wirkung	Immed. OP	B	Direkt OP	B	Index OP	B	Extend OP	B	Inher. OP	B	Bedingungsreg. 5 4 3 2 1 0 H I N Z V C
CLC	Lösche Übertrag	0→C									0C	1	o o o o o R
CLI	" Interrupt-Maske	0→I									0E	1	o R o o o o
CLV	" Überlauf	0→V									0A	1	o o o o R o
CLR	" Speicher	00→M					6F	2	7F	3			o o R S R R
CLRA	" Akku A	00→A									4F	1	o o R S R R
CLRB	" Akuu B	00→B									5F	1	o o R S R R
SEC	Setze Übertrag	1→C									0D	1	o o o o o S
SEI	" Interrupt-Maske	1→I									0F	1	o S o o o o
SEV	" Überlauf	1→V									0B	1	o o o o S o
ASL	Arith.Shift links Speicher	M					68	2	78	3			o o / / 6 /
ASLA	" " " Akku A	A									48	1	o o / / 6 /
ASLB	" " " " B	B									58	1	o o / / 6 /
ASR	" " rechts Speich.	M					67	2	77	3			o o / / 6 /
ASRA	" " " Akku A	A									47	1	o o / / 6 /
ASRB	" " " " B	B									57	1	o o / / 6 /
LSR	Logischer Shift rechts Speich.	M					64	2	74	3			o o R / 6 /
LSRA	" " " Akku A	A									44	1	o o R / 6 /
LSRB	" " " " B	B									54	1	o o R / 6 /
ROL	Kreisshift links, Speicher	M					69	2	79	3			o o / / 6 /
ROLA	" " Akku A	A									49	1	o o / / 6 /
ROLB	" " " B	B									59	1	o o / / 6 /
ROR	" rechts Speicher	M					66	2	76	3			o o / / 6 /
RORA	" " Akku A	A									46	1	o o / / 6 /
RORB	" " " B	B									56	1	o o / / 6 /

Arithmetische und Logische Befehle

Ext. code	Bezeichnung	Wirkung	Immed. OP	Immed. B	Direkt OP	Direkt B	Index OP	Index B	Extend. OP	Extend. B	Inher. OP	Inher. B	Bedingungsreg. 5 4 3 2 1 0 H I N Z V C
ABA	Addiere Akku A + B	A + B→A									1B	1	/ o / / / /
ADDA	" Sp.+Akku A	A + M→A	8B	2	9B	2	AB	2	BB	3			/ o / / / /
ADDB	" Sp.+Akku B	B + M→B	CB	2	DB	2	EB	2	FB	3			/ o / / / /
ADCA	" Sp.+Ü.+Akku A	A + M + C→A	89	2	99	2	A9	2	B9	3			/ o / / / /
ADCB	" Sp.+Ü+Akku B	B + M + C→B	C9	2	D9	2	E9	2	F9	3			/ o / / / /
SBA	Subtrahiere Akkus	A-B → A									10	1	o o / / / /
SUBA	" aus Akku A	A-M → A	80	2	90	2	A0	2	B0	3			o o / / / /
SUBB	" " " B	B-M → B	C0	2	D0	2	E0	2	F0	3			o o / / / /
SBCA	" mit Übertrag	A-M-C→A	82	2	92	2	A2	2	B2	3			o o / / / /
SBCB	" " "	B-M-C→B	C2	2	D2	2	E2	2	F2	3			o o / / / /
DAA	Dezimalangleichung										19	1	o o / / / 3
DEC	Vermindere Sp.-1	M-1→M					6A	2	7A	3			o o / / 4 o
DECA	" Akku A-1	A-1→A									4A	1	o o / / 4 o
DECB	" Akuu B-1	B-1→B									5A	1	o o / / 4 o
DEX	" Index-1	X-1→X									09	1	o o o / o o
DES	" Stack-1	Sp-1→SP									34	1	o o o o o o
INC	Erhöhe Sp. + 1	M+1→M					6C	2	7C	3			o o / / 5 o
INCA	" Akku A + 1	A+1→A									4C	1	o o / / 5 o
INCB	" Akku B + 1	B+1→B									5C	1	o o / / 5 o
INX	" Index + 1	X+1→X									08	1	o o o / o o
INS	" Stack + 1	SP+1→SP									31	1	o o o o o o

