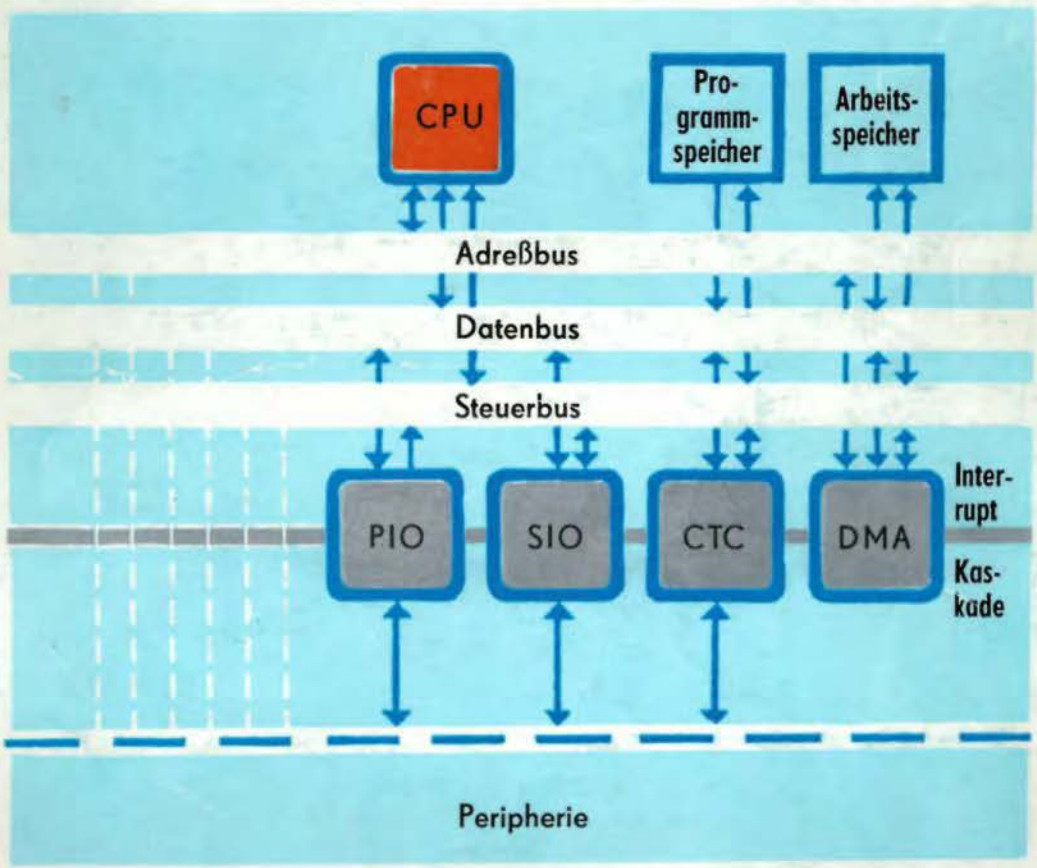
RFT

Mikroprozessorsystem

der II. Leistungsklasse

– Technische Beschreibung –



Zentrale Verarbeitungseinheit

CPU U 880 D

Technische Beschreibung

Zentrale Verarbeitungseinheit

CPU U880D

veb funkwerk erfurt

im veb kombinat mikroelektronik

Die technischen Angaben dieses Handbuches tragen reinen Informationscharakter. Verbindliche technische Liefer- und Reklamationsgrundlage sind ausschließlich die Typstandards. Die vorliegende Dokumentation gibt keine Auskunft über Liefermöglichkeiten und beinhaltet keine Verbindlichkeiten zur Produktion.

Inhaltsübersicht

1. Einführung

Die CPU U880 ist die Zentrale Verarbeitungseinheit (ZVE, Mikroprozessor) des U880-Mikroprozessorsystems. Zum System gehören u. a. folgende LSI-Schaltkreise:

- U880 D - CPU (Zentrale Verarbeitungseinheit)
- U855 D - PIO (Parallele Ein/Ausgabe)
- U856 D - SIO (Serielle Ein/Ausgabe)
- U857 D - CTC (Zähler/Zeitgeber)

Alle Schaltkreise werden in n-Kanal-Silicon-Gate-Technologie hergestellt und haben ein 40poliges DIL-Plastgehäuse (Ausnahme CTC mit 28poligem Gehäuse). Die Architektur der Schaltkreise realisiert ein einheitliches Systemkonzept das folgende wesentliche Merkmale aufweist:

- Signalzuführung über zwei Systembusse (Adressenbus, Datenbus) und eine Reihe von Steuerleitungen
- einheitliches Interruptregime mit der Möglichkeit der Prioritätsverkettung der Peripherieschaltkreise
- hochgradige Softwareprogrammierbarkeit der schaltkreisspezifischen Eigenschaften
- gemeinsamer Systemtakt als Einphasentakt mit einer maximalen Frequenz von 2,5 MHz
- einheitliche Spannungsversorgung von +5 V

Das System wird komplettiert durch Einsatz von Standard-Festwertspeichern (ROM, PROM) und statische oder dynamischen Standard-Schreib-Lesespeichern (RAM).
Die CPU U880 ist das Herz des Systems. Die Funktion der CPU besteht darin, Befehle aus dem Systemspeicher zu entnehmen und die dem Befehl zugeordnete Operation auszuführen. Solche Operationen sind im allgemeinen Erzeugung, Transport, Kombination oder Manipulation von Datenwerten. Datenquellen oder Datensenken können dabei die CPU, die Peripherieschaltkreise oder die Speicher sein. Der Datentransfer innerhalb des Systems erfolgt über die CPU (Ausnahme: Datentransfer mittels DMA-Betrieb).
Alle Peripherieschaltkreise können die CPU interrupten. Die Anzahl der Interruptquellen ist prinzipiell unbegrenzt, ihre Wertigkeiten und Interruptbedingungen sind innerhalb des Systems hardwaremäßig bzw. softwaremäßig frei wählbar.

Der Befehlssatz der CPU beinhaltet 158 Grundbefehle mit leistungsfähigen 16-Bit-, 8-Bit-, 4-Bit-, Einzel-Bit-Instruktionen sowie Block-Transport-, Block-Such-, Block-Eingabe- und Block-Ausgabe-Befehlen. Der Maschinenbefehlssatz ist aufwärtskompatibel zum international weit verbreiteten Mikroprozessor 8080 A. Der Befehlssatz wird durch seine mnemotechnischen Abkürzungen der Assemblersprache sehr eindeutig beschrieben, so daß sich die meisten Befehle selbst dokumentieren und dadurch das Erarbeiten von Software und auch das Einarbeiten in vorhandene Software wesentlich erleichtert wird.

Die vorliegende Schrift beschreibt die Technik des U880. Der Befehlssatz der CPU wird außerdem ausführlich in der Schrift "Befehlsbeschreibung des U880 D" bekanntgemacht.

2. Aufbau der CPU U 880

Das Blockschaltbild des internen Aufbaus der CPU U 880 ist in Bild 1 dargestellt. Es zeigt alle Hauptelemente in der CPU und sollte während der folgenden Beschreibung herangezogen werden.

2.1. CPU-Register

Die CPU U880 enthält 208 Bits im internen RAM, zu denen der Programmierer Zugriff hat. Bild 2 illustriert, wie dieser Speicher in 18 Register zu 8 Bit und 4 Register zu 16 Bit unterteilt ist. Alle U880-Register sind als statische RAMs ausgeführt. Die Register umfassen zwei Sätze von 6 Allgebrauchsregistern, die individuell als 8-Bitregister oder in Paaren als 16-Bitregister verwendet werden können. Ebenfalls sind zwei Akkumulatoren und zwei Flagregister vorhanden.

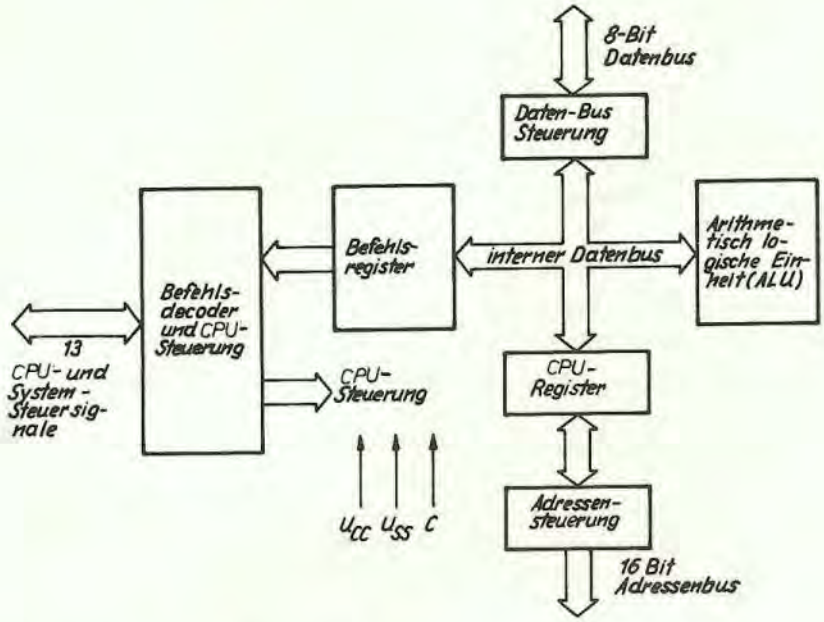


Bild 1: Blockschaltbild der CPU U880

Hauptregistersatz		Alternativsatz		
Akkumulator A	Flags F	Akkumulator A'	Flags F'	
B	C	B'	C'	Register zur allgemeinen Verwendung
D	E	D'	E'	
H	L	H'	L'	

Interrupt Vektor I	Speicher Refresh R	
Index Register	IX	Spezialregister
Index Register	IY	
Keller-Zeiger	SP	
Programmzähler	PC	

Bild 2: Aufbau der CPU-Register

Register für spezielle Zwecke

- Befehlszähler (PC)
  Der Befehlszähler enthält die 16-Bit-Adresse des aktuellen Befehls, der vom Speicher zu holen ist. Der Befehlszähler wird automatisch erhöht, nachdem sein Inhalt in die Adressenleitung überführt worden ist. Wenn Programmsprünge auftreten, wird der neue Wert automatisch in den PC überführt bei Überschreiben des erhöhten Wertes.
- Keller-Zeiger (SP = Stack pointer)
  Der Zeiger enthält die 16-Bit-Adresse des aktuellen obersten Wertes eines Kellers, der irgendwo in einem externen System-RAM untergebracht ist. Dieser externe Kellerspeicher ist als "zuletzt hinein, zuerst heraus" (LIFO)-Datei organisiert. Daten können, durch die Ausführung der PUSH- oder POP-Befehle, von speziellen CPU-Registern aus gekellert werden oder aus dem Keller in bestimmte CPU-Register zurückgeholt werden. Die aus dem Keller geholten Daten sind immer die, die als letzte zuvor gekellert wurden.
  Der Keller ermöglicht eine einfache Gestaltung von Mehrfach-Interrupts, unbegrenzter Unterprogrammtechnik und Vereinfachungen bei vielen Arten von Datenbehandlungen.
- Zwei Indexregister (IX + IY)
  Die zwei unabhängigen Register enthalten eine 16-Bit-Basis-Adresse, die bei indizierter Adressierung verwendet wird. Dabei wird ein Indexregister als Basis benutzt, um das Gebiet im Speicher festzulegen, von dem aus Daten gespeichert oder entnommen werden sollen. Ein zusätzliches Byte ist in indizierten Befehlen enthalten, um die Entfernung von dieser Basis anzugeben. Diese Entfernung ist als Zweierkomplement der entsprechenden Zahl spezifiziert. Diese Adressierungsart vereinfacht in starkem Maße viele Typen von Programmen, speziell dort, wo Tabellen-Daten benutzt werden.

- Interrupt-Vektor-Register (I)
  Die CPU U880 kann so betrieben werden, daß ein indirekter Aufruf zu irgendeinem Speicherplatz in Abhängigkeit von einem Interrupt erreicht werden kann. Das Register I enthält dann die höchsten 8 Bits der indirekten Adresse, während die den Interrupt auslösende Schaltung die unteren 8 Bits der Adresse liefert. Diese Verfahrensweise ermöglicht es, Interrupt-Routinen dynamisch irgendwo im Speicher mit absolut geringster Zugriffszeit zu dieser Routine abzuspeichern.
- Speicher-Auffrisch-Register (R)
  Die CPU U880 enthält einen Speicher-Auffrisch-Zähler, der dafür sorgt, daß dynamische Speicher genauso benutzt werden können wie statische. Dieses 7-Bit-Register wird automatisch nach jedem Befehlsaufruf erhöht. Die Daten im Auffrisch-Zähler werden auf den unteren Teil des Adressbusses mit einem Refresh-Steuersignal abgesetzt, während die CPU den aufgerufenen Befehl dekodiert und ausführt. Diese Art der Auffrischung ist vollständig programmtransparent und senkt nicht die Arbeitsgeschwindigkeit der CPU. Der Programmierer kann das Register R zu Testzwecken laden, aber das Register wird normalerweise nicht von Programmierer benutzt.

Akkumulator und Flag-Register

Die CPU enthält zwei unabhängige 8-Bit-Akkumulatoren und verbunden damit zwei 8-Bit-Flag-Register. Der Akkumulator enthält die Ergebnisse der 8-Bit-Rechenoperationen oder logischen Operationen, wohingegen die Flag-Register die speziellen Bedingungen für die 8- oder 16-Bit-Operationen anzeigen, z. B. wenn das Ergebnis einer Operation gleich Null oder ungleich Null ist. Der Programmierer wählt das Akkumulator-Flag-Paar, mit dem er arbeiten möchte durch einen einzigen Tausch-Befehl, so daß er mit jedem Paar arbeiten kann.

Allgebrauchs-Register

Es gibt zwei zueinander passende Sätze von Allgebrauchsregistern. Jeder Satz enthält sechs 8-Bit-Register, die einzeln als 8-Bit-Register oder paarweise als 16-Bit-Register durch den Programmierer verwendet werden können. Der eine Satz wird mit BC, DE und HL und der Alternativsatz mit BC', DE',und HL' bezeichnet. Zu jeder beliebigen Zeit kann der Programmierer einen Satz mittels eines Austauschbefehls zur Benutzung auswählen. In Systemen, wo schnelle Interrupt-Behandlung erforderlich ist, kann ein Satz von Allgebrauchsregister und ein Akkumulator mit Steuerflags für die Behandlung dieser schnellen Routinen reserviert werden. Nur ein einfacher Austausch-Befehl ist erforderlich, um die Anfangsbedingungen der Routine zu laden. Das verkürzt wesentlich die Interrupt-Behandlungszeit. Diese Allgebrauchsregister sind in einem großen Anwendungsbereich durch den Programmierer zu nutzen. Sie vereinfachen auch die Programmierung, besonders bei auf ROM orientierten Systemen, bei denen nur ein kleiner externer Lese-/Schreib-Speicherbereich verfügbar ist.

2.2. Rechenwerk und Logische Einheit (ALU)

Die arithmetischen und logischen 8-Bit-Befehle der CPU werden in der ALU ausgeführt. Intern steht die ALU mit dem Register und dem externen Datenbus über den internen Datenbus in Verbindung. Die Funktionsarten, die durch die ALU ausgeführt werden, sind folgende:

Addition	Links-/Rechtsverschiebung oder
Subtraktion	zyklische Verschiebung
logisches UND	(arithmetisch und logisch)
logisches ODER	Erhöhen um 1
logisches exklus. ODER	Erniedrigen um 1
Vergleichen	Bit-Setzen
	Bit-Löschen
	Bit-Testen

2.3. Befehlsregister und CPU-Steuerung

Wenn ein Befehl vom Speicher geholt worden ist, wird er ins Befehlsregister geladen und dekodiert. Das Steuerteil führt diese Funktion durch, erzeugt alle Signale, die erforderlich sind, um Daten von oder zu den Registern zu lesen oder zu schreiben, die ALU zu steuern, gibt diese Signale aus und liefert alle extern erforderlichen Steuersignale.

3. CPU U880 - Anschlußbeschreibung

Die CPU U880 ist in einem standardisierten Dual-In-Line-Gehäuse mit 40 Anschlüssen verpackt. Bild 3 zeigt die Anschlußbelegung und das logische Schaltbild des U880.

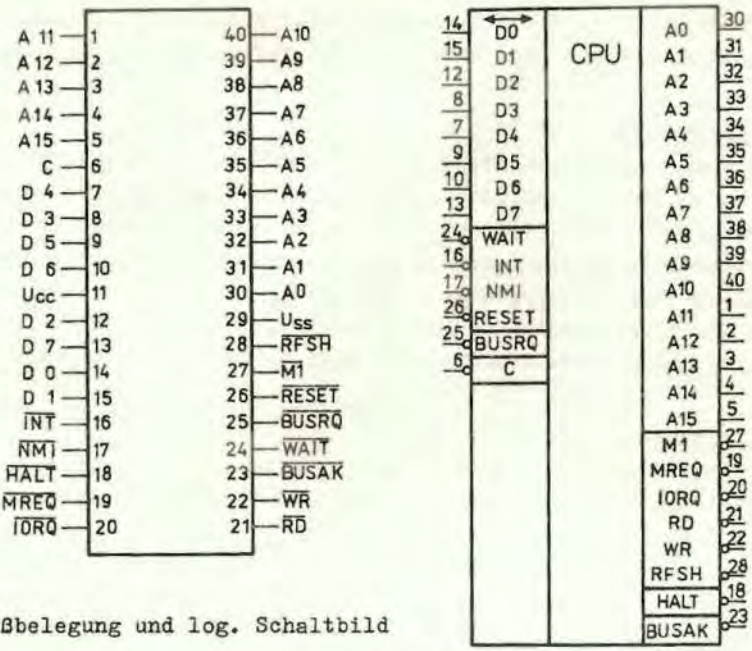


Bild 3: Anschlußbelegung und log. Schaltbild

AØ-A15 (Adressen-Bus). Drei-Zustands-Ausgang, aktiv "high". AØ-A15 bilden einen 16-Bit-Adressen-Bus. Der Adressen-Bus liefert die Adressen für Speicher-Daten-Transporte (bis zu 64 K Bytes) und Datentransporte für E/A-Geräte. Die E/A-Adressierung benutzt die unteren 8 Adressen-Bits, um eine direkte Anwahl von 256 Eingangs- oder 256 Ausgangskanälen zu ermöglichen. AØ ist das niederwertigste Adressen-Bit. Während der Auffrischzeit enthalten die unteren 7 Bits die gültige Adresse für die Speicher-Auffrischung.

DØ-D7 (Daten-Bus). Drei-Zustands-Ein- und -Ausgang, aktiv "high". DØ-D7 bilden einen bidirektionalen 8-Bit-Datenbus. Der Datenbus dient Datentransporten von oder zum Speicher und von oder zu E/A-Geräten.

$\overline{M1}$ (Maschinen Zyklus 1). Ausgang, aktiv "low". $\overline{M1}$ zeigt an, daß sich der laufende Maschinenzyklus der Befehlsabarbeitung im Zustand "Aufruf des Operationskodes" befindet. Beachte, daß $\overline{M1}$ während der Ausführung eines 2 Byte langen Operationskodes beim Aufruf des OP-Kodes eines jeden Bytes erzeugt wird. Diese 2-Byte-OP-Kodes beginnen immer mit CBH, DDH, EDH oder FDH. $\overline{M1}$ tritt auch zusammen mit $\overline{IORQ}$ auf, um einen Interrupt-Annahme-Zyklus anzuzeigen.

$\overline{MREQ}$ (Speicher-Anforderung-memory request). Drei-Zustands-Ausgang, aktiv "low". Das Signal Speicher-Anforderung zeigt an, daß der Adressen-Bus eine gültige Adresse für eine Speicher-Lese- oder -Schreiboperation enthält.

$\overline{IORQ}$ (Ein/Ausgabe-Anforderung - Input/Output request). Drei-Zustands-Ausgang, aktiv "low". Das $\overline{IORQ}$-Signal zeigt an, daß die untere Hälfte des Adressenbusses eine gültige E/A-Adresse für eine E/A-Lese- oder Schreiboperation enthält. Ein $\overline{IORQ}$-Signal wird auch während $\overline{M1}$ aktiv generiert, um bei Interrupt-Annahme anzuzeigen, daß ein entsprechender Interrupt-Vektor auf den Daten-Bus gelegt werden kann. Interrupt-Annahme-Operationen treten während der $\overline{M1}$-Zeit auf, wogegen E/A-Operationen niemals während der $\overline{M1}$-Zeit durchgeführt werden.

$\overline{RD}$ (Lesen - read). Drei-Zustands-Ausgang, aktiv "low". $\overline{RD}$ zeigt an, daß die CPU Daten vom Speicher oder von einem E/A-Gerät lesen will. Das adressierte E/A-Gerät oder der Speicher können dieses Signal dazu benutzen, die Daten auf den CPU-Daten-Bus aufzutasten.

$\overline{WR}$ (Schreiben - write). Drei-Zustands-Ausgang, aktiv "low". $\overline{WR}$ zeigt an, daß der CPU-Daten-Bus gültige Daten enthält, die im adressierten Speicherplatz oder E/A-Gerät gespeichert werden sollen.

$\overline{RFSH}$ (Speicherauffrischen - refresh). Ausgang, aktiv "low". $\overline{RFSH}$ zeigt an, daß die unteren 7 Bits des Adressenbusses eine Auffrischadresse für dynamische Speicher enthalten. Das aktuelle $\overline{MREQ}$-Signal kann für das Auffrischlesen aller dynamischen Speicher benutzt werden.

$\overline{HALT}$ (HALT-Zustand - HALT state). Ausgang, aktiv "low". $\overline{HALT}$ zeigt an, daß die CPU einen Software-HALT-Befehl ausführt und nun entweder einen nicht maskierbaren oder einen maskierbaren Interrupt erwartet bevor diese Operation abgeschlossen werden kann. Die CPU führt im HALT-Zustand NOP-Befehle (Leerbefehle) aus, um die Auffrischung der Speicher aufrecht zu erhalten.

$\overline{WAIT}$ (Warten - wait). Eingang, aktiv "low". $\overline{WAIT}$ zeigt der CPU U880, daß der adressierte Speicherplatz oder das E/A-Gerät noch nicht für eine Datenübertragung bereit sind. Die CPU generiert weiterhin WAIT-Zustände, solange dieses Signal aktiv ist. Mit Hilfe dieses Signals können Speicher oder E/A-Geräte beliebiger Geschwindigkeit mit der CPU synchronisiert werden.