Ext. code	Bezeichnung	Wirkung	Immed. OP	Immed. B	Direkt OP	Direkt B	Index OP	Index B	Extend. OP	Extend. B	Inher. OP	Inher. B	Bedingungsreg. 5 4 3 2 1 0 H I N Z V C
ANDA	Logisches UND Sp+Akku A	A.M→A	84	2	94	2	A4	2	B4	3			o o / / R o
ANDB	" " " +Akku B	B.M→B	C4	2	D4	2	E4	2	F4	3			o o / / R o
BITA	Bit Test Sp+Akku A	A.M	85	2	95	2	A5	2	B5	3			o o / / R o
BITB	" " " Akku B	B.M	C5	2	D5	2	E5	2	F5	3			o o / / R o
COM	Logische Negation Sp.	$\overline{M}$→M					63	2	73	3			o o / / R S
COMA	" " Akku A	$\overline{A}$→A									43	1	o o / / R S
COMB	" " Akku B	$\overline{B}$→B									53	1	o o / / R S
EORA	Exklusives ODER Akku A	A⊕M→A	88	2	98	2	A8	2	B8	3			o o / / R o
EORB	" " Akku B	B⊕M→B	C8	2	D8	2	E8	2	F8	3			o o / / R o
NEG	Arithmetm. Negation Sp.	00-M→M					60	2	70	3			o o / / 1 2
NEGA	" " Akku A	00-A→A									40	1	o o / / 1 2
NEGB	" " Akku 0	00-B→B									50	1	o o / / 1 2
ORAA	ODER Sp. + Akku A	A+M→A	8A	2	9A	2	AA	2	BA	3			o o / / R o
ORAB	ODER Sp. + Akku B	B+M→B	CA	2	DA	2	EA	2	FA	3			o o / / R o

Vergleichs- und Sprungbefehle

Ext. code	Bezeichnung	Wirkung	Immed. OP	Immed. B	Direkt OP	Direkt B	Index OP	Index B	Extend. OP	Extend. B	Inher. OP	Inher. B	Bedingungsreg. 5 4 3 2 1 0 H I N Z V C
CBA	Vergleiche Akku	A-B									11	1	o o / / / /
CMPA	" Akku A Sp.	A-M	81	2	91	2	A1	2	B1	3			o o / / / /
CMPB	" Akku B "	B-M	C1	2	D1	2	E1	2	F1	3			o o / / / /
CPX	" Index	XH-M, XL-(M+1)	8C	3	9C	2	AC	2	BC	3			o o 7 / 8 o
TST	Abfrage, Null oder Minus	M-00					6D	2	7D	3			o o / / R R
TSTA	" " " "	A-00									4D	1	o o / / R R
TSTB	" " " "	B-00									5D	1	o o / / R R
BRA	Springe (relativ)	None	20	2									o o o o o o
BCC	" wenn Übertrag = 0	C=0	24	2									o o o o o o
BCS	" " " = 1	C=1	25	2									o o o o o o
BEQ	" " = Null	Z=1	27	2									o o o o o o
BGE	" " ≧ Null	N⊕V=0	2C	2									o o o o o o
BGT	" " > Null	Z+(N⊕V)=0	2E	2									o o o o o o
BHI	" " größer	C+Z = 0	22	2									o o o o o o
BLE	" " ≦ Null	Z+(N⊕V)=1	2F	2									o o o o o o
BLS	" " kleiner oder gleich	C+Z=1	23	2									o o o o o o
BLT	" " < Null	N⊕V=1	2D	2									o o o o o o
BMI	" " Minus	N=1	2B	2									o o o o o o
BNE	" " nicht Null	Z=0	26	2									o o o o o o
BVC	" " Überlauf=0	V=0	28	2									o o o o o o
BVS	" " " =1	V=1	29	2									o o o o o o
BPL	" " Plus	N=0	2A	2									o o o o o o
BSR	" Unterprogramm relat.		8D	2									o o o o o o
JMP	Springe Absolut	Siehe Spezial-					6E	2	7E	3			o o o o o o
JSR	" Unterprogramm Abs.	operationen.					AD	2	BD	3			o o o o o o
NOP	Keine Operation										02	1	o o o o o o
RTI	Rücksprung von Interrupt	Siehe Spezial-									3B	1	----10-----
RTS	" " Unterprogramm	operationen.									39	1	o o o o o o
SWI	Software-Interrupt										3F	1	o S o o o o
WAI	Warten auf Interrupt										3E	1	o 11o o o o

Wirkung von Spezialoperationen

JSR - Unterprogrammsprung:

INDXD

PC	Hauptprogramm
n	AD = JSR
n + 1	K = Offset *
n + 2	Nächster Befehl

* K = 8-Bit Wert ohne Vorzeichen

SP	Stack
→SP-2	
SP-1	(n + 2) H
SP	(n + 2) L

$(n + 2)_H$ und $(n + 2)_L$ bilden n + 2

PC	Unterprogramm
X + K	1. Unterpr.-Befehl

EXTND

PC	Hauptprogramm
n	BD = JSR
n + 1	SH = Unterpr.Adr.
n + 2	SL = Unterpr.Adr.
n + 3	Nächster Befehl

SP	Stack
→SP- 2	
SP- 1	(n + 3) H
SP	(n + 3) L

→= Stackregister nach Ausführung

PC	Unterprogramm
S	1. Unterpr.-Befehl

(S aus S_H und S_L gebildet)

BSR - Unterprogrammsprung (relativ:)

PC	Hauptprogramm
n	8D = BSR
n + 1	± K = Offset*
n + 2	Nächster Befehl

* K = 7-Bit Wert mit Vorzeichen

SP	Stack
→ SP - 2	
SP - 1	(n + 2) H
SP	(n + 2) L

n + 2 aus $(n + 2)_H$ und $(n + 2)_L$ gebildet

PC	Unterprogramm
n + 2 ± K	1. Unterpr.-Befehl

JMP-Sprung:

INDXD

PC	Hauptprogramm
n	6E = JMP
n + 1	K = Offset
	:
X + K	Nächster Befehl

EXTND

PC	Hauptprogramm
n	7E = JMP
n + 1	K_H = Nächste Adresse
n + 2	K_L = Nächste Adresse
	:
K	Nächster Befehl

RTS - Rücksprung von Unterprogramm:

PC	Unterprogramm
S	39 = RTS

SP	Stack
SP	
SP + 1	N_H
SP + 2	N_L

PC	Hauptprogramm
n	Nächster Befehl

Anhang C: Relative Sprungweiten

Der Bezugspunkt für die Sprungweite ist die OP-Adresse +2

Operationsteil	Adressteil	
		- 11
		- 10
		- F
		- E
		- D
		- C
		- B
		- A
		- 9
		- 8
		- 7
		- 6
		- 5
		- 4
		- 3
BRA	FE	- 2
	FF	- 1
		0
		1
		2
		3
		4
		5
		6
		7
		8
		9
		A
		B
		C
		D
		E
		.
		.
		.
		EF
		F0
		F1
		F2
		F3
		F4
		F5
		F6
		F7
		F8
		F9
		FA
		FB
		FC
		FD

Anhang D: Zuordnung Seitennummer - Speicheradressen

Jede TV-Seite hat 32 Bytes.

Nr.	TV-Seite	Anf.-Adr.		End-Adr.	Bemerkung
0	0	0	-	1F	
1	1	20	-	3F	
2	2	40	-	5F	
3	3	60	-	7F	0,25 k Byte
4	4	80	-	9F	"direkt"
5	5	A0	-	BF	
6	6	C0	-	DF	
7	7	E0	-	FF	
8	8	100	-	11F	
9	9	120	-	13F	
10	A	140	-	15F	
11	B	160	-	17F	
12	C	180	-	19F	0,5 k Byte
13	D	1A0	-	1BF	
14	E	1C0	-	1DF	
15	F	1E0	-	1FF	
16	10	200	-	21F	
17	11	220	-	23F	
18	12	240	-	25F	
19	13	260	-	27F	0,75 k Byte
20	14	280	-	29F	
21	15	2A0	-	2BF	
22	16	2C0	-	2DF	
23	17	2E0	-	2FF	
24	18	300	-	31F	
25	19	320	-	33F	
26	1A	340	-	35F	
27	1B	360	-	37F	
28	1C	380	-	39F	1 k Byte
29	1D	3A0	-	3BF	
30	1E	3C0	-	3DF	
31	1F	3E0	-	3FF	

Anhang E :

Konvertierungstafeln, Potenzen von 2

Zahlenumrechnung

hexadezimal - dezimal

Hex	Dez						
1000	4096	0100	256	0010	16	0001	1
2000	8192	0200	512	0020	32	0002	2
3000	12288	0300	768	0030	48	0003	3
4000	16384	0400	1024	0040	64	0004	4
5000	20480	0500	1280	0050	80	0005	5
6000	24576	0600	1536	0060	96	0006	6
7000	28672	0700	1792	0070	112	0007	7
8000	32768	0800	2048	0080	128	0008	8
9000	36864	0900	2304	0090	144	0009	9
A000	40960	0A00	2560	00A0	160	000A	10
B000	45056	0B00	2816	00B0	176	000B	11
C000	49152	0C00	3072	00C0	192	000C	12
D000	53248	0D00	3328	00D0	208	000D	13
E000	57344	0E00	3584	00E0	224	000E	14
F000	61440	0F00	3840	00F0	240	000F	15

dezimal - hexadezimal

dez	Hex						
10000	2710	1000	03E8	100	0064	10	000A
20000	4E20	2000	07D0	200	00C8	20	0014
30000	7530	3000	0BB8	300	012C	30	001E
40000	9C40	4000	0FA0	400	0190	40	0028
50000	C350	5000	1388	500	01F4	50	0032
60000	EA60	6000	1770	600	0258	60	003C
		7000	1B58	700	02BC	70	0046
		8000	1F40	800	0320	80	0050
		9000	2328	900	0384	90	005A

dezimal-hexadezimal-dual

0	0	0 0000
1	1	0 0001
2	2	0 0010
3	3	0 0011
4	4	0 0100
5	5	0 0101
6	6	0 0110
7	7	0 0111
8	8	0 1000
9	9	0 1001
10	A	0 1010
11	B	0 1011
12	C	0 1100
13	D	0 1101
14	E	0 1110
15	F	0 1111
16	10	1 0000
17	11	1 0001
18	12	1 0010
19	13	1 0011
20	14	1 0100

Potenzen von 2

0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576