$\overline{INT}$ (Interrupt-Anforderung - interrupt-request). Eingang, aktiv "low". Das Interrupt-Anforderungssignal wird durch ein E/A-Gerät erzeugt. Eine Anforderung wird am Ende des laufenden Befehls beachtet, wenn das Interrupt-Annahme-Flip-Flop, das durch die interne Software gesteuert wird, bereit ist und wenn das $\overline{BUSRQ}$-Signal nicht aktiv ist. Nimmt die CPU den Interrupt an, so wird das Interrupt-Annahme-Signal bei Beginn des nächsten Befehlszyklus ($\overline{IORQ}$ während M1) ausgesandt. Die CPU kann einen Interrupt auf drei verschiedene Arten beantworten (siehe Abschnitt 8.2.).

$\overline{NMI}$ (Nicht maskierbarer Interrupt - non maskable interrupt). Eingang, auf der negativen Flanke getriggert. Triggerflanke aktiviert ein internes NMI-Flip-Flop. Die Leitung "Nicht maskierbarer Interrupt" hat eine höhere Priorität als $\overline{INT}$ und wird immer am Ende des anliegenden Befehls getestet, unabhängig von der Lage des Interrupt-Annahme-Flip-Flops. $\overline{NMI}$ zwingt die CPU automatisch zu einem RESTART ab Speicherplatz 0066_H. Der Befehlszähler wird automatisch im externen Keller gerettet, so daß der Anwender zu dem Programm zurückkehren kann, das unterbrochen wurde.
Achtung: kontinuierliche WAIT-Zyklen können das Ende des anliegenden Befehls verhindern, ein $\overline{BUSRQ}$ überschreibt ein $\overline{NMI}$!

$\overline{RESET}$ Eingang, aktiv "low". $\overline{RESET}$ stellt den Befehlszähler auf Null und weist der CPU die Anfangswerte zu.
Diese Anfangswertzuweisung umfaßt:

- Ausschalten des Interrupt-Annahme-Flip-Flops
- Setzen des Registers I = 00_H
- Setzen des Registers R = 00_H
- Setzen der Interrupt-Art 0

Während der RESET-Zeit gehen der Adressen-Bus und der Daten-Bus in den hochohmigen Zustand und alle Steuersignal-Ausgänge gehen in den inaktiven Zustand.

$\overline{BUSRQ}$ (Busanforderung - bus request). Eingang, aktiv "low". Das Busanforderungssignal wird benutzt, um die CPU aufzufordern, den Adressen- und Daten-Bus und die Drei-Zustands-Ausgangssignale in den hochohmigen Zustand zu bringen, damit andere Geräte diese Busse steuern können. Wenn $\overline{BUSRQ}$ aktiviert wird, dann versetzt die CPU diese Busse in den hochohmigen Zustand, so bald der laufende Maschinenzyklus der CPU abgeschlossen ist.

$\overline{\text{BUSAK}}$ (Busbestätigung - Bus acknowledge). Ausgang, aktiv "low". Die Busbestätigung wird benutzt, um dem anfordernden Gerät zu melden, daß der CPU-Adressen- und Daten-Bus und die Drei-Zustands-Steuerbus-Signale in den hochohmigen Zustand gegangen sind, und daß das externe Gerät nun diese Signale steuern kann.

C Takteingang für Einphasentakt

4. Zeitabläufe in der CPU

Die CPU U880 führt alle Befehle durch schrittweises Abarbeiten eines genau festgelegten Satzes weniger Grundoperationen durch
Diese umfassen:

Speicherlesen oder -schreiben
E/A-Gerätelesen oder -schreiben
Interruptannahme.

Alle Befehle bestehen aus einer Aneinanderreihung von diesen Grundoperationen. Jede dieser Grundoperationen kann 3 bis 6 Takte dauern oder sie können durch zusätzliche WAIT-Zeiten verlängert werden, wenn die CPU mit der Geschwindigkeit externer Geräte synchronisiert werden muß. Die Taktperioden des Taktsignals sind als T-Zyklen und die Grundoperationen als M-Zyklen (für Maschinen-) bezeichnet.
Bild 4 zeigt, daß sich ein typischer Befehl nur aus einer Reihe von speziellen M- und T-Zyklen zusammensetzt. Man erkennt, daß dieser Befehl aus drei Maschinenzyklen (M1, M2, M3) besteht. Der erste Maschinenzyklus jedes Befehls ist der OP-Kode-Aufruf-Zyklus, der vier, fünf oder sechs T-Zyklen lang ist (wenn er nicht durch ein $\overline{\text{WAIT}}$-Signal verlängert wird). Im Aufruf-Zyklus (M1) wird der Operationskode des nächsten Befehls aufgerufen, um ihn danach auszuführen. Die folgenden Maschinenzyklen transportieren die Daten zwischen der CPU und dem Speicher oder E/A-Geräten und können zwischen 3 und 5 T-Zyklen lang sein (sie können ebenfalls wieder durch ein $\overline{\text{WAIT}}$-Signal verlängert werden, wenn die CPU mit externen Geräten zu synchronisieren ist). In den folgenden Abschnitten werden die Zeitverläufe behandelt, die innerhalb eines beliebigen Grund-Maschinenzyklusses auftreten. Im Abschnitt 7 wird der exakte Zeitbedarf für jeden Befehl angegeben.

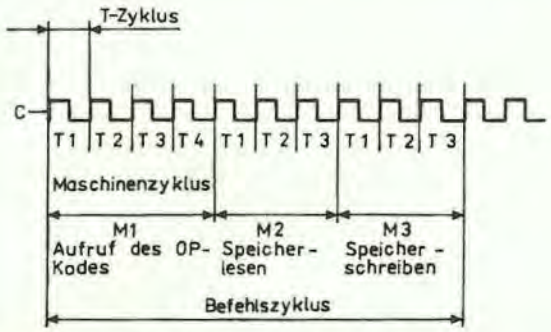


Bild 4: Beispiel des generellen Zeitablaufes in der CPU

Alle CPU-Zeitverläufe können in wenige sehr einfache Zeitabläufe unterteilt werden, die in den Bildern 5 bis 15 dargestellt sind.

Aufruf des Operationskodes des Befehls (M1-Zyklus)
Speicher-Daten-Schreib- oder Lese-Zyklus
E/A-Lese- oder Schreibzyklus
Busanforderungs/-bestätigungszyklus
Interruptanforderung/-annahme-Zyklus
Nicht maskierbarer Interrupt: Anforderungs-/Annahme-Zyklus
Ausführung eines HALT-Befehls

Befehlsaufruf

Bild 5 zeigt die Zeitverläufe während eines M1-Zyklus (Aufruf des OP-Kodes). Zu beachten ist, daß der PC (Befehlszähler) zu Beginn des M1-Zyklus auf den Adressenbus gegeben wird. Eine halbe Taktzeit später wird das $\overline{\text{MREQ}}$-Signal aktiv. Bis zu diesem Zeitpunkt hatte die Adresse für den Speicher Zeit gehabt, um sich zu stabilisieren, so daß die Abfallflanke von $\overline{\text{MREQ}}$ direkt als Chip-Freigabetakt für die Speicher benutzt werden kann. Die $\overline{\text{RD}}$-Leitung wird gleichzeitig dazu aktiv und zeigt damit an, daß die vom Speicher gelesenen Daten auf den CPU-Datenbus gegeben werden können. Die CPU übernimmt die Daten vom Datenbus mit der Anstiegsflanke des Taktes T3. Die gleiche Flanke wird durch die CPU auch benutzt, um die Signale $\overline{\text{RD}}$ und $\overline{\text{MREQ}}$ abzuschalten. Folglich sind die Daten schon durch die CPU übernommen bevor das $\overline{\text{RD}}$-Signal inaktiv wird. Die Takte T3 und T4 des Befehlsaufrufzyklus werden benutzt, um die dynamischen Speicher aufzufrischen. In der CPU wird diese Zeit benutzt, um den aufgerufenen Befehl zu dekodieren und auszuführen. Während T3 und T4 enthalten die unteren 7 Bits des Adressenbusses die Speicheradresse für die Auffrischung und das $\overline{\text{RFSH}}$-Signal wird aktiv und zeigt damit an, daß ein Auffrischlesen aller dynamischen Speicher durchgeführt werden kann. Ein $\overline{\text{RD}}$-Signal wird während der Auffrischzeit nicht erzeugt, um zu vermeiden, daß Daten von verschiedenen Speicherbereichen auf den Datenbus gegeben werden. Das $\overline{\text{MREQ}}$-Signal kann dazu benutzt werden, während der Auffrischzeit ein Auffrischungslesen aller Speicherelemente durchzuführen. Das Auffrischsignal selbst kann dazu nicht verwendet werden, weil die Auffrischadresse nur während der $\overline{\text{MREQ}}$-Zeit garantiert stabil ist.

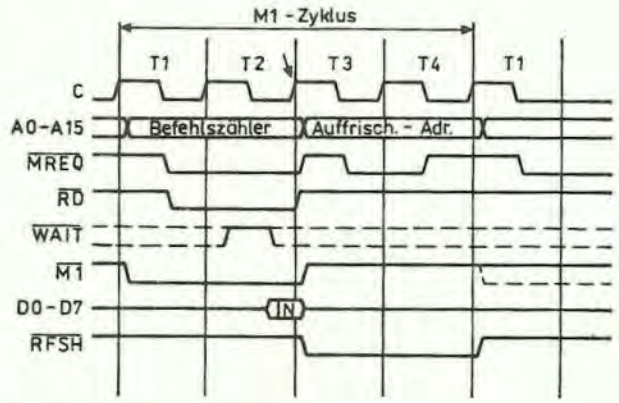


Bild 5: Aufruf des OP-Kodes eines Befehls

Das Bild 6 illustriert, wie der Befehlsaufrufzyklus verzögert wird, wenn der Speicher die $\overline{\text{WAIT}}$-Leitung aktiviert. In T2 und jedem nachfolgendem TW tastet die CPU die $\overline{\text{WAIT}}$-Leitung während der Abfallflanke des Taktes ab. Wenn die $\overline{\text{WAIT}}$-Leitung während dieser Zeit aktiv ist, wird ein weiterer WAIT-Zustand im folgenden T-Zyklus erzeugt.
Mit diesem Verfahren kann der Lesezyklus beliebig verlängert und so an die Zugriffszeit von beliebigen Speicheranordnungen angepaßt werden.

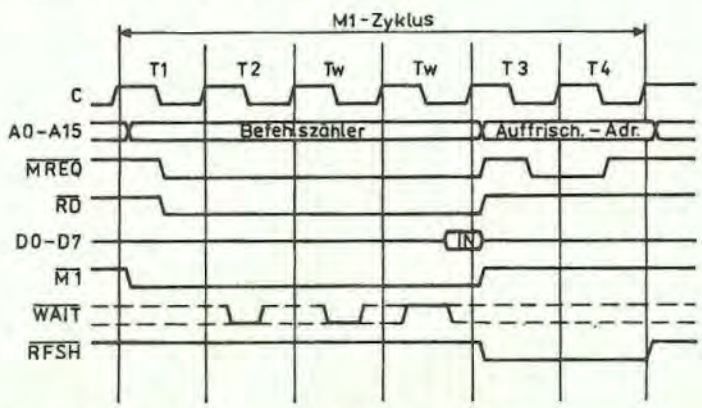


Bild 6: Aufruf des OP-Kodes eines Befehls mit WAIT-Zuständen

Speicherlesen oder -schreiben

Bild 7 stellt die Zeitverläufe während der Speicherlese- oder -schreibzyklen dar, die sich vom M1-Zyklus unterscheiden. Diese Zyklen sind allgemein 3 Taktperioden lang, wenn nicht über das $\overline{WAIT}$-Signal durch den Speicher WAIT-Zustände erzwungen werden.
Das $\overline{MREQ}$- und das $\overline{RD}$-Signal werden genauso verwendet wie im Aufrufzyklus. Im Falle des Speicherschreibzyklus wird $\overline{MREQ}$ aktiv, wenn der Adressenbus stabil ist, so daß es direkt als Chip-Freigabe für Speicher verwendet werden kann. Die $\overline{WR}$-Leitung ist aktiv, wenn die Daten auf dem Datenbus stabil sind, so daß dieses Signal direkt als ein R/W-Impuls für jede Type von Halbleiterspeichern verwendet werden kann. Das $\overline{WR}$-Signal wird einen halben Takt, bevor die Adressenbus- und die Datenbus-Inhalte verändert werden, inaktiv, so daß die Überlappungserfordernisse im Prinzip für alle Typen von Halbleiterspeichern erfüllt werden.

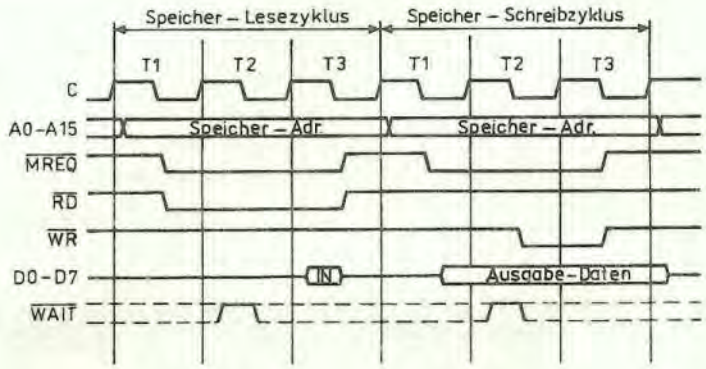


Bild 7: Speicher-Lese-oder-Schreibzyklus

In Bild 8 ist dargestellt, wie eine $\overline{WAIT}$-Forderung jede Speicher- oder Schreiboperation verlängert. Diese Operation läuft genauso ab, wie sie im Aufrufzyklus beschrieben wurde.
Zu beachten ist, daß in diesem Bild ein separater Lesezyklus und ein Schreibzyklus gemeinsam dargestellt sind, obwohl Lese- und Schreibzyklen niemals gleichzeitig auftreten können.

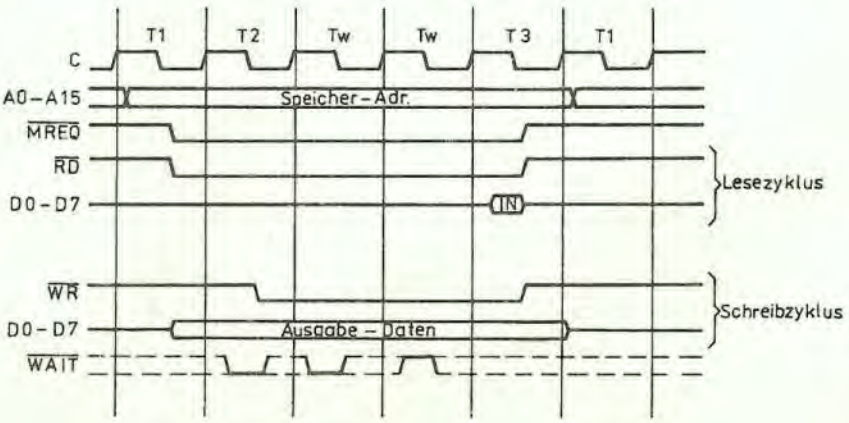


Bild 8: Speicher-Lese-oder-Schreibzyklen mit WAIT-Zuständen

Ein- oder Ausgabe-Zyklen

Bild 9 zeigt die E/A-Lese- oder E/A-Schreiboperation. Zu beachten ist, daß während der E/A-Operationen automatisch ein WAIT-Zustand eingebaut wird. Der Grund dafür ist, daß bei E/A-Operationen die Zeit vom Aktivwerden des $\overline{IORQ}$-Signals bis zum Zeitpunkt, zu dem die CPU die WAIT-Leitung abfragt, sehr kurz ist und damit ohne diesen extra WAIT-Zustand für die E/A-Kanäle nicht genügend Zeit zur Verfügung steht, um die Adresse zu dekodieren und gegebenenfalls die WAIT-Leitung zu aktivieren. Während dieses WAIT-Zustandes wird das $\overline{WAIT}$-Forderungssignal abgetastet. Während einer Lese-E/A-Operation wird die $\overline{RD}$-Leitung benutzt, um den adressierten Kanal auf den Datenbus zu schalten, wie es in dem Falle des Speicher-Lesens der Fall war. Bei E/A-Schreib-Operationen wird die $\overline{WR}$-Leitung als Takt für die E/A-Kanäle benutzt, wiederum mit automatisch zur Verfügung stehender ausreichender Überlappungszeit, so daß die

Anstiegsflanke als Daten-Takt benutzt werden kann.

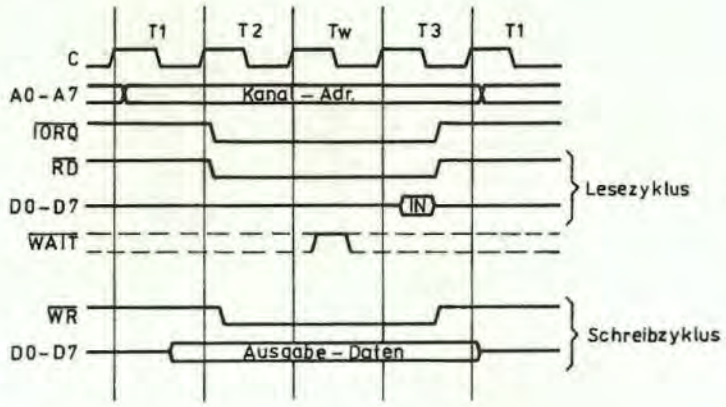


Bild 9: Eingabe- oder Ausgabe-Zyklen

In Bild 10 ist dargestellt, wie zusätzliche WAIT-Zustände über die $\overline{\text{WAIT}}$-Leitung eingefügt werden können. Diese Operation ist identisch mit den bereits beschriebenen.

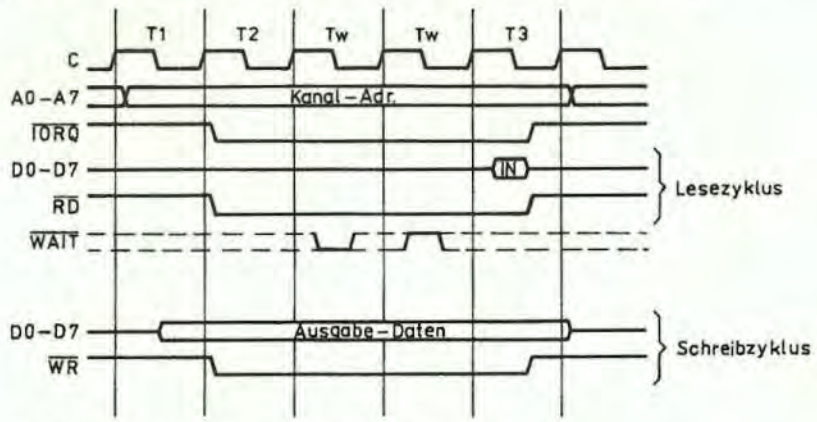


Bild 10: Eingabe- oder Ausgabe-Zyklen mit WAIT-Zuständen

Bus-Anforderungs-/Bus-Bestätigungszyklus

In Bild 11 ist das Zeitverhalten für einen Busanforderungs-/Busbestätigungszyklus dargestellt. Das $\overline{\text{BUSRQ}}$-Signal wird von der CPU mit der Anstiegsflanke des letzten Taktes eines jeden Maschinenzyklus abgetastet. Wenn das $\overline{\text{BUSRQ}}$-Signal aktiv ist, setzt bei der Anstiegsflanke des nächsten Taktimpulses die CPU ihre Adressen-Daten- und Drei-Zustands-Steuersignale in den hochohmigen Zustand. Zu dieser Zeit kann jedes externe Gerät die Busse steuern und die Daten zwischen Speicher und E/A-Geräten transportieren. Dies wird allgemein als direkter Speicherzugriff bezeichnet (DMA - direct memory access). Die maximale Zeitdauer bis zur Antwort der CPU auf eine Busanforderung ist die Länge eines Maschinenzyklus. Die externe Steuerung kann die Bus-Steuerung über so viele Takte behalten, wie es erforderlich ist. Zu beachten ist, daß bei sehr langen DMA-Zyklen und bei Verwendung von dynamischen Speichern die Auffrischfunktion durch die externe Steuerung übernommen werden muß. Diese Situation tritt nur auf, wenn sehr große Blöcke von Daten mit DMA-Steuerung übertragen werden. Zu beachten ist weiterhin, daß die CPU während des Busanforderungszyklus weder durch $\overline{\text{NMI}}$ noch durch ein $\overline{\text{INT}}$-Signal unterbrochen werden kann.

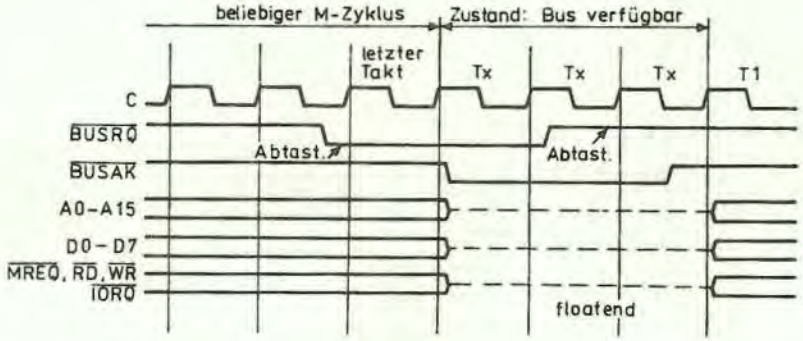


Bild 11: Busanforderungs-/ Bestätigungs-Zyklus

Interrupt-Anforderungs-/Annahme-Zyklus

Bild 12 zeigt das Zeitverhalten bei einem Interrupt-Zyklus. Das Interrupt-Signal ($\overline{INT}$) wird durch die CPU an der Anstiegsflanke des letzten Taktes am Ende eines Befehls abgefragt. Das Signal wird nicht berücksichtigt, wenn das Interrupt-Annahme-Flip-Flop, das durch die interne CPU-Software gesteuert wird, nicht eingeschaltet ist oder wenn das $\overline{BUSRQ}$-Signal aktiv ist. Wenn das Signal angenommen wird, wird ein spezieller M1-Zyklus erzeugt. Während dieses speziellen M1-Zyklus wird das $\overline{IORQ}$-Signal aktiv (im Gegensatz zum Normalfall mit $\overline{MREQ}$), um zu melden, daß das die Unterbrechung anfordernde Gerät seinen 8-Bit-Vektor auf den Datenbus geben kann. In diesen Zyklus werden automatisch zwei WAIT-Zustände aufgenommen. Diese Takte wurden eingefügt, um eine Interruptanordnung mit gestufter Priorität leichter verwirklichen zu können. Diese zwei WAIT-Zustände geben den Signalen genügend Zeit, um sich zu stabilisieren und festzustellen, welches E/A-Gerät in den Antwortvektor eingesetzt werden muß.

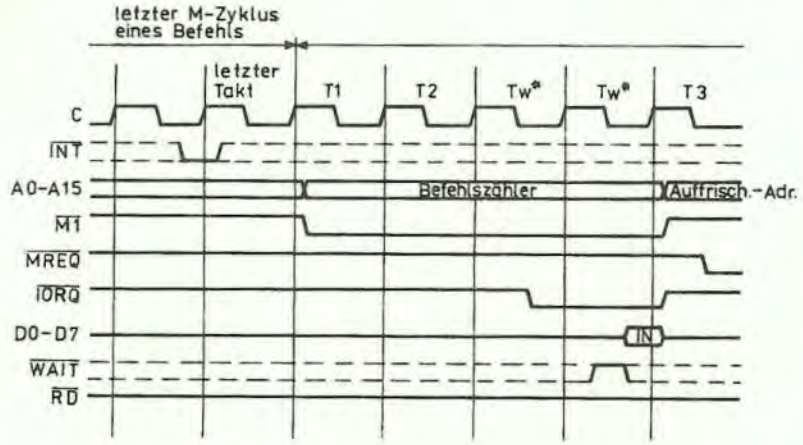


Bild 12: Interrupt-Anforderungs-/ Annahme-Zyklus

Bild 13 zeigt, wie zusätzliche WAIT-Zustände in den Interruptantwortzyklus aufgenommen werden können. Die Operation ist wieder identisch zu der bereits beschriebenen.

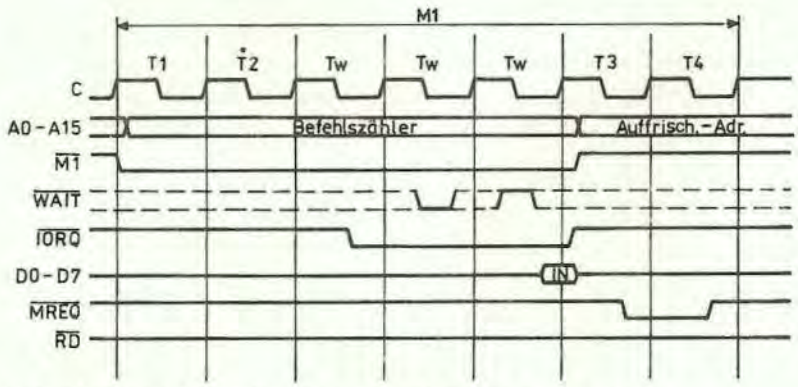


Bild 13: Interrupt-Anforderungs-/ Annahme-Zyklus mit WAIT-Zuständen

Nicht maskierbares Interrupt-Verhalten

Bild 14 zeigt den Anforderungs-/Annahme-Zyklus für einen nicht maskierbaren Interrupt. Das Signal wird zur gleichen Zeit wie die Interrupt-Leitung abgefragt, aber diese Leitung hat eine höhere Priorität als der normale Interrupt und kann durch eine Software-Steuerung nicht abgewiesen werden. Seine Funktion besteht darin, sofort eine Antwort auf wichtige Signale (z. B. drohender Netzausfall) zu liefern. Die CPU-Antwort auf einen nichtmaskierbaren Interrupt ist ähnlich einer normalen Speicherlese-Operation. Der einzige Unterschied besteht darin, daß der Inhalt des Datenbusses ignoriert wird, weil die CPU automatisch den Befehlszählerstand im externen Keller speichert und zur Adresse 0066_H springt. Die Routine zur Behandlung eines nicht maskierbaren Interrupts muß an diesem Speicherplatz beginnen, wenn diese Interruptart verwendet werden soll.

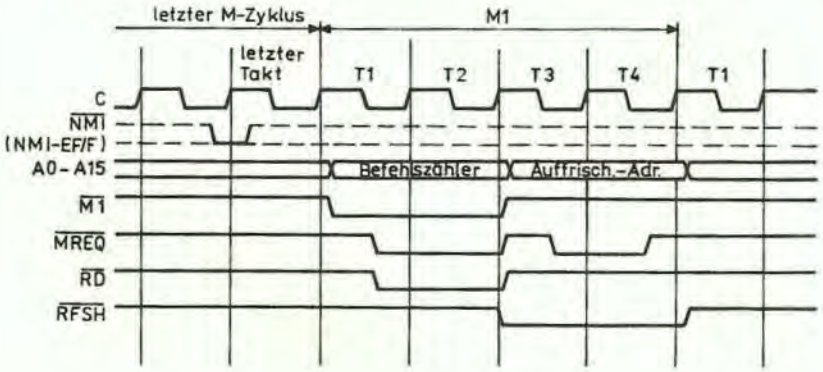


Bild 14: Anforderung eines nicht maskierbaren Interrupts

HALT Ausführung

Wenn ein Software-Halt-Befehl abzuarbeiten ist, beginnt die CPU Leerbefehle (NOP's) auszuführen, bis ein Interrupt empfangen wird (entweder ein nicht maskierbarer oder ein maskierbarer Interrupt, wenn das Interrupt-Annahme-Flip-Flop eingeschaltet ist). Die zwei Interrupt-Leitungen werden an jeder Anstiegsflanke des T4-Taktes abgefragt, wie das in Bild 15 gezeigt wird. Wenn ein nicht maskierbarer Interrupt oder ein maskierbarer Interrupt bei eingeschaltetem Interrupt-Annahme-Flip-Flop empfangen wird, dann wird der HALT-Zustand bei der nächsten Anstiegsflanke noch ausgeführt. Der folgende Zyklus wird dann der Interrupt-Annahme-Zyklus entsprechend dem Typ des empfangenen Interrupts. Wenn zu dieser Zeit beide empfangen wurden, dann wird der nicht maskierbare Interrupt angenommen, weil er die höchste Priorität besitzt. Die Leerbefehle (NOP's) werden während des HALT-Zustandes ausgeführt, um die Speicher-Auffrisch-Signale aktiv zu halten. Jeder Zyklus im HALT-Zustand ist ein normaler M1-(Aufruf) Zyklus mit der Ausnahme, daß die vom Speicher empfangenen Daten ignoriert werden und daß die Leerbefehle intern in der CPU erzeugt werden. Das HALT-Annahmesignal ist während dieser Zeit aktiv, um zu melden, daß sich die CPU im HALT-Zustand befindet.

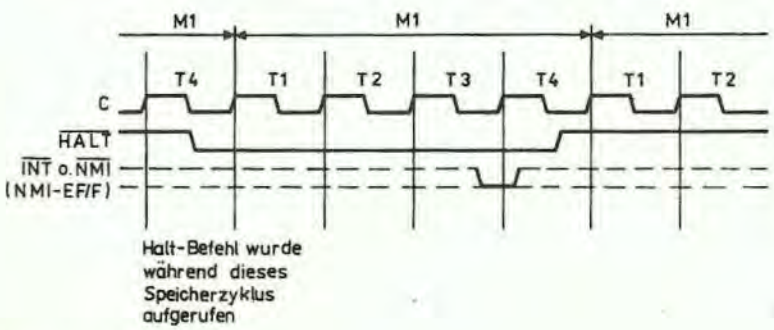


Bild 15: HALT-Ablauf

5. Befehls-Satz der CPU

Die CPU U880 kann 158 verschiedene Befehlsarten ausführen, wobei alle 78 Befehle der CPU 8080 A enthalten sind.
Die Befehle oder CPU U880 können in folgende Hauptgruppen unterteilt werden:

- Lade- und Austausch-Befehle
- Blocktransport und -suchbefehle
- arithmetische und logische Befehle
- Verschiebungs- und zyklische Verschiebungsbefehle
- Bit-Operationen (Setzen, Rückstellen, Test)
- Sprünge, UP-Aufruf und Rücksprung
- Ein/Ausgabe
- Grundsteuerung der CPU

5.1. Einführung in die Befehlsarten

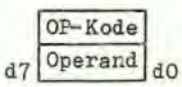
Die Lade-Befehle transportieren Daten intern zwischen den CPU-Registern oder zwischen CPU-Registern und dem externen Speicher. Die Befehle müssen eine Ausgangsadresse, von der die Daten zu entnehmen sind, und eine Zieladresse enthalten. Der Quellspeicherplatz wird durch den Ladebefehl nicht verändert. Die Gruppe umfaßt auch das Direktwert-Laden eines CPU-Registers oder einer externen Speicherzelle. Andere Arten von Ladebefehlen ermöglichen den Transport zwischen den CPU-Registern und Speicherplätzen. Die Austauschbefehle können die Inhalte von 2 Registern vertauschen.
Ein besonderer Satz von **Blocktransport-** und Blocksuchbefehlen ist ebenfalls vorhanden. Mit einem einzigen Befehl kann ein beliebig großer Block des Speichers zu einem anderen Speicherbereich transportiert werden. Der Satz von Blocktransportbefehlen ist außerordentlich wertvoll, wenn große Kettendaten verarbeitet werden müssen. Die **Blocksuchbefehle** sind für die Kettendatenverarbeitung sehr geeignet. Mit einem einzigen Befehl kann ein externer Speicherblock der gewünschten Größe nach einem beliebigen 8-Bit-Zeichen durchsucht werden. Wenn das Zeichen gefunden ist, endet der Befehl automatisch. Beide, sowohl die Blocktransporte als auch die **Blocksuchbefehle**, können während ihrer Abarbeitung unterbrochen werden, so daß sie die CPU nicht für längere Zeitabschnitte besetzen.
Die arithmetischen und logischen Befehle arbeiten mit Daten, die im Akkumulator (AC) und in anderen Allgebrauchs-CPU-Registern oder auf externen Speicherplätzen gespeichert sind. Die Ergebnisse dieser Operationen werden wiederum im AC abgelegt. Geeignete Flags werden entsprechend dem Ergebnis der Operation gesetzt. Diese Gruppe umfaßt auch die 16-Bit-Addition und Subtraktion zwischen den 16-Bit-CPU-Registern.
Die Bit-Operations-Befehle gestatten es, ein Bit am AC, in einem Allgebrauchsregister oder einem externen Speicherplatz mit einem einzigen Befehl zu setzen, zu löschen oder zu testen. Diese Gruppe ist besonders nützlich in Steuerungsanwendungen und zur Steuerung von Software-Flags für allgemeine Programmierzwecke.
Die Sprung-, Ruf- und Rücksprung-Befehle werden verwendet, um zwischen verschiedenen Speicherplätzen des Anwenderprogrammes zu springen. Diese Gruppe benutzt verschiedene Methoden, um die neue Programmadresse in den Befehlszähler von den verschiedenen speziellen Speicherplätzen zu überführen. Eine Variante davon ist der RESTART-Befehl. Dieser Befehl enthält im Falle des Aufrufs die neue Adresse als einen Teil des 8-Bit-Operationskodes. Das ist möglich, weil nur 8 verschiedene, im Teil 0 des externen Speichers angeordnete, Adressen angesprungen werden können. Programmsprünge können auch durch Laden der Register HL, IX, IY direkt in den Befehlszähler erreicht werden, d. h. die Sprungadresse kann eine komplexe Funktion der ausgeführten Routine sein.
Die Ein/Ausgabegruppe der U880-Befehle gestattet einen weiten Bereich von Datentransfers zwischen externen Speicherplätzen oder den Allgebrauchsregistern der CPU und den externen E/A-Geräten. In jedem Falle wird die Kanal-Nummer über die unteren 8 Bits des Adressenbusses während jeder E/A-Operation geliefert. Ein Befehl gibt die Möglichkeit, die Kanal-Nr. im 2.Byte des Befehls festzulegen, während die anderen U880-Befehle den Inhalt des Registers C als Kanal-Nr. verwenden. Ein Hauptvorteil, den Inhalt von Register C als Zeiger für das E/A-Gerät zu verwenden, ist darin zu sehen, daß es so möglich ist, gewöhnliche Software-Aufbereitungsroutinen den verschiedenen E/A-Kanäle zuzuweisen.

Ein anderes Merkmal dieser Eingabebefehle ist, daß sie das Flagregister automatisch setzen, so daß keine zusätzlichen Operationen nötig sind, um den Status der Eingabedaten zu ermitteln (z. B. ihre Parität). Die CPU U880 verfügt auch über Einzelbefehle, die Datenblöcke (bis zu 256 Bytes) automatisch zu oder von einem E/A-Kanal und direkt zu irgendeinem Speicherplatz transportieren können. In Verbindung mit dem Alternativsatz der Allgebrauchsregister liefern diese Befehle hohe E/A-Block-Transportgeschwindigkeiten.
Schließlich ermöglichen die Basissteuerbefehle die Wahl von verschiedenen Bedingungen und Betriebsarten der CPU. Diese Gruppe enthält auch solche Befehle wie das Ein- und Ausschalten des Interrupt-Annahme-Flip-Flops oder das Setzen der Betriebsart für das Interrupt-Verhalten.

5.2. Adressierungsarten

Die meisten der U880-Befehle arbeiten mit Daten, die in internen CPU-Registern, im externen Speicher oder in den E/A-Kanälen gespeichert sind. Unter Adressierung wird verstanden, wie in jedem Befehl die Adresse dieser Daten gebildet wird. Dieser Abschnitt gibt einen kurzen Überblick über die Adressierungsarten, die in der CPU U880 angewendet werden. In den folgenden Abschnitten wird auf Einzelheiten der Adressierungsarten eingegangen, die für jede Gruppe von Befehlen möglich sind.

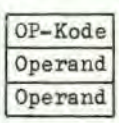
Direktwert (Immediate) Bei dieser Art der Adressierung enthält das dem OP-Kode folgende Byte direkt den aktuellen Operanden.



(1 oder 2 Byte)

Das Laden des Akkumulators mit einer Konstanten ist ein Beispiel für diese Art von Befehlen, wo die Konstante direkt das dem OP-Kode folgende Byte ist.

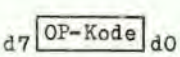
Direktwert erweitert (Immediate extended) Diese Art ist nur eine Erweiterung der Direktwertadressierung, indem 2 Bytes nach dem OP-Kode den Operand bilden.



(1 oder 2 Bytes)
(unterer Teil)
(oberer Teil)

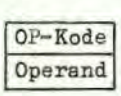
Als Beispiel dieser Befehlsart wäre das Laden des HL-Register-Paars (16-Bit-Register) mit 16 Bits (2 Bytes) zu nennen.

Modifizierte Null-Teil-Adressierung (medified page zero add.) Die CPU U880 hat einen speziellen Ein-Byte-Unterprogramm-Aufruf-Befehl für 8 Speicherplätze im Teil Null des Speichers. Dieser Befehl, der als RESTART bezeichnet wird, stellt den Befehlszähler auf eine effektive Adresse im Teil Null des Speichers. Dieser Befehl dient dazu, mit einem einzigen Byte eine vollständige 16-Bit-Adresse festzulegen, bei der gewöhnlich Unterprogramme stehen. Damit wird Speicherplatz gespart.



(1 Byte)
eff. Adresse: $(d5\ d4\ d3\ 0\ 0\ 0)_2$

Relative Adressierung (relativ addr.) Die relative Adressierung benutzt 1 Datenbyte, das auf den OP-Kode folgt und eine Distanz zur Zieladresse angibt zu der ein Programmsprung erfolgen soll. Die Entfernung ist ein vorzeichenbehaftetes Zweier-Komplement, das zur Adresse des OP-Kodes des folgenden Befehls addiert wird.



(rel. Sprung, 1 oder 2 Byte)
8 Bit-Zweierkomplement zur
Adresse addiert (A + 2)

Der Wert dieser Adressierungsart besteht darin, daß Sprünge zu den Nachbarspeicherplätzen möglich sind, wobei nur zwei Bytes Speicherplatz erforderlich sind. In den meisten Programmen sind die relativen Sprünge die am häufigsten auftretenden Arten von Sprüngen wegen der Nähe

der entsprechenden Programmteile. Daher können diese Befehle den Speicherbedarf erheblich reduzieren. Der vorzeichenbehaftete Abstand kann zwischen + 127 und - 128 von A + 2 aus liegen. D. h. es ist ein Abstandsbereich von + 129 bis - 126, von der Adresse des OP-Kodes des relativen Sprungs gerechnet, zu realisieren. Ein anderer Hauptvorteil ist der, daß so ein verschiebbares Maschinenprogramm ermöglicht wird.

Erweiterte Adressierung (extended addr.) Die erweiterte Adressierung enthält 2 Adressenbytes (16 Bits), auf die sich der Befehl bezieht. Diese Daten können eine Adresse sein, zu der zu springen ist, oder es kann die Adresse eines Operanden sein.

OP-Kode (1 oder 2 Bytes)
unterer Adressenteil o. unterer Teil d. Operanden
oberer Adressenteil o. oberer Teil d. Operanden

Die erweiterte Adressierung ist für Programmsprünge von einem Speicherbereich in einen anderen oder das Laden und Speichern von Daten in beliebigen Speicherplätzen erforderlich.
Wenn die erweiterte Adressierung angewendet wird, um die Quell- oder Zieladresse eines Operanden anzugeben, dann wird die Schreibweise (nn) verwendet, um den Inhalt des Speicherplatzes nn darzustellen, wobei nn die 16-Bit-Adresse ist, die durch den Befehl festgelegt wird. D. h. die zwei Adressenbytes nn werden als Zeiger zum Speicherplatz verwendet. Der Gebrauch der Klammern bedeutet immer, daß der Wert innerhalb der Klammern als ein Zeiger zum Speicherplatz zu betrachten ist. Z. B. bezieht sich (1200) auf den Inhalt des Speicherplatzes 1200.

Indizierte Adressierung (indexed addr.) Bei dieser Adressierungsart enthält das dem OP-Kode folgende Datenbyte eine Abstandsangabe, die zu einem der beiden Indexregister addiert wird, um den Zeiger zum Speicher zu bilden. Der OP-Kode legt fest, welches Indexregister benutzt wird. Der Inhalt der Indexregister wird durch diese Operation nicht verändert.

OP-Kode
OP-Kode
Abstandsangabe

Operand wird zum Inhalt des Indexregisters addiert, um den Zeiger zum Speicher zu bilden.

Ein indizierter Befehl würde z. B. das Laden des Inhalts eines Speicherplatzes (Indexregister + Abstand) in den Akkumulator sein. Die Abstandsangabe ist ein vorzeichenbehaftetes Zweier-Komplement. Mit Hilfe der indizierten Adressierung werden Programme wesentlich vereinfacht, wenn Tabellendaten verarbeitet werden sollen, da das Indexregister den Startpunkt einer Tabelle angeben kann. Es sind zwei Indexregister verfügbar, weil sehr oft die Operationen zwei oder mehr Tabellen erfordern. Indizierte Adressierung ermöglicht auch verschiebbare Maschinenprogramme. Die zwei Indexregister in der CPU U880 werden als IX und IY bezeichnet. Um die indizierte Adressierung anzugeben, benutzt man folgende Schreibweise:

(IX + d) oder (IY + d)

Wobei d die Abstandsangabe ist, die nach dem OP-Kode spezifiziert wird. Die Klammern deuten an, daß dieser Wert als Zeiger zum externen Speicher verwendet wird.

Register Adressierung Viele der U880-OP-Kodes enthalten Informtionsbits, die festlegen, welches CPU-Register für die Operation benutzt werden soll. Ein Beispiel für eine Register Adressierung würde das Laden von Daten des Registers B in das Register C sein.

Implizite Adressierung (Implied addr.) Die Implizite Adressierung tritt bei Operationen auf, bei denen automatisch durch den Operations-Kode ein oder mehrere CPU-Register als Träger der Operanden angenommen werden. Ein Beispiel dafür ist der Satz an arithmetischen Operationen, bei denen immer der Akkumulator das Resultat enthält.

Indirekte Register Adressierung Dieser Adressierungstyp bestimmt ein 16-Bit-Register-Paar (z. B. HL), das als Zeiger für einen Speicherplatz verwendet werden soll. Diese Befehlsart ist sehr leistungsfähig und kann sehr vielfältig angewendet werden.

OP-Kode	(1 oder 2 Bytes)

Ein Beispiel dieser Befehlsart würde das Laden des Akkumulators mit den Daten des Speicherplatzes sein, der durch den Inhalt des HL-Registers angezeigt wird. Die indizierte Adressierung ist eine Form der indirekten Registeradressierung mit dem Unterschied, daß bei der indizierten Adressierung noch ein Abstand addiert wird. Die indirekte Registeradressierung ermöglicht es, sehr leistungsfähig und dabei einfach den Speicherzugriff zu realisieren. Die Blocktransport- und -suchbefehle in der CPU U880 sind Erweiterungen dieser Adressierungsart, wobei automatisch die Registererhöhung/-erniedrigung und -vergleiche hinzugefügt werden. Die Schreibweise für die Darstellung der indirekten Registeradressierung ist so vereinbart, daß Klammern um den Namen des als Zeiger verwendeten Registers gesetzt werden. Z. B. bedeutet das Symbol

(HL),

daß der Inhalt des HL-Registers als Zeiger für einen Speicherplatz verwendet werden soll. Oft wird die indirekte Registeradressierung benutzt, um einen 16-Bit-Operanden anzugeben. In diesem Falle zeigt der Registerinhalt den niederwertigeren Teil des Operanden an. Die Registerinhalte werden dann automatisch erhöht, um den oberen Teil des Operanden zu erhalten.

Bitadressierung Die CPU U880 enthält eine große Anzahl von Bit-Setz-, -Lösch-und-Test-Befehlen. Diese Befehle lassen Bitoperationen an Speicherplätzen oder in den CPU-Registern zu, die durch eine der drei beschriebenen Adressierungsarten definiert werden (Register-, indirekte Register- und indizierte Adressierung). Welches der 8 Bits verarbeitet werden soll, wird durch 3 Bits im Operationskode des Befehls festgelegt.

Kombination von Adressierungsarten

Viele Befehle enthalten mehr als einen Operanden (z. B. arithmetische oder Ladebefehle). In diesen Fällen können zwei Adressierungsarten angewendet werden. Z. B. kann beim Laden die Direktwertadressierung verwendet werden, um die Quelle anzugeben und die indirekte Registeradressierung oder die indizierte Adressierung wird für die Zielangabe benutzt.

5.3. Operations-Kodes der Befehle

In diesem Abschnitt wird jeder der U880-Befehle beschrieben und es werden Tabellen mit den Operations-Kodes aller Befehle und ihren mnemotechnischen Abkürzungen der Assemblersprache angegeben. Alle OP-Kodes sind in hexadezimaler Schreibweise notiert. Ein-Byte-OP-Kodes erfordern entsprechend 2 hex. Zeichen und Doppel-Byte-OP-Kodes 4 hex. Zeichen. Die Umwandlungstabelle von hexadezimalen Zahlen in binäre ist hier zur Übersicht noch einmal aufgeführt.

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

Die mnemotechnischen Abkürzungen der U880-Befehle bestehen aus einem OP-Kode ohne, mit einem oder mit zwei Operanden. Bei Befehlen mit implizitem Operanden wird kein Operand gegeben. Bei Befehlen, die nur einen logischen Operanden haben oder bei denen der Operand invariant ist (z. B. der logische ODER-Befehl), wird nur ein mnemotechnischer Operand angegeben. Bei Befehlen, die zwei variable Operanden haben, werden zwei Operanden angegeben.
Für die Adressierungsart "(HL)" ist es auch gebräuchlich "M" (wie Memory) anzugeben.

Laden und Register-Tausch

Die Tabelle 1 zeigt die OP-Kodes für alle 8-Bit-Lade-Befehle, die in der CPU U880 implementiert sind. Es wird in dieser Tabelle auch für jeden Befehl die zulässige Adressierungsart angegeben. Die Quelle der Daten ist in der oberen Reihe und das Ziel der Daten in der linken Spalte angegeben. Zum Beispiel hat der Befehl - Laden des Registers C vom Register B - den OP-Kode 48 H. In allen Tabellen ist der OP-Kode in hexadezimaler Schreibweise angegeben. Der 48-H-Kode (binär = 0100 1000) wird durch die CPU während der M1-Zeit aus dem externen Speicher aufgerufen und dekodiert. Dann wird der Datentransport automatisch durch die CPU durchgeführt.

Die mnemotechnische Abkürzung der Assemblersprache für die ganze Gruppe heißt: LD (von Load). Diesem Kode folgt die Zielangabe, dann die Quellangabe (LD ZIEL, QUELLE). Zu beachten ist, daß etliche Kombinationen von Adressierungsarten zulässig sind. Z. B. kann man für die Quelle die Registeradressierung und für das Ziel die indirekte Registeradressierung benutzen. Z. B. Laden eines Speicherplatzes, der durch den Inhalt des HL-Registers bestimmt wird, mit dem Inhalt des Registers D. Der OP-Kode für diese Operation würde 72 H lauten. Die Assembler-Schreibweise für diesen Ladebefehl lautet dementsprechend:

LD (HL), D

Die Klammern um HL bedeuten, daß der Inhalt von HL als Zeiger für den Speicherplatz dienen soll. In allen U880-Befehlen der Assemblersprache werden immer die Zieladressen zuerst notiert, dann folgt die Angabe der Quelle. Die U880-Assemblersprache ist unter dem Gesichtspunkt der leichten Programmierbarkeit definiert worden. Die Befehle dokumentieren sich im wesentlichen selbst. Programme, die in der U880-Sprache geschrieben wurden, sind deshalb leicht zu erfassen.
Alle Ladebefehle, in denen entweder für die Quell- oder die Zieladresse die indizierte Adressierung eingesetzt ist, benötigen 3 Bytes Speicherplatz, wobei das dritte Byte den Abstand d angibt. Z. B. Laden des Registers E mit dem Operanden, dessen Adresse durch das Indexregister IX und eine Distanz von + 8 gebildet wird, muß wie folgt geschrieben werden:

LD E, (IX + 8)

Im Speicher ergibt sich dann folgendes Bild für diesen Befehl:

Adresse A	DD	
A + 1	5E	OP-Kode
A + 2	08	Abstandsoperand

Die zwei Befehle für die erweiterte Adressierung benötigen auch 3 Bytes, Z. B. lautet der Befehl für das Laden des Akkumulators mit dem Operanden aus dem Speicherplatz 6F32H:

LD A, (6F32H)

Der Maschinenkode dieses Befehls lautet:

Adresse A	3A	OP-Kode
A + 1	32	unterer Teil der Adresse
A + 2	6F	oberer Teil der Adresse

Zu beachten ist, daß der untere Teil der Adresse immer der erste Operand ist.

Die Direktwert-Ladebefehle sind 2-Byte-Befehle für die Allgebrauchs-8-Bit-Register. Der Befehl Laden des Registers H mit dem Direktwert 36H wird wie folgt geschrieben:

LD H, 36H

Sein Maschinenkode lautet:

Adresse A	26	OP-Kode
A + 1	36	Operand

Das Laden eines Speicherplatzes bei Angabe der Zieladresse über indizierte Adressierung und Direktwert für die Quelle erfordert einen 4-Byte-Befehl:

LD (IX-15), 21H

Maschinenkode:

Adresse A	DD	OP-Kode
A + 1	36	OP-Kode
A + 2	F1	Abst. (-15 als vorzeichenbehaft. Zweierkomplem.)
A + 3	21	Direktwert, der geladen werden soll

Achtung: Bei einer indizierten Adressierung folgt die Abstandsangabe immer direkt nach dem OP-Kode.

Ziel		Quelle: implizit I	implizit R	Register A	Register B	Register C	Register D	Register E	Register H	Register L	Register indirekt M (HL)	Register indirekt (BC)	Register indirekt (DE)	indiziert (IX+d)	indiziert (IY+d)	erweiterte Adresse (nn)	Direktwert n
REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
Register indirekt	M (HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
indiziert	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
erweiterte Adresse	(nn)			32 n n													
implizit	I			ED 47													
	R			ED 4F													

Tabelle1: 8-Bit-Lade-Gruppe "LD" Ziel, Quelle

Tabelle 2 spezifiziert die 16-Bit-Lade-Operationen. Diese Tabelle ist der vorangegangenen sehr ähnlich. Zu beachten ist, daß die Möglichkeit zur erweiterten Adressierung für alle Registerpaare gegeben ist und daß die indirekten Register-Operationen, die den Kellerzeiger betreffen, die PUSH- und POP-Befehle sind. Die mnemotechnischen Abkürzungen für diese Befehle heißen "PUSH" und "POP". Der Unterschied zu gewöhnlichen Ladebefehlen besteht darin, daß der Kellerzeiger automatisch erniedrigt oder erhöht wird, wenn ein Byte gekellert oder aus dem Keller geholt wird.

Z. B. PUSH AF

ist ein Ein-Byte-Befehl mit dem OP-Kode F5H. Wenn dieser Befehl ausgeführt wird, ergibt sich folgender Ablauf:

```
Erniedrigen SP
LD (SP), A
Erniedrigen SP
LD (SP), F
```

Entsprechend dazu ist nun der externe Kellerspeicher aufgebaut:

(SP)	F	oberste Kellerebene
(SP + 1)	A	
	.	
	.	
	.	

Der POP-Befehl ist das exakte Gegenteil von PUSH. Zu beachten ist, daß alle POP- und PUSH-Befehle einen 16-Bit-Operanden benutzen, dessen höherwertiges Byte zuerst gekellert und zuletzt heraufgeholt wird.

D. h.

```
PUSH BC     heißt PUSH B dann C
PUSH DE     heißt PUSH D dann E
PUSH HL     heißt PUSH H dann L
POP  HL     heißt POP  L dann H
```

Der Befehl, der die erweiterte Direktwertadressierung als Quelle benutzt, benötigt natürlich 2 Datenbytes nach dem OP-Kode. Z. B.:

```
LD DE, 0659H
```

Maschinenkode:

Adresse A	11	OP-Kode
A + 1	59	unterer Teil des Operanden nach Reg. E
A + 2	06	oberer Teil des Operanden nach Reg. D

Bei allen erweiterten Adressierungsarten oder erweiterten Direktwert-Adressierungsarten erscheint das niederwertige Byte zuerst nach dem OP-Kode.

Tabelle 3 ist die Aufstellung der 16-Bit-Austauschbefehle, die in der CPU U880 implementiert sind.

Der OP-Kode 08H ermöglicht es dem Programmierer zwischen den zwei Paaren von Akkumulator und Flagregistern umzuschalten, wogegen der OP-Kode D9H die Umschaltung zum Duplikatsatz der 6 Allgebrauchsregister bewirkt. Diese OP-Kodes sind nur ein-byte-lang, um die Zeit, die für den Austausch benötigt wird, so weit wie möglich herabzusetzen, so daß der Duplikatsatz dazu benutzt werden kann, sehr kurze Interruptantwort-Zeiten zu realisieren.

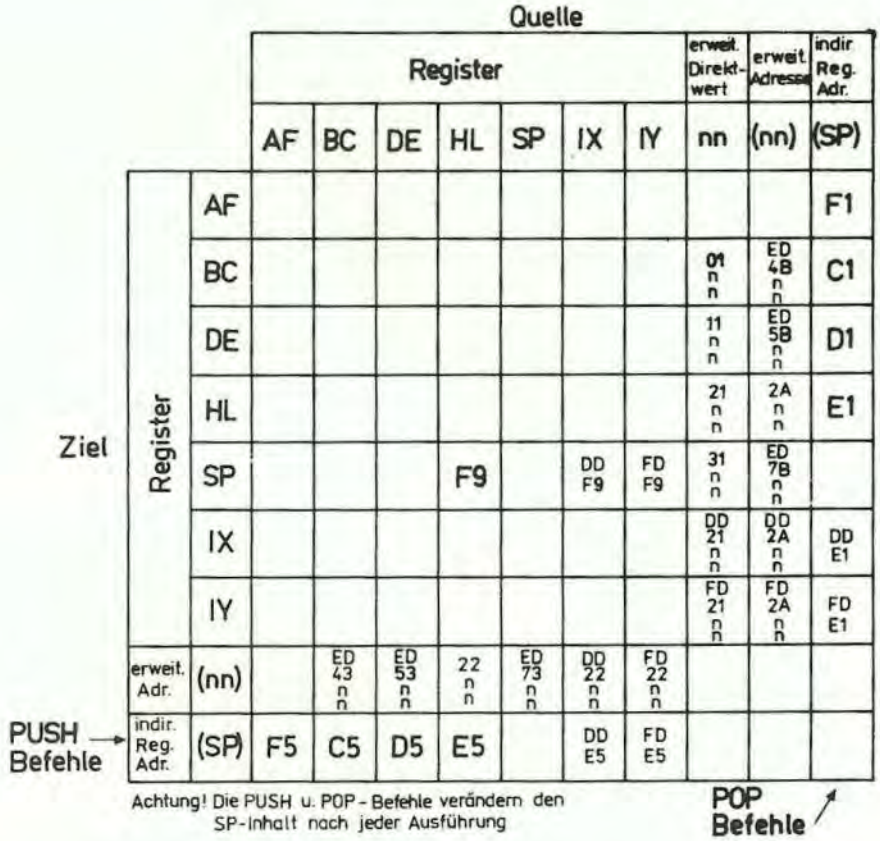


Tabelle 2: 16-Bit-Lade-Gruppe "LD" - "PUSH" - "POP"

		implizite Adressierung				
		AF'	BC',DE' u. HL'	HL	IX	IY
implizit	AF	08				
	BC DE u. HL		D9			
	DE			EB		
indir. Reg. Adr.	(SP)			E3	DD E3	FD E3

Tabelle 3: Tausch-Befehl "EX" - "EXX"

Blocktransporte und -suche

In Tabelle 4 sind die extrem leistungsfähigen Blocktransportbefehle aufgelistet. Alle diese Befehle arbeiten mit 3 Registern.

HL legt den Quellspeicherplatz fest
DE bestimmt die Zieladresse
BC ist der Byte-Zähler

Nachdem der Programmierer diese 3 Register mit den Anfangswerten geladen hat, kann einer der 4 Befehle benutzt werden. Der LDI-Befehl (Load and increment) transportiert ein Byte, das durch HL adressiert ist, zum Speicherplatz, der durch DE bestimmt wird. Die Registerpaare HL und DE werden dann automatisch erhöht. Damit sind sie bereit, die nächsten Speicherplätze anzugeben. Der Bytezähler (Registerpaar BC) wird in dieser Zeit auch automatisch erniedrigt. Dieser Befehl wird dann

verwendet, wenn Datenblöcke transportiert, und nach jedem Transportschritt noch andere Operationen durchgeführt werden müssen. Der LDIR-Befehl (Load, Increment and Repeat - Laden, Erhöhen und Wiederholen) ist eine Erweiterung des LDI-Befehls. Der gleiche Lade- und Erhöhungsvorgang wird solange wiederholt, bis der Bytezählerstand Null ist. Folglich kann dieser eine Befehl einen Datenblock von einem Speicherplatz zum anderen transportieren.
Achtung: Weil 16-Bit-Register verwendet werden, kann die Blocklänge bis zu 64 K Bytes lang sein (1 K = 1024). Der Block kann von einem beliebigen Speicherplatz zu einem beliebigen anderen transportiert werden. Außerdem können sich die Blöcke überlappen, weil es keinerlei Einschränkungen für die Daten in den drei Registerpaaren gibt.
Die LDD und LDDR-Befehle sind den LDI-und LDIR-Befehlen sehr ähnlich. Der einzige Unterschied besteht darin, daß die Registerpaare nach jedem Transport erniedrigt werden, so daß der Blocktransport bei der höchsten Adresse des jeweiligen Blocks beginnt, dann bis zur untersten fortgesetzt wird.

		Quelle: indir. Reg. Adr. M	
Ziel: indir. Reg. Adr.	(DE)	ED A0	"LDI"-Laden (DE)← M Erhöhen HL u. DE, Erniedrigen BC
		ED B0	"LDIR"-Laden (DE)← M Erhöhen HL u. DE, Erniedrigen BC, wdh. bis BC=0
		ED A8	"LDD" Laden (DE)← M Erniedrigen HL u. DE; Erniedrigen BC
		ED B8	"LDDR" Laden (DE)← M Erniedr. HL u. DE, Erniedr. BC, wdh. bis BC=0

Reg. HL - bezeichnet die Quelle Reg. DE - bezeichnet das Ziel
Reg. BC ist der Bytezähler

Tabelle 4: Blocktransport-Gruppe

Tabelle 5 spezifiziert die OP-Kodes für vier Blocksuchbefehle. Der erste, CPI (Compare and Increment - Vergleichen und Erhöhen) vergleicht die Daten im Akkumulator mit dem Inhalt des Speicherplatzes, der durch das HL-Register festgelegt ist. Das Ergebnis des Vergleichs wird in einem Flagbit gespeichert (s. Abschn. 6. - detaillierte Erklärung der Flagoperationen). Das HL-Registerpaar wird dann erhöht und der Bytezähler (Registerpaar BC) wird vermindert. Der Befehl CPIR ist nur eine Erweiterung des CPI-Befehls, in der Art, daß der Vergleich solange wiederholt wird, bis entweder der Vergleich erfüllt ist oder der Bytezähler (Registerpaar BC) zu Null wird. Folglich kann man mit diesem einen Befehl den ganzen Speicher nach einem 8-Bit-Zeichen durchsuchen.
Die CPD (Compare and Decrement) - und CPDR (Compare, Decrement and Repeat - Vergleichen, Vermindern und Wiederholen)-Befehle sind den eben genannten Befehlen sehr ähnlich. Der einzige Unterschied besteht darin, daß nach jedem Vergleich das HL-Registerpaar erniedrigt wird, so daß der Suchvorgang im Speicher in der entgegengesetzten Richtung abläuft. (Die Suche beginnt beim höchsten Speicherplatz im Speicherblock). Es soll nochmals hervorgehoben werden, daß diese Blocktransport- und -vergleichsbefehle für die Verarbeitung von Kettendaten außerordentlich leistungsfähig sind.

SUCH-Adresse

indir. Reg. Adr.	
M	
ED A1	"CPI" Erhöhen HL Erniedrigen BC
ED B1	"CPIR" Erhöhen HL, Erniedr. BC wdh. bis BC=0 oder bis Übereinstimmung gefunden wird
ED A9	"CPD" Erniedrigen von HL und BC
AD B9	"CPDR" Erniedrigen von HL und BC wdh. bis BC=0 oder bis Übereinstimmung gefunden wird

HL bezeichnet den Speicherplatz, der mit dem AC-Inhalt verglichen werden soll

BC ist der Bytezähler

Tabelle 5: Blocksuch - Gruppe

Arithmetik und Logik

In der Tabelle 6 sind alle arithmetischen 8-Bit-Operationen aufgeführt, die mit dem Akkumulator durchgeführt werden können. Die Tabelle enthält auch die Erhöhungs- und Erniedrigungsbefehle: INC (Increment) und DEC (Decrement). In allen diesen Befehlen mit Ausnahme von INC und DEC werden die spezifizierten 8-Bit-Operationen zwischen den Akkumulatordaten und den Quelldaten durchgeführt, die in der Tabelle aufgeführt sind. Das Ergebnis der Operation wird immer im Akkumulator untergebracht mit Ausnahme des Vergleiches (CP - Compare), der den Akkumulator unbeeinflußt läßt. Alle diese Operationen stellen auch das Flagregister entsprechend dem Ergebnis der jeweiligen Operation. (In Abschnitt 6. sind alle Details angegeben, wie die Flags durch jede Befehlsart beeinflußt werden). Die INC und DEC-Befehle legen ein Register oder einen Speicherplatz sowohl als Quelle als auch als Ziel des Ergebnisses fest.

Quelle

	Register - Adressierung							indir. Reg.-Adr.	indizierte Adressierung		Direktwert
	A	B	C	D	E	H	L	M	(IX-d)	(IY+d)	n
Addition ADD	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
Addition mit Übertrag ADC	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
Subtraktion SUB	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
Subtraktion mit Übertrag SBC	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
"UND" AND	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
exklusives oder XOR	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
oder OR	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
Vergleich CP	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
Erhöhen um 1 INC	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
Erniedr. um 1 DEC	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Tabelle 6: 8-Bit-Arithmetik und -Logik

Wenn zur Adressierung der Quelle des Operanden Indexregister benutzt werden, dann muß die Abstandsangabe unmittelbar folgen. Bei Direktwertadressierung folgt der Operand direkt.
Z. B. der Befehl:

AND 07H

Maschinenkode:

Adresse A	E6	OP-Kode
A + 1	07	Operand

Enthält der Akkumulator den Wert F3H, so würde nach der Operation im Akkumulator das Ergebnis 03H stehen:

Akkum. vor der Op.	1111 0011 = F3H
Operand	0000 0111 = 07H
Ergebnis im Akkum.	0000 0011 = 03H

Der Additionsbefehl (ADD) führt eine binäre Addition mit den Daten des Quellspeicherplatzes und den Daten im Akkumulator durch. Der Subtraktionsbefehl (SUB) bewirkt eine binäre Subtraktion. Wenn Addition mit Übertrag (ADC) oder Subtraktion mit Übertrag (SBC) durchgeführt wird, dann wird das Übertragsflag zusätzlich addiert bzw. subtrahiert. Die Flags und der Befehl zur Dezimalkorrektur (DAA) (vollständig im Abschnitt 6. beschrieben) ermöglichen in der CPU U880 arithmetische Operationen für:

mehrstellige gepackte BCD-Zahlen
mehrstellige Binärzahlen mit oder ohne Vorzeichen
mehrstellige Zahl mit Vorzeichen im 2er-Komplement.

Andere Befehle dieser Gruppe sind logische Befehle: "Und" (AND), logisches "Oder" (OR), exklusives Oder (XOR) und der Vergleich (CP),
Es gibt fünf arithmetische Befehle für allgemeine Verwendung, die mit dem Akkumulator oder dem Übertragsflag arbeiten. Diese fünf Befehle sind in der Tabelle 7 aufgelistet. Der Befehl zur Dezimalkorrektur kann sowohl die Subtraktion als auch die Addition korrigieren, so daß arithmetische BCD-Operationen vereinfacht werden. Zu beachten ist, daß für diese Operation das N-Flag verwendet wird. Dieses Flag wird gesetzt, wenn die letzte arithmetische Operation eine Subtraktion war. Der Befehl zur Negation des Akkumulators (NEG) bildet das Zweierkomplement der im Akkumulator befindlichen Zahl. Schließlich soll noch erwähnt werden, daß ein Löschbefehl für das Übertragsflag in der CPU U880 nicht vorgesehen wurde, weil sich diese Operation leicht durch einen anderen Befehl bewirken läßt. Z. B. logisches "Und" des Akkumulators mit sich selbst.

Einstellen der Dezimale des Akkumulators	DAA	27
Komplement des Akkumulators	CPL	2F
Vorzeichenumkehr des Akkumulators (Zweierkomplement)	NEG	ED 44
Komplement des Übertrags - Flags	CCF	3F
Setzen des Übertrags - Flags	SCF	37

Tabelle 7: AF-Operationen für allgemeinen Gebrauch

In Tabelle 8 sind alle arithmetischen 16-Bitoperationen zwischen den 16-Bit-Registern aufgelistet. Es gibt fünf Befehlsgruppen einschließlich der Addition mit Übertrag und der Subtraktion mit Übertrag. ADC und SBC beeinflussen alle Flags. Diese zwei Gruppen vereinfachen die Adressenrechnung oder andere arithmetische 16-Bitoperationen.

Ziel		Quelle BC	DE	HL	SP	IX	IY
ADD	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
Addition mit Übertrag und Flagsetzer ADC	HL	ED 4A	ED 5A	ED 6A	ED 7A		
Subtraktion mit Übertrag und Flagsetzer SBC	HL	ED 42	ED 52	ED 62	ED 72		
Erhöhen um 1 INC		03	13	23	33	DD 23	FD 23
Erniedrigen um 1 DEC		0B	1B	2B	3B	DD 2B	FD 2B

Tabelle 8: 16-Bit-Arithmetik

Verschiebung und zyklische Verschiebung

Eine Hauptleistung der CPU U880 ist die Möglichkeit, im Akkumulator, in einem Allgebrauchsregister oder an einem Speicherplatz Daten einfach oder zyklisch zu verschieben. Die Verschiebungs-OP-Kodes sind alle in der Tabelle 9 gezeigt. Die CPU U880 enthält auch arithmetische und logische Verschiebebefehle. Diese Operationen sind in einem sehr großen Gebiet von Anwendungen einschließlich der ganzzahligen Multiplikation und Division anwendbar. Zwei Befehle für die zyklische Verschiebung von BCD-Zahlen (RRD und RLD) ermöglichen es, eine Zahl im Akkumulator mit den zwei Zahlen eines Speicherplatzes auszutauschen, der durch das Registerpaar HL adressiert ist (siehe Bild 16). Diese Befehle gestatten eine effektive BCD-Arithmetik.

Quelle und Ziel

	A	B	C	D	E	H	L	M	(IX+d)	(IY+d)
RLC	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

Tabelle 9:
Verschiebung und
zyklische Verschiebung

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

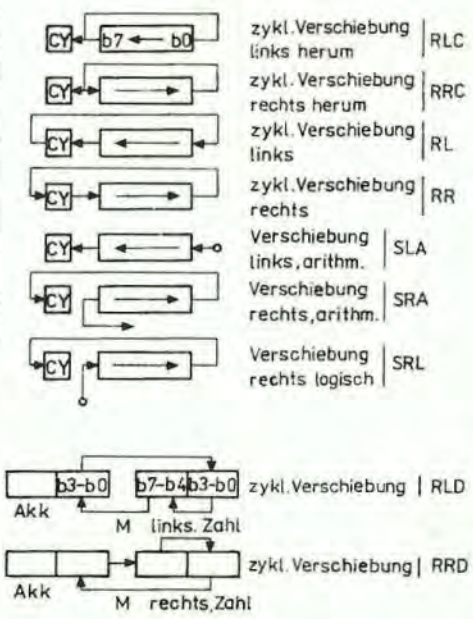


Bild: 16
Flußbilder für Verschiebung und
zyklische Verschiebung

Bit Manipulationen

Fast in jedem Programm wird die Möglichkeit genutzt, Bits in einem Register oder auf einem Speicherplatz zu setzen, zu löschen oder zu testen. Diese Bits können Flags in einer allgemein verwendbaren Software-Routine, Anzeigen von externen Steuerbedingungen oder gepackte Daten im Speicher sein, um die Speicherausnutzung effektiver zu gestalten.

		Registeradressierung							indir. Reg.	indiziert	
		A	B	C	D	E	H	L	M	(IX+d)	(IY+d)
TEST "BIT"	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
LÖSCHEN "RES"	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
BITSETZEN "SET"	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Die CPU U880 ist in der Lage, im Akkumulator, in einem Allgebrauchsregister oder an einem Speicherplatz mit einem einzigen Befehl ein Bit zu setzen, zu löschen oder zu testen. Die Tabelle 10 enthält die 240 Befehle, die für diesen Zweck zur Verfügung stehen. Die Registeradressierung kann sich auf den Akkumulator oder ein beliebiges Allgebrauchsregister beziehen, mit dem die Operation durchgeführt werden soll. Indirekte Register- und indizierte Adressierung stehen für die Arbeit mit den externen Speicherplätzen zur Verfügung. Die Bit-Testoperation setzt das Null-Flag (Z), wenn das getestete Bit eine Null ist (s. hierzu auch Abschn. 6).

Tabelle 10: Bit-Manipulations-Gruppe

Sprünge, Unterprogrammaufrufe und Rückkehr aus Unterprogrammen

Die Tabelle 11 stellt eine Liste aller Sprung-, Ruf- und Rückkehrbefehle dar, die in der CPU U880 implementiert sind. Ein Sprung ist eine Programmverzweigung, bei der der Befehlszähler mit dem 16-Bit-Wert geladen wird, der durch eine der drei möglichen Adressierungsarten festgelegt wird. (Erweiterter Direktwert, relative oder indirekte Registeradressierung). Zu beachten ist, daß die Befehle der Sprunggruppe die verschiedensten Bedingungen berücksichtigen können, die entsprechend der Spezifikation erfüllt sein müssen, bevor der Sprung ausgeführt wird. Wenn diese Bedingungen nicht erfüllt sind, wird das Programm mit dem Befehl fortgesetzt, der dem Sprungbefehl als nächster folgt. Die Bedingungen sind alle abhängig von den Daten im Flagregister (s. Abschn. 6. für eine genauere Beschreibung des Flagregisters). Die erweiterte Direktwertadressierung wird benutzt, um zu einem anderen Speicherbereich zu springen. Dieser Befehl erfordert drei Bytes (2 davon, um die 16-Bit-Adresse zu definieren), wobei das niederwertige Adressenbyte das erste ist, gefolgt von dem höherwertigen Adressenbyte.
Z. B. würde ein unbedingter Sprung zum Speicherplatz 3E32H wie folgt aussehen:

Adresse A	C3	OP-Kode
A + 1	32	niederwertiger Teil der Adresse
A + 2	3E	höherwertiger Teil der Adresse

Die Sprungbefehle mit relativer Adressierung benötigen nur zwei Bytes. Das zweite Byte ist das vorzeichenbehaftete Zweierkomplement der Distanz zwischen Zieladresse und dem aktuellen Wert des Befehlszählers. Diese Distanz kann sich in dem Bereich von + 127 bis - 128 bewegen (bezogen auf die Adresse des OP-Kodes des dem Sprungbefehl folgenden Befehls).

Sprungarten mit indirekter Registeradressierung sind auch vorhanden. Diese Befehle sind so implementiert, daß der Inhalt entweder des HL-Registerpaares oder eines der Indexregister IX und IY direkt in den Befehlszähler geladen wird. Damit sind Programmsprünge in Abhängigkeit von vorangegangenen Berechnungen möglich.

Der Unterprogrammaufruf (CALL) ist ein spezieller Sprungbefehl, bei dem die CPU die Adresse des auf den CALL-Befehl folgenden Bytes kellert, bevor der Sprung ausgeführt wird. Der Rückkehrbefehl ist das Gegenteil des CALL-Befehls, weil die Daten aus der obersten Ebene des Kellers direkt in den Befehlszähler geladen werden, um so die Sprungadresse zu bilden. Die Ruf- und Rückkehrbefehle vereinfachen die Unterprogramm- und Interruptbehandlung. Weiterhin sind zwei spezielle Rückkehrbefehle RETI und RETN in der U880-Bauelementenfamilie enthalten. Die Rückkehr von einem Interruptbefehl (RETI) und die Rückkehr von einem nicht maskierbaren Interruptbefehl (RETN) werden von der CPU wie unbedingte Rückkehrsprünge mit dem OP-Kode C9H behandelt. Der Unterschied besteht darin, daß RETI am Ende einer Interrupt-Routine aufgerufen werden kann, um alle peripheren Chips des Systems durch Dekodierung dieses Befehls an die richtige Steuerung der Interrupt-Behandlung mit gestufter Priorität zu erinnern. Dieser Befehl vereinfacht zusammen mit der Implementierung von peripheren Schaltkreisen des U880-Systems die normale Rückkehr von einem gestuften Interrupt. Ohne diese Implementierung wäre der folgende Software-Ablauf erforderlich, um die den Interrupt anfordernde Schaltung darüber zu informieren, daß die Interrupt-Routine abgearbeitet ist:

```
Abschalten der Interruptannahme - weitere Interrupts
.                                 abweisen, bis die
.                                 Routine beendet ist
.
LD A,          - der Peripherie mitteilen, daß
                   die Routine beendet ist
OUT n
Einschalten der Interruptannahme
Rückkehr
```

Diese Folge von 7 Bytes kann durch den Zwei-Byte-Befehl RETI der CPU U880 ersetzt werden. Das ist wichtig, weil oft die Interrupt-Behandlungszeit minimiert werden muß.
Für die Programmschleifensteuerung kann der Befehl DJNZ e vorteilhaft verwendet werden. Dieser zwei-byte-lange relative Sprungbefehl erniedrigt das Register B und es wird gesprungen, solange das Register B ungleich Null ist. Der relative Abstand wird als ein vorzeichenbehaftetes Zweierkomplement ausgedrückt.

Ein einfaches Beispiel könnte sein:

Adresse:	Befehl:	Kommentar:
N, N + 1	LD B, 7	; Setzen des Registers B, um von 7 abwärts zu zählen.
N + 2 ...N + 9	(Folge von abzuarbeitenden Befehlen)	Schleife soll 7mal durchlaufen ; werden.
N + 10, N + 11	DJNZ - 10	; Sprung von N+12 zu N+2
N + 12	(nächster Befehl)	

			Bedingung									
			unbedingt	Übertrag C	kein Übertr. NC	=0 Z	≠0 NZ	paarig PE	unpaarig PO	<0 M	>0 P	Reg. B≠0
Sprung "JMP" "J"	erweiterter Direktwert	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
Sprung "JR"	relativ	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
Sprung "JMP"	indirekt Register	M	E9									
Sprung "JMP"		(IX)	DD E9									
"JMP"		(IY)	FD E9									
"CALL" "CA"	erweiterter Direktwert	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
Emiedrigen B, Sp. bei ≠ 0 DJNZ	relativ	PC+e										10 e-2
Rücksprung "RET" "R"	indirekt Register	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
Rücksprung v. Interrupt RETI	indirekt Register	(SP) (SP+1)	ED 4D									
Rückspr. v. nicht mask. Interrupt RETN	indirekt Register	(SP) (SP+1)	ED 45									

Achtung: Gewisse Flags haben mehr als eine Bedeutung (siehe Abschn. 6.)

Tabelle 11: SPRUNG, RUF und Rückkehrgruppe

Die Tabelle 12 zeigt die 8 OP-Kodes für den RESTART-Befehl. Dieser Befehl ist ein 1-Byte-Ruf zu einer der 8 angegebenen Adressen.
Die mnemotechnischen Abkürzungen für diese 8 Unterprogrammaufrufe sind ebenfalls angegeben. Der Wert dieses Befehls besteht darin, daß ständig benutzte Unterprogramme mit diesem Befehl speichersparend aufgerufen werden können.

		OP KODE		
CALL ADRESSE	0000_H	C7	RST	0
	0008_H	CF	RST	8
	0010_H	D7	RST	16
	0018_H	DF	RST	24
	0020_H	E7	RST	32
	0028_H	EF	RST	40
	0030_H	F7	RST	48
	0038_H	FF	RST	56

Tabelle 12: RESTART-Gruppe

Ein/Ausgabe

Die CPU U880 hat einen ausgebauten Satz von Ein- und Ausgabebefehlen, die in den Tabellen 13 und 14 aufgelistet sind. Die Adressierung der Ein- oder Ausgabeschaltungen kann entweder absolut oder über das Register C indirekt erfolgen. Zu beachten ist, daß die Daten bei der indirekten Registeradressierung zwischen den E/A-Schaltungen und einem der internen Register transportiert werden können. Zusätzlich sind 8 Blockübertragungsbefehle implementiert worden. Diese Befehle sind den Speicherblock-Transportbefehlen ähnlich, mit dem Unterschied, daß sie das Registerpaar HL für die Angabe der Quelladresse (bei Ausgabebefehlen) oder der Zieladresse (bei Eingabebefehlen) benutzen. Das Register B wird als Bytezähler verwendet. Das Register C enthält die Adresse des Kanals, für den der Ein- oder Ausgabebefehl gewünscht ist. Weil das Register B nur eine Länge von 8 Bits hat, kann ein E/A-Block-Übertragungsbefehl nur 256 Bytes verarbeiten.
In den Befehlen IN n und OUT n erscheint die Adresse der E/A-Schaltung in der unteren Hälfte des Adressenbusses (A0-A7) und der Akkumulatorinhalt wird in die obere Hälfte des Adressenbusses überführt. In allen E/A-Befehlen mit indirekter Registeradressierung einschließlich der E/A-Blockübertragung wird der Inhalt des Registers C in die untere Hälfte des Adressenbusses überführt (Kanaladresse) wogegen der Inhalt des Registers B in die obere Hälfte des Adressenbusses überführt wird.

Eingabe Ziel			Kanaladresse Direktwert n	Kanaladresse Indir. Reg. c	
Eingabe IN	REG. ADRESSIERUNG	A	DB n	ED 78	
		B		ED 40	
		C		ED 48	
		D		ED 50	
		E		ED 58	
		H		ED 60	
		L		ED 68	
		F		ED 70	
Eingabe + Erh. HL Ern. B INI	indirekt Register	M		ED A2	Block - Eingabe - Befehle
Eing. Erh. HL, Ern. B.wdn. - B=0 INIR				ED B2	
Eing. + Ern. HL + Ern. B IND				ED AA	
Eingabe + Ern. HL + Ern. B wdh - B = 0 INDR				ED BA	

Tabelle 13: Eingabe-Gruppe

			Quelle							
			Register							indir. Reg.
			A	B	C	D	E	H	L	M
OUT	Direkt-wert	n	D3 n							
	indir. Reg.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
Ausgabe+Erh. HL+ Ern. B OUTI	indir. Reg.	(C)								ED A3
Ausgabe + Ern. HL+ Ern. B wdh.-B=0 OUTIR	indir. Reg.	(C)								ED B3
Ausgabe+Ern. HL+B OUTD	indir. Reg.	(C)								ED AB
Ausgabe + Ern. HL+B wdh.-B=0 OUTDR	indir. Reg.	(C)								ED BB

Block-Ausgabe-Befehle

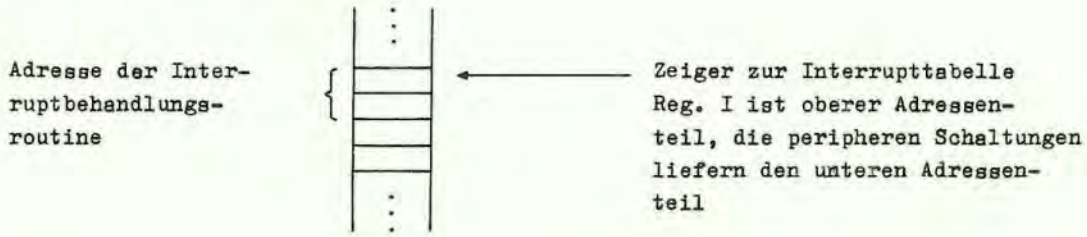
Ziel-Kanaladresse

Tabelle 14: Ausgabe-Gruppe

CPU-Steuer-Gruppe

Tabelle 15 zeigt die 6 CPU-Steuerbefehle für allgemeine Verwendung. Der NOP ist ein Leerbefehl, bei dem keine Operation ausgeführt wird. Der HALT-Befehl unterbricht die Arbeit der CPU, bis ein nachfolgender Interrupt empfangen wird. Die Befehle DI und EI dienen dazu, Interrupts abzuweisen oder zuzulassen. Die drei Interruptartbefehle setzen die CPU in eine der drei möglichen Interruptantwortmoden. Wenn die Mode 0 gesetzt ist, kann die den Interrupt anfordernde Schaltung einen Befehl auf den Datenbus ausgeben und der CPU gestatten, ihn auszuführen. Die Mode 1 ist eine vereinfachte Betriebsart, bei der die CPU automatisch einen RESTART-Befehl zum Speicherplatz 38H ausführt, so daß keine zusätzliche externe Hardware erforderlich ist. (Der alte Befehlszählerstand wird gekellert). Mode 2 ist sehr leistungsfähig, weil sie einen indirekten Aufruf eines Speicherplatzes zuläßt. Bei dieser Betriebsart bildet die CPU die 16-Bit-Speicheradresse, indem sie für die oberen 8 Bits den Inhalt des Registers I verwendet und die unteren 8 Bits durch die den Interrupt fordernde Schaltung geliefert werden. Diese Adresse bezeichnet das erste von zwei aufeinander folgenden Bytes in einer Tabelle, in der die Adressen der Behandlungsroutinen gespeichert sind. Die CPU übernimmt automatisch die Startadresse und führt einen Unterprogrammaufruf zu dieser Adresse durch.

Prinzip der Interrupttabellenadressierung:



NOP		00
HALT		76
INT abweisen	DI	F3
INT annehmen	EI	FB
Setzen INT mode 0	IM0	ED 46
Setzen INT mode 1	IM1	ED 56
Setzen INT mode 2	IM2	ED 5E

8080 A-Mode

CALL zu Adresse 0038H

Indirektes CALL unter Verwendung von Reg. I u. 8 Bit vom INT auslösenden Gerät als Zeiger

Tabelle 15: Verschiedene CPU-Steuerbefehle

6. Flags

Jedes der zwei U880-Flagregister enthält 6 Informationsbits, die durch verschiedene CPU-Operationen beeinflußt werden können. Vier dieser Bits können getestet werden, d. h., sie werden als Bedingung für Sprünge, UP-Rufe oder Rückkehrbefehle verwendet. Z. B. kann ein Sprung erwünscht sein, wenn ein spezielles Bit im Flagregister gesetzt ist. Die vier abfragbaren Flagbits sind:

BIT 0 Übertrag-Flag - Dieses Flag ist der Übertrag vom Akkumulatorbit mit der höchsten Wertigkeit. Z. B. wird das Übertragsflag während eines Additionsbefehls gesetzt, wenn ein Übertrag vom höchsten Bit des Akkumulators erzeugt wird. Dieses Flag wird auch gesetzt, wenn bei einer Subtraktion ein negativer Übertrag entsteht. Die einfachen und zyklischen Verschiebebefehle beeinflussen dieses Bit ebenfalls.

BIT 6 Null-Flag - Dieses Flag wird gesetzt, wenn im Ergebnis einer Operation im Akkumulator eine Null geladen wird. Anderenfalls wird das Bit gelöscht.

BIT 7 Flag für negatives Vorzeichen - Dieses Flag ist gedacht für die Verarbeitung von Zahlen mit Vorzeichen. Das Flag wird gesetzt, wenn das Ergebnis der Operation negativ war. Weil das Bit 7 (MSB) das Vorzeichen der Zahl ist (eine negative Zahl trägt eine 1 im Bit 7), speichert dieses Flag den Zustand des Bits 7 des Akkumulators.

BIT 2 Paritäts/Überlauf-Flag - Dieses Flag hat zwei Aufgaben, indem es einmal die Parität des Ergebnisses im Akkumulator anzeigt, wenn logische Operationen durchgeführt wurden (z. B. AND, B), und zum anderen einen Überlauf anzeigt, wenn arithmetische Operationen mit vorzeichenbehafteten Zweierkomplementen durchgeführt wurden. Das U880-Überlaufflag signalisiert, daß ein solches Zweierkomplement fehlerhaft ist, weil es den zulässigen Bereich von max. + 127 oder von min. - 128, der in der Zweierkomplementschreibweise dargestellt werden kann, überschreitet. Betrachten wir beispielsweise folgende Addition:

```
+ 120 = 0111 1000
+ 105 = 0110 1001
  C     1110 0001 = - 95 (falsch!) Überlauf ist aufgetreten!
```

Hier ist das Ergebnis falsch. Es ist zwar ein Überlauf aufgetreten, aber kein Übertrag, aus dem man den Fehler erkennen könnte.
In diesem Falle würde aber das Überlauf-Flag gesetzt. Betrachten wir auch die Addition zweier negativer Zahlen:

```
-   5 = 1111 1011
-  16 = 1111 0000
  C   =1 1110 1011 = - 21 richtig
```

Diese Antwort ist richtig, aber das Übertrag-Flag wird gesetzt, so daß dieses Flag in keinem Falle als Überlaufanzeige benutzt werden kann.
Bei logischen Operationen (AND, OR, XOR) wird dieses Flagbit 2 gesetzt, wenn das Ergebnis paarig ist und es wird gelöscht, wenn das Ergebnis unpaarig ist.

Weiterhin sind noch 2 nicht abfragbare Bits im Flagregister vorhanden. Beide werden für die BCD-Arithmetik benutzt. Diese Bits sind:

BIT 4 Halbbyte-Übertrag - Das ist das BCD-Übertragsergebnis (Add. oder Subtr.) der letzten 4 signifikanten Bits der Operation. Wenn der DAA (Dezimalkorrektur) -Befehl benutzt wird, dient dieses Flag dazu, das Ergebnis der vorausgegangenen gepackten Dezimaladdition oder -subtraktion zu korrigieren.

Bit 1 (N) Subtraktions-Flag - Weil der Algorithmus für die Korrektur der BCD-Operationen für Addition und Subtraktion unterschiedlich ist, dient dieses Flag der Anzeige, der Befehlsart, die ausgeführt wurde, so daß die DAA-Operation die Korrektur entweder für Addition oder Subtraktion durchführen kann.

Der Programmierer hat Zugriff zum Flag-Register. Es hat folgendes Format:

7							0
S	Z	X	H	X	P/V	N	C

X: Bit hat keine Bedeutung

In Tabelle 16 ist zusammengestellt, wie jedes Flag durch die verschiedenen Befehle beeinflußt wird. In dieser Tabelle bedeutet ein • , daß dieser Befehl das Flag nicht beeinflußt. Ein X heißt, daß das Flag einen unbestimmten Zustand annimmt. Eine 0 bedeutet, das Flag wird gelöscht, eine 1, das Flag wird gesetzt. Das Symbol ↕ bedeutet, daß das Flag entsprechend der vorangegangenen Beschreibung gesetzt wird.
Ein Befehl, der in dieser Liste nicht erscheint, beeinflußt kein Flag.

Befehl	Flags C	Z	P/V	S	N	H	Bemerkungen
ADD s, ADC s	↕	↕	V	↕	0	↕	8-Bit-Addition oder Addition mit Übertrag
SUB s, SBC s, CMP s, NEG	↕	↕	V	↕	1	↕	8-Bit-Subtraktion, Subtraktion mit Übertrag, Vergleich und Negation des Akkumulators
AND s	0	↕	P	↕	0	1	Logische Operationen
OR s, XOR s	0	↕	P	↕	0	0	
INC s	•	↕	V	↕	0	↕	8-Bit-Erhöhung
DEC s	•	↕	V	↕	1	↕	8-Bit-Erniedrigung
ADD HL, dd	↕	•	•	•	0	X	16-Bit-Addition
ADC HL, dd	↕	↕	V	↕	0	X	16-Bit-Addition mit Übertrag
SBC HL, dd	↕	↕	V	↕	1	X	16-Bit-Subtraktion mit Übertrag
RLA, RLCA, RRA, RRCA	↕	•	•	•	0	0	zyklische Verschiebung - Akkumulator
RL s, RLC s, RR s, RRC s	↕	↕	P	↕	0	0	zyklische Verschiebung - Speicherplatz s
SLA s, SRA s, SRL s	↕	↕	P	↕	0	0	Verschiebung - Speicherplatz s
RLD, RRD	•	↕	P	↕	0	0	zyklische Verschiebung - Zahl links und rechts
DAA	↕	↕	P	↕	•	↕	Dezimalen Einrichtung - Akkumulator
CPL	•	•	•	•	1	1	Komplement des Akkumulators
SCF	1	•	•	•	0	0	Setzen des Übertrags
CCF	↕	•	•	•	0	X	Komplement des Übertrags
IN r, INF	•	↕	P	↕	0	0	Eingabe, indirekte Registeradresse
INI, IND, OUTI, OUTD	•	↕	X	X	1	X	Block-Ein- und Ausgabe, Z = 0 wenn B ≠ 0, sonst Z = 1
INIR, INDR, OTIR, OTDR	•	1	X	X	1	X	Z = 0 wenn B ≠ 0, sonst Z = 1
LDI, LDD	•	X	↕	X	0	0	Blocktransfer-Befehle
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 wenn BC≠0, sonst P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	X	1	X	Block-Such-Befehle, Z=1 wenn A=(HL), sonst Z=0; P/V=1 wenn BC≠0, sonst P/V=0
LD A, I; LD A, R	•	↕	IFF2	↕	0	0	Inhalt des Interrupt-Annahme-Flip-Flops 2 (IFF2) ins P/V-Flag überführt
BIT b,s	•	↕	X	X	0	1	Zustand des Bits b im Speicherplatz s ins Z-Flag überführt

Tabelle 16: Zusammenfassung der Flag-Operationen

Zeichenerklärung zur Tabelle 16

Symbol	Bedeutung
C	Übertragsflag. C = 1, wenn die Operation einen Übertrag vom MSB des Operanden oder des Ergebnisses erzeugt.
Z	Null-Flag. Z = 1, wenn das Ergebnis der Operation Null ist.
S	Vorzeichen-Flag. S = 1, wenn das MSB des Ergebnisses eins ist.
P/V	Paritäts- oder Überlauf-Flag. Parität (P) und Überlauf (V) benutzen das gleiche Flag. Logische Operationen beeinflussen das Flag entsprechend der Parität des Ergebnisses, arithmetische Operationen stellen dieses Flag entsprechend dem Überlauf des Ergebnisses. P/V = 1, wenn das Ergebnis paarig ist, P/V = 0, wenn das Ergebnis unpaarig ist. P/V = 1, wenn das Ergebnis einen Überlauf enthält.
H	Halbbyte-Übertragsflag. H = 1, wenn Addition oder Subtraktion einen Übertrag innerhalb von 4 Akkumulatorbits erzeugen.
N	Additions-/Subtraktionsflag. N = 1, wenn vorangegangene Operation eine Subtraktion war. H- und N-Flags werden für die Dezimalkorrektur (DAA) benutzt, um das Ergebnis einer Addition oder Subtraktion von gepackten BCD-Zahlen in das Format gepackter BCD-Zahlen zu wandeln.
↕	Flag wird entsprechend dem Ergebnis der Operation gestellt
•	Flag wird durch die Operation nicht beeinflußt
0	Flag wird durch die Operation gelöscht
1	Flag wird durch die Operation gesetzt
X	Flag unbestimmt
V	P/V-Flag entspricht dem Ergebnis-Überlauf der Operation
P	P/V-Flag entspricht der Parität des Ergebnisses der Operation
r	eines der U880 D-Register A, B, C, D, E, H, L.
s	ein 8-Bit-Speicherplatz, der durch eine der für den jeweiligen Befehl zulässigen Adressierungsarten definiert ist.
dd	ein 16-Bit-Speicherplatz, der durch eine der für diesen Befehl zulässigen Adressierungsarten definiert ist.
ii	eines der zwei Indexregister IX oder IY
R	Auffrischzähler
n	8-Bit im Bereich 0 - 255
nn	16-Bit im Bereich 0 - 655 35

In der Tabelle 16 sind auch einige Spezialfälle enthalten, die der Klarheit wegen beschrieben werden müssen. Der Blocksuchbefehl setzt das Z-Flag, wenn bei der letzten Vergleichsoperation eine Übereinstimmung zwischen den Speicherdaten und dem Akkumulator gefunden wird. Es wird auch das Paritätsflag gesetzt, wenn der Bytezähler (Register BC) ungleich Null ist. Gleichermaßen wird auch das Paritätsflag in den Blocktransportbefehlen verwendet. Ein anderer Spezialfall tritt während der Blockein- und Ausgabebefehle auf. Hier wird das Z-Flag benutzt, um den Zustand des Registers B anzuzeigen, das als Bytezähler benutzt wird. Wenn der E/A-Blocktransport beendet ist, wird das Null-Flag auf Null gelöscht (d. h. B = 0), während im Falle des Blocktransportbefehls das Paritätsflag zum Ende der Operation gelöscht wird. Ein letzter Spezialfall tritt auf, wenn das Auffrisch- oder I-Register in den Akkumulator geladen wird. Dann wird gleichzeitig das Interrupt-Annahme-Flip-Flop in das Paritätsflag geladen, so daß der vollständige Zustand der CPU jederzeit gerettet werden kann.

7. Zusammenfassung der OP-Kodes und der Ausführungszeiten

Der folgende Abschnitt enthält die Zusammenfassung des U 880-Befehlssatzes. Die Befehle sind logisch zu Gruppen zusammengefaßt, wie das aus der Tabelle 17 zu erkennen ist. Jede Tabelle enthält die mnemotechnische Abkürzung der Assemblersprache, den aktuellen Operations-Kode, die symbolische Operation, den Inhalt des Flag-Registers, der nach der Ausführung des jeweiligen Befehls auftritt, die Zahl der für jede Operation erforderlichen Bytes und Speicherzyklen sowie die Gesamtzahl der T-Zyklen (externe Takte), die für den Aufruf und die Ausführung jedes Befehls erforderlich sind.

Tabelle 17

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	By-tes	M-Zyk-len	Takte	Bemerkung
8 - Bit - Ladegruppe							
LD r_1,r_2	$r_1 \leftarrow r_2$	. . . . .	01 r_1 r_2	1	1	4	r_1,r_2 | Register 000 | B 001 | C 010 | D 011 | E 100 | H 101 | L 111 | A r_1,r_2 steht für eines der Register A,B,C,D,E,H,L.
LD r,n	r←n	. . . . .	00 r 110 - n -	2	2	7	
LD r,M	r← M	. . . . .	01 r 110	1	2	7	
LD r,(IX+d)	r←(IX+d)	. . . . .	11 011 101 01 r 110 - d -	3	5	19	
LD r,(IY+d)	r←(IY+d)	. . . . .	11 111 101 01 r 110 - d -	3	5	19	
LD M,r	M ←r	. . . . .	01 110 r	1	2	7	
LD (IX+d),r	(IX+d)←r	. . . . .	11 011 101 01 110 r - d -	3	5	19	
LD (IY+d),r	(IY+d)←r	. . . . .	11 111 101 01 110 r - d -	3	5	19	
LD M,n	M ←n	. . . . .	00 110 110 - n -	2	3	10	
LD (IX+d),n	(IX+d)←n	. . . . .	11 011 101 00 110 110 - d - - n -	4	5	19	
LD (IY+d),n	(IY+d)←n	. . . . .	11 111 101 00 110 110 - d - - n -	4	5	19	
LD A,(BC)	A←(BC)	. . . . .	00 001 010	1	2	7	
LD A,(DE)	A←(DE)	. . . . .	00 011 010	1	2	7	
LD A,(nn)	A←(nn)	. . . . .	00 111 010 - n - - n -	3	4	13	
LD (BC),A	(BC)←A	. . . . .	00 000 010	1	2	7	
LD (DE),A	(DE)←A	. . . . .	00 010 010	1	2	7	
LD (nn),A	(nn)←A	. . . . .	00 110 010 - n - - n -	3	4	13	
LD A,I	A←I	. ↕ IFF ↕ 0 0	11 101 101 01 010 111	2	2	9	
LD A,R	A←R	. ↕ IFF ↕ 0 0	11 101 101 01 011 111	2	2	9	
LD I,A	I←A	. . . . .	11 101 101 01 000 111	2	2	9	
LD R,A	R←A	. . . . .	11 101 101 01 001 111	2	2	9	
16 - Bit - Ladegruppe							
LD dd,nn	dd←nn	. . . . .	00 dd0 001 - n - - n -	3	3	10	dd | Paar 00 | BC 01 | DE 10 | HL 11 | SP
LD IX,nn	IX←nn	. . . . .	11 011 101 00 100 001 - n - - n -	4	4	14	

Fortsetzung der Tabelle 1

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operationscode 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
LD IY,nn	IY←nn	. . . . .	11 111 101 00 100 001 - n - - n -	4	4	14	
LD HL,(nn)	H←(nn+1) L←(nn)	. . . . .	00 101 010 - n - - n -	3	5	16	
LD dd,(nn)	ddH←(nn+1) ddL←(nn)	. . . . .	11 101 101 01 dd1 011 - n - - n -	4	6	20	
LD IX,(nn)	IXH←(nn+1) IXL←(nn)	. . . . .	11 011 101 00 101 010 - n - - n -	4	6	20	
LD IY,(nn)	IYH←(nn+1) IYL←(nn)	. . . . .	11 111 101 00 101 010 - n - - n -	4	6	20	
LD (nn),HL	(nn+1)←H (nn)←L	. . . . .	00 100 010 - n - - n -	3	5	16	
LD (nn),dd	(nn+1)←ddH (nn)←ddL	. . . . .	11 101 101 01 dd0 011 - n - - n -	4	6	20	dd ist eines der Registerpaare BC,DE,HL,SP
LD (nn),IX	(nn+1)←IXH (nn)←IXL	. . . . .	11 011 101 00 100 010 - n - - n -	4	6	20	
LD (nn),IY	(nn+1)←IYH (nn)←IYL	. . . . .	11 111 101 00 100 010 - n - - n -	4	6	20	
LD SP,HL	SP←HL	. . . . .	11 111 001	1	1	6	
LD SP,IX	SP←IX	. . . . .	11 011 101 11 111 001	2	2	10	
LD SP,IY	SP←IY	. . . . .	11 111 101 11 111 001	2	2	10	
PUSH qq	(SP-2)←qqL (SP-1)←qqH SP←SP-2	. . . . .	11 qq0 101	1	3	11	qq | Paar 00 | BC 01 | DE 10 | HL 11 | AF
PUSH IX	(SP-2)←IXL (SP-1)←IXH SP←SP-2	. . . . .	11 011 101 11 100 101	2	4	15	
PUSH IY	(SP-2)←IYL (SP-1)←IYH SP←SP-2	. . . . .	11 111 101 11 100 101	2	4	15	qq ist eines der Registerpaare AF,BC,DE,HL.
POP qq	qqH←(SP+1) qqL←(SP) SP←SP+2	. . . . .	11 qq0 001	1	3	10	(Paar)H bzw. (Paar)L bezieht sich auf die oberen bzw. unteren 8 Bits des entsprechenden Registerpaars, d.h. BCL=C, AFH=A.
POP IX	IXH←(SP+1) IXL←(SP) SP←SP+2	. . . . .	11 011 101 11 100 001	2	4	14	
POP IY	IYH←(SP+1) IYL←(SP) SP←SP+2	. . . . .	11 111 101 11 100 001	2	4	14	
Austausch-, Blocktransfer- und Suchgruppe [1]							
EX DE,HL	DE↔HL	. . . . .	11 101 011	1	1	4	
EXAF	AF↔AF'	. . . . .	00 001 000	1	1	4	
EXX	(BC, DE, HL) ↔ (BC', DE', HL')	. . . . .	11 011 001	1	1	4	Vertauschung Registersatz/Alternativregistersatz
EX (SP),HL	H↔(SP+1) L↔(SP)	. . . . .	11 100 011	1	5	19	
EX (SP),IX	IXH↔(SP+1) IXL↔(SP)	. . . . .	11 011 101 11 100 011	2	6	23	
EX (SP),IY	IYH↔(SP+1) IYL↔(SP)	. . . . .	11 111 101 11 100 011	2	6	23	
LDI	(DE)← M DE←DE+1 HL←HL+1 BC←BC-1	. X ↕ A X 0 0	11 101 101 10 100 000	2	4	16	

1) A:P/V-Flag ist 0, wenn das Ergebnis von BC-1=0, sonst P/V=1
B:Z-Flag ist 1, wenn A=(M), sonst Z=0

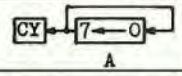
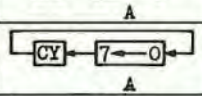
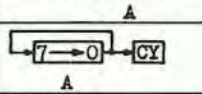
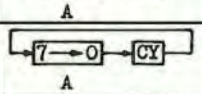
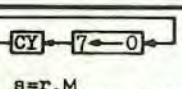
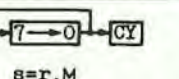
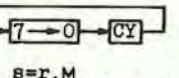
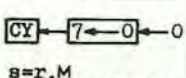
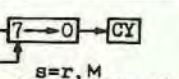
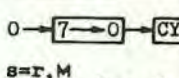
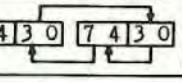
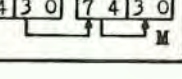
Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
LDIR	(DE)← M DE←DE+1 HL←HL+1 BC←BC-1 Wiederholung bis BC=0	. X 0 X 0 0	11 101 101 10 110 000	2 2	5 4	21 16	wenn BC≠0 wenn BC=0
LDD	(DE)← M DE←DE-1 HL←HL-1 BC←BC-1	. X ↕ X 0 0 A	11 101 101 10 101 000	2	4	16	
LDDR	(DE)← M DE←DE-1 HL←HL-1 BC←BC-1 Wiederholung bis BC=0	. X 0 X 0 0	11 101 101 10 111 000	2 2	5 4	21 16	wenn BC≠0 wenn BC=0
CPI	A— M HL←HL+1 BC←BC-1	. ↕ ↕ X 1 X B A	11 101 101 10 100 001	2	4	16	
CPIR	A— M HL←HL+1 BC←BC-1 Wiederholung bis BC=0 oder A=M	. ↕ ↕ X 1 X B A	11 101 101 10 110 001	2 2	5 4	21 16	wenn BC≠0 und A≠M wenn BC=0 oder A=M
CPD	A— M HL←HL-1 BC←BC-1	. ↕ ↕ X 1 X B A	11 101 101 10 101 001	2	4	16	
CPDR	A— M HL←HL-1 BC←BC-1 Wiederholung bis BC=0 oder A=M	. ↕ ↕ X 1 X B A	11 101 101 10 111 001	2 2	5 4	21 16	wenn BC≠0 und A≠M wenn BC=0 oder A=M
8 Bit-Arithmetik und logische Gruppe [2]							
ADD r	A←A+r	↕ ↕ V ↕ 0 ↕	10 [000] r	1	1	4	
ADD n	A←A+n	↕ ↕ V ↕ 0 ↕	11 [000] 110 - n -	2	2	7	
ADD M	A←A+M	↕ ↕ V ↕ 0 ↕	10 [000] 110	1	2	7	
ADD (IX+d)	A←A+(IX+d)	↕ ↕ V ↕ 0 ↕	11 011 101 10 [000] 110 - d -	3	5	19	
ADD (IY+d)	A←A+(IY+d)	↕ ↕ V ↕ 0 ↕	11 111 101 10 [000] 110 - d -	3	5	19	
ADC s	A←A+s+CY	↕ ↕ V ↕ 0 ↕	[001]				s ist eines der r, n, M ,(IX+d),(IY+d) wie beim ADD-Befehl. umrandete Bits ersetzen 000 in ADD-Befehl.
SUB s	A←A-s	↕ ↕ V ↕ 1 ↕	[010]				
SBC s	A←A-s-CY	↕ ↕ V ↕ 1 ↕	[011]				
AND s	A←A∧s	0 ↕ P ↕ 0 1	[100]				
OR s	A←A∨s	0 ↕ P ↕ 0 0	[110]				
XOR s	A←A•s	0 ↕ P ↕ 0 0	[101]				
CMP s	A - s	↕ ↕ V ↕ 1 ↕	[111]				
INC r	r←r+1	. ↕ V ↕ 0 ↕	00 r [100]	1	1	4	
INC M	M←M+1	. ↕ V ↕ 0 ↕	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d)← (IX+d)+1	. ↕ V ↕ 0 ↕	11 011 101 00 110 [100] - d -	3	6	23	

r	Register
000	B
001	C
010	D
011	E
100	H
101	L
111	A

2) Das V-Symbol in der P/V-Flag-Spalte bedeutet, daß das P/V-Flag den Überlauf des Ergebnisses einer Operation enthält. Ebenso weist das Symbol P auf die Parität hin. V=1 bedeutet Überlauf, V=0 bedeutet kein Überlauf. P=1 bedeutet, das Ergebnis ist paarig; P=0 bedeutet, das Ergebnis ist unpaarig.

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operationscode 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
INC (IY+d)	(IY+d)←(IY+d)+1	↕ V ↕ 0 ↕	11 111 101 00 110 100 - d -	3	6	23	
DEC f	f←f-1	↕ V ↕ 1 ↕	101				f ist eines der r, M, (IX+d), (IY+d) wie bei INC; gleiches Format und Zustände wie INC. 100 durch 101 im Operationscode ersetzen.
Allgemeine Arithmetik und U 880 D-Steuergruppe							
DAA	Wandelt AC-Inhalt in gepacktes BCD-Format nach Add oder Subtraktion von gepackten BCD-Zahlen	↕ ↕ P ↕ ↕	00 100 111	1	1	4	Dezimalkorrektur im Akkumulator
CPL	A←$\overline{A}$	. . . . 1 1	00 101 111	1	1	4	Komplement des Akkumulators; Einerkomplement
NEG	A←$\overline{A}$+1	↕ ↕ V ↕ 1 ↕	11 101 101 01 000 100	2	2	8	Negation AC; Zweierkomplement
CCF	CY←$\overline{CY}$	↕ . . . 0 X	00 111 111	1	1	4	3); Komplement des Übertrags-Flag
SCF	CY←1	1 . . . 0 0	00 110 111	1	1	4	3); Setzen des Übertrags-Flag
NOP	keine Operation	. . . . .	00 000 000	1	1	4	
HALT	U 880 D im HALT-Zustand	. . . . .	01 110 110	1	1	4	
3) CY ist das Übertrags-Flag							
DI	IFF1←0, IFF2←0	. . . . .	11 110 011	1	1	4	4)
EI	IFF1←1, IFF2←1	. . . . .	11 111 011	1	1	4	
IM0	Setzen der Interrupt-Mode 0	. . . . .	11 101 101 01 000 110	2	2	8	
IM1	Setzen der Interrupt-Mode 1	. . . . .	11 101 101 01 010 110	2	2	8	
IM2	Setzen der Interrupt-Mode 2	. . . . .	11 101 101 01 011 110	2	2	8	
16-Bit-Arithmetik							
ADD HL,dd	HL←HL+dd	↕ . . . 0 X	00 dd1 001	1	3	11	dd | Register 00 | BC 01 | DE 10 | HL 11 | SP dd ist eines der Registerpaare: BC, DE, HL, SP
ADC HL,dd	HL←HL+dd+CY	↕ ↕ V ↕ 0 X	11 101 101 01 dd1 010	2	4	15	
SBC HL,dd	HL←HL-dd-CY	↕ ↕ V ↕ 1 X	11 101 101 01 dd0 010	2	4	15	
ADD IX,pp	IX←IX+pp	↕ . . . 0 X	11 011 101 00 pp1 001	2	4	15	pp | Register 00 | BC 01 | DE 10 | IX 11 | SP pp ist eines der Registerpaare: BC, DE IX, SP
ADD IY,pp	IY←IY+pp	↕ . . . 0 X	11 111 101 00 pp1 001	2	4	15	pp | Register 00 | BC 01 | DE 10 | IY 11 | SP pp ist eines der Registerpaare: BC, DE, IY, SP
INC dd	dd←dd+1	. . . . .	00 dd0 011	1	1	6	
INC IX	IX←IX+1	. . . . .	11 011 101 00 100 011	2	2	10	
INC IY	IY←IY+1	. . . . .	11 111 101 00 100 011	2	2	10	
DEC dd	dd←dd-1	. . . . .	00 dd1 011	1	1	6	

4) IFF1 ist das Interrupt-Annahme-Flip-Flop, IFF2 das Interrupt-Zwischenspeicher-Flip-Flop

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
DEC IX	IX←IX-1	. . . . .	11 011 101 00 101 011	2	2	10	
DEC IY	IY←IY-1	. . . . .	11 111 101 00 101 011	2	2	10	
Befehlsgruppe: Verschiebung und zyklische Verschiebung							
RLCA		↕ . . . 0 0	00 000 111	1	1	4	zyklische Verschiebung AC, linksherum
RLA		↕ . . . 0 0	00 010 111	1	1	4	zyklische Verschiebung AC, nach links
RRCA		↕ . . . 0 0	00 001 111	1	1	4	zyklische Verschiebung AC, rechtsherum
RRA		↕ . . . 0 0	00 011 111	1	1	4	zyklische Verschiebung AC, nach rechts
RLC r		↕ ↕ P ↕ 0 0	11 001 011 00 000 r	2	2	8	zyklische Verschiebung Register, linksherum r | Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC M		↕ ↕ P ↕ 0 0	11 001 011 00 000 110	2	4	15	
RLC (IX+d)		↕ ↕ P ↕ 0 0	11 011 101 11 001 011 - d - 00 000 110	4	6	23	
RLC (IY+d)		↕ ↕ P ↕ 0 0	11 111 101 11 001 011 - d - 00 000 110	4	6	23	
RL s		↕ ↕ P ↕ 0 0	010				Befehlsformat und Zustände wie bei RLC s; im Operationscode 000 durch die umrandeten Bits ersetzen
RRC s		↕ ↕ P ↕ 0 0	001				
RR s		↕ ↕ P ↕ 0 0	011				
SLA s		↕ ↕ P ↕ 0 0	100				
SRA s		↕ ↕ P ↕ 0 0	101				
SRL s		↕ ↕ P ↕ 0 0	111				
RLD		. ↕ P ↕ 0 0	11 101 101 01 101 111	2	5	18	zyklische Zahlenverschiebung links und rechts zwischen AC und M, Inhalt der oberen Hälfte des AC wird nicht beeinflußt
RRD		. ↕ P ↕ 0 0	11 101 101 01 100 111	2	5	18	

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
Gruppe Bit setzen, löschen und testen							
BIT b,r	$Z \leftarrow \overline{r_b}$	. ↕ X X 0 1	11 001 011 01 b r	2	2	8	
BIT b,M	$Z \leftarrow \overline{M_b}$	. ↕ X X 0 1	11 001 011 01 b 110	2	3	12	
BIT b,(IX+d)	$Z \leftarrow \overline{(IX+d)_b}$	. ↕ X X 0 1	11 011 101 11 001 011 - d - 01 b 110	4	5	20	
BIT b,(IY+d)	$Z \leftarrow \overline{(IY+d)_b}$	. ↕ X X 0 1	11 111 101 11 001 011 - d - 01 b 110	4	5	20	
SET b,r	$r_b \leftarrow 1$	. . . . .	11 001 011 [11] b r	2	2	8	
SET b,M	$M_b \leftarrow 1$	. . . . .	11 001 011 [11] b 110	2	4	15	
SET b,(IX+d)	$(IX+d)_b \leftarrow 1$	. . . . .	11 011 101 11 001 011 - d - [11] b 110	4	6	23	
SET b,(IY+d)	$(IY+d)_b \leftarrow 1$	. . . . .	11 111 101 11 001 011 - d - [11] b 110	4	6	23	
RES b,s	$s_b \leftarrow 0$ s=r,M (IX+d),(IY+d)		[10]				Zur Bildung des neuen Operationscodes 11 in SET durch 10 ersetzen. Flags und Zeiten wie bei SET. 5)
Befehlsgruppe: Sprünge 6)							
JMP nn	PC←nn	. . . . .	11 000 011 - n - - n -	3	3	10	
JP cc,nn	wenn Bedingung cc wahr ist, PC←nn, sonst weiter	. . . . .	11 cc 010 - n - - n -	3	3	10	
JR e	PC←PC+e	. . . . .	00 011 000 - e-2 -	2	3	12	
JRC e	wenn C=0, kein Sprung wenn C=1, PC←PC+e	. . . . .	00 111 000 - e-2 -	2 2	2 3	7 12	Bedingung nicht erfüllt Bedingung erfüllt
JRNC e	wenn C=1, kein Sprung wenn C=0, PC←PC+e	. . . . .	00 110 000 - e-2 -	2 2	2 3	7 12	Bedingung nicht erfüllt Bedingung erfüllt

r	Register
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	getestetes Bit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

cc	Bedingung
000	NZ nicht Null
001	Z Null
010	NC kein Übertrag
011	C Übertrag
100	PO unpaarig
101	PE paarig
110	P Vorzeichen positiv
111	M Vorzeichen negativ

5) Die Schreibweise s_b bezeichnet das Bit b (0-7) des Speicherplatzes s.

6) e stellt die Abstandsangabe in der relativen Adressierungsart dar, bezogen auf das 1. Byte des Sprungbefehls. e ist ein Zweierkomplement mit Vorzeichen im Bereich von -126 bis +129.

e-2 liefert im Operationscode die tatsächliche Adresse PC+e, weil der Befehlszähler vor der Addition von e um 2 erhöht worden ist.

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	By-tes	M-Zyk-len	Tak-te	Bemerkung
JRZ e	wenn Z=0, kein Sprung wenn Z=1, PC←PC+e	. . . . .	00 101 000 - e-2 -	2 2	2 3	7 12	Bedingung nicht erfüllt Bedingung erfüllt
JRNZ e	wenn Z=1, kein Sprung wenn Z=0, PC←PC+e	. . . . .	00 100 000 - e-2 -	2 2	2 3	7 12	Bedingung nicht erfüllt Bedingung erfüllt
JMP M	PC←M	. . . . .	11 101 001	1	1	4	
JMP (IX)	PC←IX	. . . . .	11 011 101 11 101 001	2	2	8	
JMP (IY)	PC←IY	. . . . .	11 111 101 11 101 001	2	2	8	
DJNZ e	B←B-1 wenn B=0, kein Sprung wenn B≠0, PC←PC+e	. . . . .	00 010 000 - e-2 -	2 2	2 3	8 13	wenn B=0 wenn B≠0
Befehlsgruppe: Unterprogrammaufruf und Rücksprung							
CALL nn	(SP-1)←PCH (SP-2)←PCL PC←nn SP←SP-2	. . . . .	11 001 101 - n - - n -	3	5	17	
CA cc nn	wenn Bedingung cc falsch ist, kein Sprung, sonst wie CALL nn	. . . . .	11 cc 100 - n - - n -	3 3	3 5	10 17	wenn cc falsch ist wenn cc wahr ist
RET	PCL←(SP) PCH←(SP+1) SP←SP+2	. . . . .	11 001 001	1	3	10	
Rcc	Wenn Bedingung cc falsch ist, kein Sprung, sonst wie RET	. . . . .	11 cc 000	1 1	1 3	5 11	wenn cc falsch ist wenn cc wahr ist
RETI	Rücksprung vom Interrupt	. . . . .	11 101 101 01 001 101	2	4	14	
RETN	Rücksprung vom nicht maskierbaren Interrupt	. . . . .	11 101 101 01 000 101	2	4	14	
RST p	(SP-1)←PCH (SP-2)←PCL PCH←0 PCL←p SP←SP-2	. . . . .	11 t 111	1	3	11	

cc	Bedingung
000	NZ nicht Null
001	Z Null
010	NC kein Übertrag
011	C Übertrag
100	PO unpaarig
101	PE paarig
110	P Vorzeichen positiv
111	M Vorzeichen negativ

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Befehlsgruppe: Ein- und Ausgabe							
IN n	A←(n)	. . . . .	11 011 011 - n -	2	3	11	n zu A0 - A 7 AC zu A8 - A15
IN r INF	r←(C) wenn r=110, werden nur Flags gestellt	. ↕ P ↕ 0 0	11 101 101 01 r 000	2	3	12	C zu A0 - A 7 B zu A8 - A15
INI	M ←(C) B←B-1 HL←HL+1	. ↕ X X 1 X a	11 101 101 10 100 010	2	4	16	C zu A0 - A 7 B zu A8 - A15 7)
INIR	M ←(C) B←B-1 HL←HL+1 Wiederholung bis B=0	. 1 X X 1 X	11 101 101 10 110 010	2 2	5 B≠0 4 B=0	21 16	C zu A0 - A 7 B zu A8 - A15
IND	M ←(C) B←B-1 HL←HL-1	. ↕ X X 1 X a	11 101 101 10 101 010	2	4	16	C zu A0 - A 7 B zu A8 - A15 7)

7) a: Wenn das Ergebnis von B-1 = 0, dann wird das Z-Flag gesetzt, sonst ist es gelöscht.

Assembler Sprache	symbolische Operation	Flags C Z P/V S N H	Operations-code 76 543 210	Bytes	M-Zyklen	Takte	Bemerkung
INDR	M ←(C) B←B-1 HL←HL-1 Wiederholung bis B=0	. 1 X X 1 X	11 101 101 10 111 010	2 2	5 B≠0 4 B=0	21 16	C zu A0 - A 7 B zu A8 - A15
OUT n	(n)←A	. . . . .	11 010 011 - n -	2	3	11	n zu A0 - A 7 AC zu A8 - A15
OUT r	(C)←r	. . . . .	11 101 101 01 r 001	2	3	12	C zu A0 - A 7 B zu A8 - A15
OUTI	(C)← M B←B-1 HL←HL+1	. ↕ a X X 1 X	11 101 101 10 100 011	2	4	16	C zu A0 - A 7 B zu A8 - A15 7)
OTIR	(C)← M B←B-1 HL←HL+1 Wiederholung bis B=0	. 1 X X 1 X	11 101 101 10 110 011	2 2	5 B≠0 4 B=0	21 16	C zu A0 - A 7 B zu A8 - A15
OUTD	(C)← M B←B-1 HL←HL-1	. ↕ a X X 1 X	11 101 101 10 101 011	2	4	16	C zu A0 - A 7 B zu A8 - A15 7)
OTDR	(C)← M B←B-1 HL←HL-1 Wiederholung bis B=0	. 1 X X 1 X	11 101 101 10 111 011	2 2	5 B≠0 4 B=0	21 16	C zu A0 - A 7 B zu A8 - A15

7) siehe Seite 42

8. Interrupt-Verhalten

Der Zweck eines Interrupts besteht darin, es den peripheren Geräten zu ermöglichen, die CPU-Operation in einer sinnvollen Weise zu unterbinden und die CPU zu zwingen, daß eine Routine zur Bedienung der Peripherie gestartet wird. Gewöhnlich sind in diesen Routinen Austauschoperationen für Daten, Status- oder Steuerinformationen zwischen der CPU und der Peripherie eingeschlossen. Wenn die Bedienungsroutine abgearbeitet ist, kehrt die CPU zu der Operation zurück, bei der sie unterbrochen wurde.

8.1. Interrupt-Annahme/-Abweisung

Die CPU U880 hat zwei Interrupteingänge, einen durch die Software maskierbaren ($\overline{INT}$) und einen nicht maskierbaren Interrupt ($\overline{NMI}$). Die Annahme des nicht maskierbaren Interrupts kann durch den Programmierer nicht verhindert werden. Er wird immer angenommen, wenn ein peripheres Gerät ihn fordert. Dieser Interrupt wird i. a. für die wichtigsten Funktionen reserviert, die beim Auftreten sofort bedient werden müssen, z. B. ein bevorstehender Stromausfall.

Neben den zwei Interrupteingängen besitzt die CPU noch den BUSRQ-Steuereingang für die Busübergabe, der speziell bei DMA-Betrieb Verwendung findet. Dieser Steuereingang hat gegenüber allen Interruptanforderungen höchste Priorität. Insgesamt ergibt sich in der CPU folgende Prioritätsbewertung:

1. Priorität: Bus-Request ($\overline{BUSRQ}$)
2. " : nicht maskierbarer Interrupt ($\overline{NMI}$)
3. " : maskierbarer Interrupt ($\overline{INT}$)

Der maskierbare Interrupt kann durch den Programmierer selektiv zugelassen oder abgewiesen werden. Damit hat der Programmierer die Möglichkeit, Interrupts abzuweisen, wenn während bestimmter Perioden das Programm ein Verhalten realisieren muß, in dem ein Interrupt nicht zulässig ist. In der CPU U880 gibt es ein Annahme-Flip-Flop (IFF1), das durch den Programmierer mit den Befehlen Interrupt-Annehmen (Enable interrupt - EI) bzw. Interrupt-Abweisen (Disable interrupt - DI) ein- bzw. ausgeschaltet werden kann. Wenn IFF1 ausgeschaltet ist, kann durch die CPU kein Interrupt angenommen werden.
Tatsächlich gibt es in der CPU U880 2 Annahme-Flip-Flops: IFF1 und IFF2.

IFF1	IFF2
Verhindert aktuell die Annahme von Interrupts	zeitweiliger Speicherplatz für IFF1

Die Stellung von IFF1 wird benutzt, um aktuelle Interrupts abzuweisen, während IFF2 nur als zeitweiliger Speicherplatz für IFF1 dient. Der Zweck, IFF1 zu speichern, ist folgender:

Ein RESET der CPU schaltet u. a. IFF1 und IFF2 aus, so daß Interrupts abgewiesen werden. Sie können zu beliebiger Zeit durch den Programmierer mit dem EI-Befehl zugelassen werden. Wenn ein EI-Befehl ausgeführt ist, wird eine schon anliegende Interruptanforderung erst nach der Abarbeitung des auf EI folgenden Befehls angenommen. Diese Verzögerung um einen Befehl ist für den Fall erforderlich, wenn der auf EI folgende ein Rücksprung (return) ist, da ein Interrupt nicht zugelassen werden kann, bis der Rücksprung abgearbeitet ist. Der EI-Befehl setzt sowohl IFF1 als auch IFF2 in den Annahme-Zustand. Wenn ein Interrupt durch die CPU angenommen wird, werden sowohl IFF1 als auch IFF2 automatisch ausgeschaltet, um weitere Interrupts zu verhindern, bis der Programmierer einen neuen EI-Befehl benutzt. D. h. in den oben genannten Fällen sind IFF1 und IFF2 immer gleich.
Der Zweck, in IFF2 den Zustand von IFF1 zu speichern, wird deutlich, wenn ein nichtmaskierbarer Interrupt auftritt. Wenn ein nicht maskierbarer Interrupt angenommen wird, wird IFF1 ausgeschaltet, um weitere Interrupts zu verhindern, bis der Programmierer sie wieder zulassen will. Folglich werden, nachdem ein nicht maskierbarer Interrupt angenommen wurde, maskierbare Interrupts abgewiesen, aber der vorherige Zustand von IFF1 ist gerettet worden, so daß der komplette Zustand der CPU, wie er vor dem nicht maskierbaren Interrupt bestanden hatte, wieder hergestellt werden kann. Wenn der Befehl "Laden des Akkumulators vom Register I" (LD A, I) oder der Befehl "Laden des Akkumulators vom Register R" (LD A, R) ausgeführt ist, ist der Zustand von IFF2 in das Paritäts-Flag überführt worden, wo er getestet oder gespeichert werden kann.

Eine zweite Methode, den Zustand von IFF 1 zurückzugewinnen, ist die Abarbeitung des Befehls "Rücksprung vom nicht maskierbaren Interrupt" (RETN). Da dieser Befehl anzeigt, daß die Behandlungsroutine eines nicht maskierbaren Interrupts abgearbeitet ist, wird der Inhalt von IFF 2 nach IFF 1 überführt, so daß der Zustand von IFF 1 automatisch so wiederhergestellt ist, wie er vor der Annahme des nicht maskierbaren Interrupts bestanden hatte.

In Tabelle 18 ist die Wirkung verschiedener Befehle auf die zwei Interrupt-Annahme-Flip-Flops zusammengestellt:

Aktion	IFF 1	IFF 2			
CPU RESET	0	0			
DI	0	0			
EI	1	1			
LD A, I	.	.	IFF 2	⟶	Paritäts-Flag
LD A, R	.	.	IFF 2	⟶	Paritäts-Flag
Annahme von NMI	0	.			
RETN	IFF 2	.	IFF 2	⟶	IFF 1

"." bedeutet: keine Veränderung

Tabelle 18: Interrupt-Annahme/-Abweisungs-Flip-Flops

Bild 17 zeigt den logischen Ablauf der Bewertung der drei Steuereingänge $\overline{\text{BUSRQ}}$, $\overline{\text{NMI}}$ und $\overline{\text{INT}}$. Die $\overline{\text{BUSRQ}}$-Leitung wird zuerst getestet. Das interne BUSRQ-Flip-Flop (BUSRQ-F/F) wird gesetzt, wenn die $\overline{\text{BUSRQ}}$-Leitung bei der Anstiegsflanke des letzten Taktes jedes Maschinenzyklus aktiv ist. Ist dieser Maschinenzyklus gleichzeitig der letzte Maschinenzyklus der in Ausführung befindlichen Anweisung, so ist die CPU auch zur Übernahme der nichtmaskierbaren oder maskierbaren Interruptsignale bereit. In diesem Fall werden nach der $\overline{\text{BUSRQ}}$-Leitung die $\overline{\text{NMI}}$-Leitung bzw. die $\overline{\text{INT}}$-Leitung getestet. Da die $\overline{\text{NMI}}$-Leitung flankengetriggert ist und ihre negativen Flanken ein NMI-Eingangs-Flip-Flop (NMI-EF/F) setzen, wird beim Test der $\overline{\text{NMI}}$-Leitung eigentlich der $\overline{\text{NMI}}$-Eingangs-Flip-Flop abgefragt und ausgewertet. Bei einem gesetzten NMI-Eingangs-Flip-Flop werden dann das interne NMI-Flip-Flop gesetzt und das INT-Annahme-Flip-Flop IFF 2 rückgesetzt. Wenn das NMI-Eingangs-Flip-Flop seinerseits nicht gesetzt ist, prüft die CPU nun den Zustand der $\overline{\text{INT}}$-Leitung und setzt bei einer aktiven $\overline{\text{INT}}$-Leitung und einer nicht aktiven Interruptverhinderung (IFF 1 $\neq \emptyset$), das interne INT-Flip-Flop.
Die Abarbeitung der angemeldeten Interrupts erfolgt dann durch Auswertung der internen Flip-Flops in der Reihenfolge BUSRQ-F/F, NMI-F/F und INT-F/F.

In Tabelle 19 sind nochmals alle Reaktionen der CPU auf die drei Steuerleitungen $\overline{BUSRQ}$, $\overline{NMI}$ und $\overline{INT}$ unter Berücksichtigung aller Randbedingungen aufgelistet.
Die steigende Flanke des letzten Taktes (T_L) innerhalb eines Maschinenzyklus ist immer für die Übernahme der Steuerinformation in die internen Flip-Flops verantwortlich. Im Falle des Maschinenzyklus $\overline{M1}$ ist $T_L = T_4$, im Falle von Speicher-Schreib-/Lesezyklen ist $T_L = T_3$. Bei der Priorisierung von $\overline{NMI}$ gegenüber $\overline{INT}$ ist zu beachten, daß NMI nur dann vorgezogen wird, wenn die Anforderung auch wirklich vor der steigenden Flanke des letzten Taktes T_L des entsprechenden Befehls erscheint. In diesem Fall wird $\overline{INT}$ nicht in das interne INT-F/F übernommen, also bis zum Ende der NMI-Service-Routine ignoriert. Ansonsten erfolgt die Ausführung des ersten Befehls im Service-Programm für $\overline{INT}$ und erst dann der Sprung zur $\overline{NMI}$-Service-Routine.

Signal	Auftreten des Signals	Übernahme des Signals	Wirkung	Bemerkung
$\overline{BUSRQ}$		nur bei steigender Flanke eines jeden T_L	Bestätigung (BUSAK) erscheint sofort vor dem nächsten Maschinenzyklus	$\overline{BUSRQ}$ hat Priorität vor $\overline{NMI}$ und $\overline{INT}$
$\overline{NMI}$	innerhalb eines Befehlszyklus vor der steigenden Flanke des nächsten T_L	wird jederzeit im NMI-Eingangs-Flip-Flop abgespeichert (Mindestimpulsbreite 80 ns)	sofort nach Abarbeitung des Befehls, Sprung zur NMI-Service-Routine (auch unmittelbar nach einem EI-Befehl)	zur wiederholten Ausführung einer NMI-Service-Routine muß das $\overline{NMI}$-Signal kurzzeitig inaktiv werden
	innerhalb eines Befehlszyklus nach der steigenden Flanke des nächsten T_L	- " -	Abarbeitung des laufenden und des nächstfolgenden Befehls, erst dann Sprung zur NMI-Service-Routine	- " -
$\overline{INT}$		nur bei steigender Flanke des letzten T_L eines jeden Befehls und nur dann, wenn Interrupt freigegeben ist. Andernfalls wird $\overline{INT}$ nicht akzeptiert und ignoriert.		Ausnahme: während der Ausführung eines EI-Befehls wird $\overline{INT}$ nicht akzeptiert, unabhängig vom vorherigen Zustand von IFF 1 und IFF 2

Tabelle 19

8.2. Interrupt-Beantwortung

CPU

- nicht maskierbar_
Ein nicht maskierbarer Interrupt wird durch die CPU zu jeder Zeit angenommen. Wenn dieser auftritt, ignoriert die CPU den nächsten aufzurufenden Befehl und führt dafür einen RESTART zur Adresse 0066 H durch. Folglich verhält sie sich genauso, als hätte sie einen RESTART-Befehl aufgerufen, mit dem Unterschied, daß diese Adresse keine der 8 Software-RESTART-Adressen ist. Ein RESTART ist ein Unterprogrammaufruf von einer bestimmten Adresse im Anfangsbereich des Speichers.

- maskierbar_
Die CPU kann so programmiert werden, daß sie auf einen maskierbaren Interrupt in einer der drei möglichen Arten (Mode) antwortet.

Mode 0
Dieser Mode ist identisch dem Interruptverhalten der CPU 8080 A. Bei diesem Mode kann die den Interrupt anfordernde Schaltung einen Befehl auf den Datenbus ausgeben und die CPU wird ihn ausführen. Folglich liefert die den Interrupt anfordernde Schaltung den nächsten abzuarbeitenden Befehl anstelle des Speichers. Meist wird das ein RESTART-Befehl sein, weil die den Interrupt anfordernde Schaltung nur einen Ein-Byte-Befehl einspeisen kann. Mit anderen Worten, es kann kein beliebiger anderer Befehl, wie z. B. ein 3-Byte-Unterprogrammruf, ausgeführt werden. Die Taktzahl für die Aus-

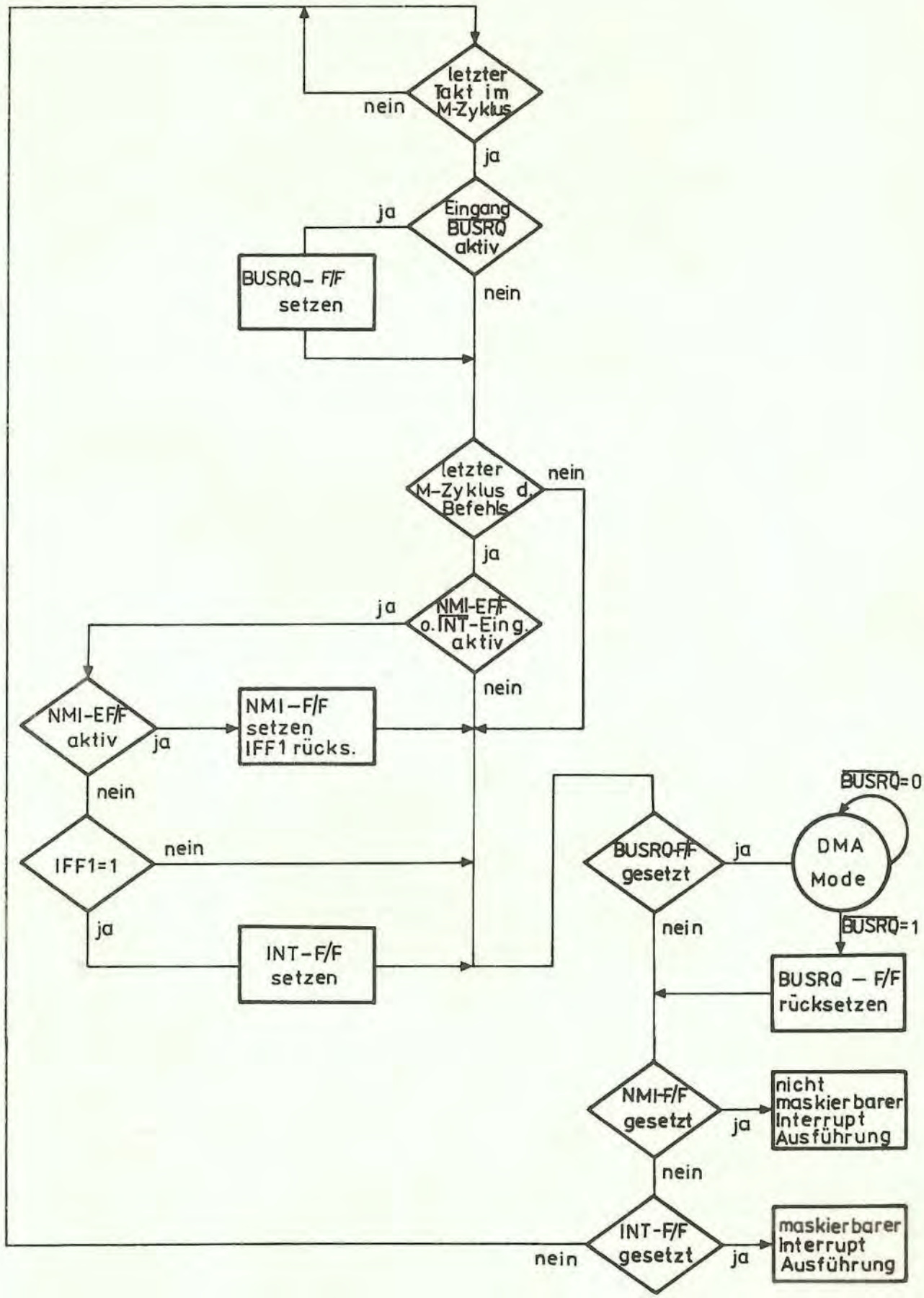
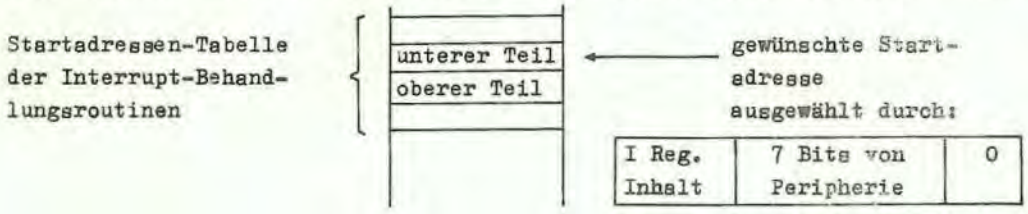


Bild 17: Logischer Ablauf der Bewertung von $\overline{\text{BUSRQ}}$, $\overline{\text{NMI}}$ und $\overline{\text{INT}}$

führung dieses Befehls ist um 2 Takte größer als die normale Zahl für diesen Befehl. Das kommt daher, weil die CPU automatisch 2 WAIT-Zustände in den Interrupt-Antwortzyklus einfügt, um der externen Logikkette (daisy chain) genügend Zeit für die Prioritätssteuerung zur Verfügung zu stellen. In Abschnitt 5. ist das Zeitverhalten für eine Interrupt-Antwort detailliert dargestellt. Nach einem RESET der CPU ist immer automatisch Interrupt Mode 0 gesetzt.

Mode 1
Wenn dieser Mode durch den Programmierer ausgewählt ist, wird die CPU einen Interrupt mit einem RESTART zur Adresse 0038 H beantworten. Folglich ist diese Antwort mit der auf einen nicht maskierbaren Interrupt identisch mit der Ausnahme, daß jetzt die Adresse 0038 H anstelle von 0066 H aufgerufen wird. Ein weiterer Unterschied besteht darin, daß die erforderliche Zyklenzahl um 2 gegenüber dem normalen RESTART entsprechend den zwei eingefügten WAIT-Zuständen vergrößert ist.

Mode 2
Dieser Mode ist die leistungsfähigste der Interrupt-Antwort-Moden. Mit einem einzigen 8-Bit-Byte kann vom Benutzer ein indirekter Unterprogrammaufruf zu einem beliebigen Speicherplatz durchgeführt werden. Bei diesem Mode stellt der Programmierer eine Tabelle mit 16-Bit-Startadressen für jede Interrupt-Behandlungsroutine auf. Diese Tabelle kann irgendwo im Speicher untergebracht sein. Wenn ein Interrupt angenommen wird, muß ein 16-Bit-Zeiger gebildet werden, um die Startadresse der gewünschten Interrupt-Behandlungsroutine aus der Tabelle holen zu können. Die oberen 8 Bit des Zeigers werden aus dem Inhalt des Registers I gebildet. Das Register I muß zuvor mit dem vom Programmierer gewünschten Wert geladen werden, d. h. LD I, A. Zu beachten ist, daß ein RESET der CPU auch das Register I zurückstellt, das heißt, daß dort eine Null eingeschrieben wird. Die unteren 8 Bit müssen von der den Interrupt anfordernden Schaltung geliefert werden, wobei das niederwertigste Bit eine 0 sein muß. Das ist deshalb notwendig, weil der 16-Bit-Zeiger dazu verwendet wird, zwei aufeinanderfolgende Bytes aus einer Interrupt-Tabelle zu holen, um die vollständige 16-Bit-Startadresse der Behandlungsroutine zu bilden. Die Startadressen der Interrupt-Routinen müssen in der Tabelle immer an geraden Speicherplätzen stehen.



Das erste Byte jeder Adresse in dieser Tabelle ist der niederwertige Teil der Adresse. Der Programmierer muß selbstverständlich diese Tabelle mit den gewünschten Adressen füllen, bevor ein Interrupt angenommen werden darf.
Zu beachten ist, daß diese Tabelle zu jeder Zeit durch den Programmierer verändert werden kann, wenn verschiedene periphere Schaltungen mit verschiedenen Behandlungsroutinen bedient werden sollen. Voraussetzung dafür ist, daß die Tabelle in einem Lese-Schreib-Speicher untergebracht ist.

Wenn eine den Interrupt anfordernde Schaltung den unteren Teil des Zeigers liefert, kellert die CPU den Befehlszählerstand automatisch. Sie holt dann die Startadresse aus der Tabelle und führt einen Sprung zu dieser Adresse aus. Dieser Antwort-Mode benötigt 19 Takt-Perioden (7, um die unteren 8 Bit von der den Interrupt anfordernden Schaltung aufzurufen, 6, um den Befehlszähler zu retten, und 6, um die Sprungadresse zu bilden).
Zu beachten ist, daß die peripheren Schaltungen des U880-Systems für die Interrupt-Prioritäten eine Logikkettenstruktur (daisy chain structure) aufweisen, die der CPU während der Interrupt-Annahme automatisch einen programmierten Vektor liefert. Nähere Informationen dazu können aus den Technischen Beschreibungen für PIO, SIO und CTC entnommen werden.

9. Hinweise zum Hardwareaufbau

Dieses Kapitel gibt einige grundsätzliche Hinweise für den Aufbau von Systemen mit der CPU U880.

Minimalsystem

Bild 18 zeigt ein Blockschaltbild eines möglichen U880-Minimalsystems. Jedes U880-System muß folgende Elemente enthalten:

1) 5-V-Stromversorgung
2) Taktgenerator
3) Speicherbauelemente
4) E/A-Schaltkreise
5) CPU

Da die CPU U880 nur eine einzige 5-V-Stromversorgung benötigt, können kleine Systeme mit einer einzigen Spannungsquelle ausgerüstet werden, wenn die verwendeten Speicher nicht zusätzliche Spannungen benötigen. Der Taktgenerator ist ein einfacher Rechteckwellengenerator. Für Systeme, die nicht mit der maximal möglichen Geschwindigkeit laufen, reicht ein RC-Generator aus. Wenn die CPU in der Nähe der maximal zulässigen Frequenz betrieben wird, ist i. a. ein Quarzgenerator erforderlich, weil in diesem Fall das Zeitverhalten der CPU Drift oder andere Zeitunregelmäßigkeiten, die bei RC-Netzwerken auftreten, nicht zulassen kann.

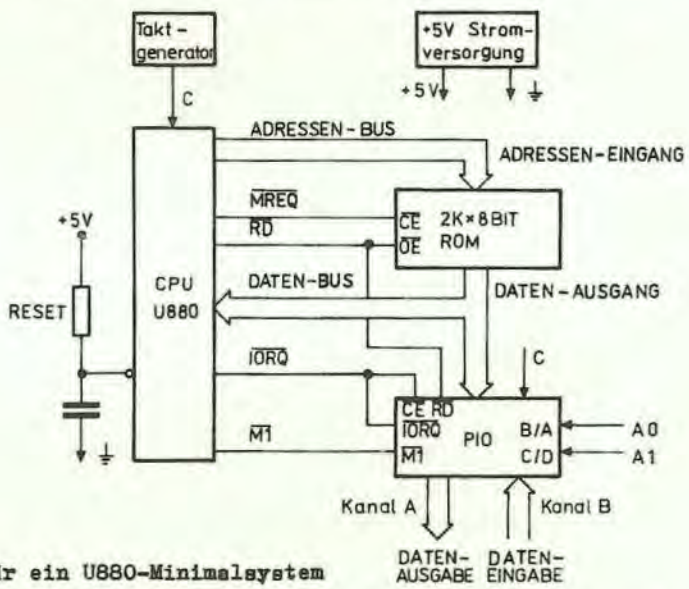


Bild 18: Prinzipaufbau für ein U880-Minimalsystem

Der externe Speicher kann eine Kombination aus Standard-RAM, ROM oder PROM sein. In dem Prinzipschaltbild für ein Minimalsystem ist ein einzelnes 16-K-Bit-ROM (2 K Byte) als Speichersystem angenommen worden. Für dieses Beispiel ist weiterhin angenommen worden, daß die U880-Register eine ausreichende Lese/Schreibspeicherkapazität bilden, so daß externe RAMs nicht erforderlich sind. Jedes Rechnersystem benötigt E/A-Schaltungen, um mit der Umwelt in Verbindung treten zu können. In diesem einfachen System wird beispielsweise als Ausgang ein 8-Bit-Steuersignal und als Eingang ein 8-Bit-Statuswort gefordert. Die Eingangsdaten können auf den Datenbus unter Verwendung eines Standard-Drei-Zustandstreibers eingespeist werden. Die Ausgabedaten lassen sich mit irgendeiner Standard-TTL-Halteschaltung abnehmen. Im angenommenen Beispiel ist die PIO U855 als E/A-Schaltkreis eingesetzt worden. Dieser eine Schaltkreis ist an den Datenbus angeschlossen und liefert die erforderlichen 16 Bits der TTL-kompatiblen E/A-Leitungen (s. auch Technische Beschreibung zur PIO).
In dem aufgeführten Beispiel werden nur 3 LSI-Schaltkreise, ein einfacher Taktgenerator und eine 5-V-Stromversorgung benötigt, um einen leistungsfähiges Minimalsystem aufzubauen.

Einfügung von RAMs

Die meisten Rechnersysteme erfordern eine gewisse Menge Speicherplatz in einem externen Lese/Schreib-Speicher zur Datenspeicherung und zur Bildung eines Kellerspeichers. In Bild 19 ist dar-

gestellt, wie ein 256 Byte-statischer Speicherblock an das System des vorigen Beispiels angeschlossen wird. In diesem Beispiel ist der Speicherplatz wie folgt aufgeteilt worden:

Adresse	
0000 H 07 FFH	2 K Bytes ROM
0800 H 08 FFH	256 Bytes RAM

Das Adressenbit A10 trennt hier den ROM-Bereich vom RAM-Bereich, so daß es für die Chip-Auswahlfunktion benutzt werden kann. Für größere Speichersysteme sind zusätzliche TTL-Dekoder für die Chip-Auswahl erforderlich.

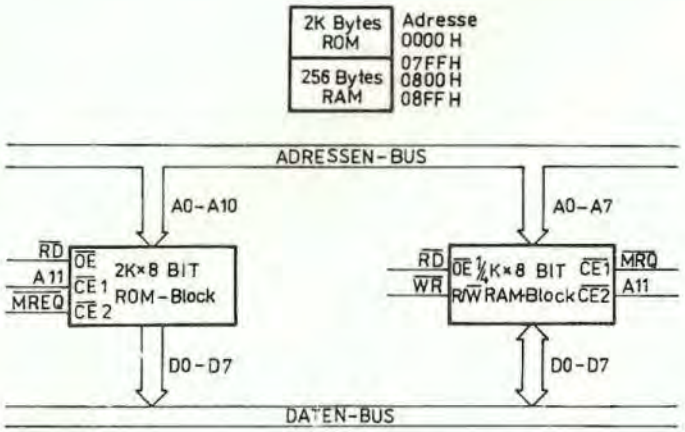


Bild 19: Beispiel einer Systemerweiterung mit ROM und RAM
(Die Blöcke beinhalten die Speicher und die Chip-Auswahl-Logik)

Steuerung der Speichergeschwindigkeit

Die $\overline{WAIT}$-Leitung zur CPU liefert die Voraussetzung, daß die CPU U880 mit Speichern beliebiger Zugriffszeit arbeiten kann. Im Abschnitt 4. wurde erläutert, daß die schärfsten Forderungen an die Speicherzugriffszeit im M1-Zyklus, dem Befehlsaufruf, gestellt werden. Alle anderen Speicherzugriffe sind einen halben Taktzyklus länger. Aus diesem Grunde kann es in manchen Fällen sinnvoll sein, zusätzliche WAIT-Zustände in den M1-Zyklus einzubauen, so daß langsamere Speicher verwendet werden können.

In Bild 20 ist ein Beispiel einer einfachen Schaltung angegeben, die einen WAIT-Zustand in jedem M1-Zyklus einfügt. Diese Schaltung kann so abgewandelt werden, daß ein WAIT-Zustand bei jedem Speicherzugriff eingefügt wird (siehe Bild 21).

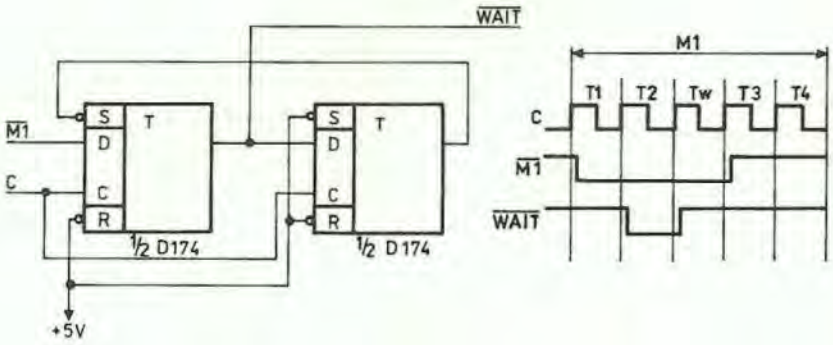


Bild 20: Schaltung und Impulsbild für das Einfügen eines WAIT-Zustandes
in jedem M1-Zyklus

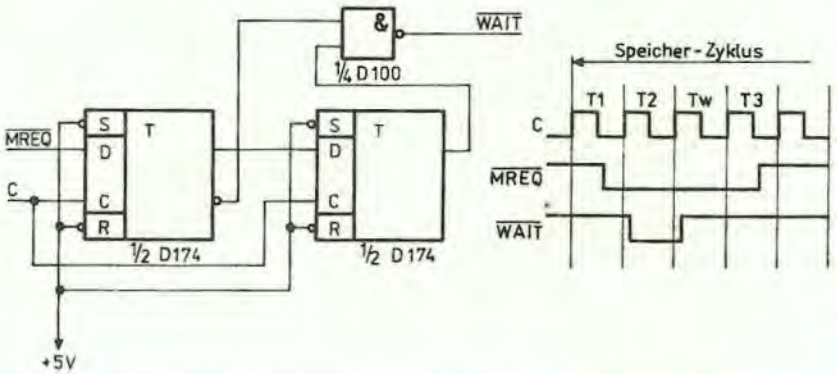


Bild 21: Schaltung und Impulsbild für das Einfügen eines WAIT-Zustandes in jeden Speicher-Zyklus

Anschluß von dynamischen Speichern

Dieser Abschnitt ist nur als eine kurze allgemeine Information darüber aufzufassen, wie dynamische Speicher anzuschließen sind. Jedes dynamische RAM hat andere Spezifikationen, so daß geringe Modifikationen zu dieser Beschreibung notwendig werden.
Bild 22 illustriert, welche Logik notwendig ist, um 32 K Byte eines 16poligen 16 K dynamischen RAMs anzuschließen. Das Beispiel setzt voraus, daß diese RAMs die einzigen im System sind, so daß das Adressenbit A 14 dazu benutzt werden kann, zwischen den zwei Speicherteilen auszuwählen. Während der Auffrischzeit müssen alle Speicher im System gelesen werden. Die CPU liefert geeignete Auffrischungsadressen auf den Leitungen A 0 bis A 6. Sollen zusätzliche Speicher an das System angeschlossen werden, so ist nur erforderlich, die Block-Selekt-Logik durch einen Dekoder zu ersetzen, der mit allen benötigten Adressenbits arbeitet. Bei großen Systemen ist i. a. auch die Pufferung für den Adressen- und Datenbus erforderlich.

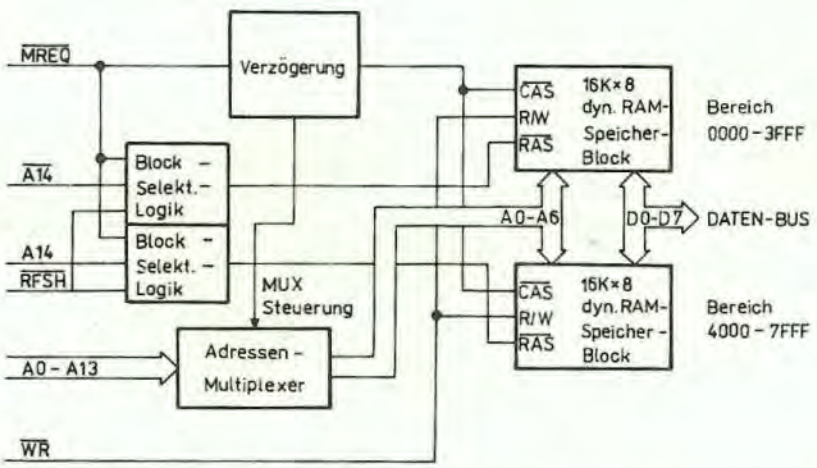


Bild 22: Prinzipaufbau einer dyn. Speicheranordnung mit 2 x 16 K x 8 bit Speicherkapazität

10. Software-Implementierungs-Beispiele

10.1. Software-Merkmale der CPU U880

Der Befehlssatz der CPU U880 bietet dem Anwender ein breites und flexibles Repertoire an Operationen, mit denen die Steuerung der CPU U880 formuliert werden kann.

Die Haupt-, Alternativ- und Indexregister können dazu verwendet werden, die Argumente von arithmetischen und logischen Operationnen zu speichern, Speicheradressen zu bilden oder als Speicher mit schnellem Zugriff für kurzfristig benötigte Daten zu dienen.

Informationen können überführt werden: direkt von Register zu Register, vom Speicher zum Speicher, vom Speicher zu Registern und von Registern zum Speicher. Zusätzlich können die Registerinhalte und Register/Speicherinhalte ohne zeitweilige Zwischenspeicherung ausgetauscht werden. Speziell können die Inhalte der Haupt- und Alternativregister mit nur 2 Befehlen (EXAF, EXX) komplett aus-

getauscht werden. Dieser Registeraustausch kann sowohl dazu benutzt werden, die Sätze von Arbeitsregistern für verschiedene logische Programmteile auszuwählen, als auch innerhalb eines Programmteils die Anzahl der zur Verfügung stehenden Register zu erweitern.

Die Speicherung und das Wiederfinden der Daten in Registerpaaren und im Speicher kann auf der LIFO- (letzte hinein, erste heraus) Basis durch die Befehle PUSH und POP gesteuert werden, die ein spezielles Keller-Zeigerregister (SP) verwenden. Dieses Keller-Register steht sowohl für die Manipulation von Daten zur Verfügung als auch zur automatischen Abspeicherung und zum Wiederaufruf von Unterprogramm-Rücksprungadressen. Wenn z. B. ein Unterprogramm aufgerufen wird, wird die Adresse, die dem CALL-Befehl folgt, in die oberste Ebene des Kellers eingetragen, die durch SP angezeigt wird. Wenn das Unterprogramm zum Hauptprogramm zurückspringt, wird die Adresse in der obersten Ebene des Kellers dazu benutzt, den Befehlszähler für die Adresse des nächsten Befehls zu stellen. Der Kellerzeiger wird automatisch eingestellt, so daß der Kellerzeiger während der PUSH, POP, CALL und RET-Befehle immer die aktuelle oberste Ebene des Kellers anzeigt. Dieser Kellermechanismus gestattet es, das Kellern von Daten und Unterprogrammaufrufen praktisch bis in jede beliebige Tiefe zu schachteln, weil der Kellerbereich im Prinzip so groß wie der ganze Speicher sein kann.

Die Reihenfolge der Befehlsabarbeitung kann durch 6 verschiedene Flags (Übertrag, Null, Vorzeichen, Parität/Überlauf, Add. - Subtraktion, Halbbyte-Übertrag) gesteuert werden. Diese Flags spiegeln die Ergebnisse von arithmetischen, logischen, Verschiebe- und Vergleichsoperationen wieder. Nach einer Befehlsabarbeitung, die ein oder mehrere Flags beeinflußt hat, kann ein Flag benutzt werden, um einen bedingten Sprung oder einen Rückkehrbefehl zu steuern und dadurch die logische Programmverzweigung ermöglichen.

Ein voller Satz logischer Befehle der AND, OR, XOR (exklusives Oder), CPL (Einerkomplement) und NEG (Zweierkomplement) umfaßt, steht für Boolsche Operationen zwischen Akkumulator und 1. allen 8-Bit-Registern, 2. den Speicherplätzen oder 3. Direktwerten zur Verfügung.
Zusätzlich steht ein voller Satz von arithmetischen und logischen Verschiebebefehlen in beiden Richtungen zur Verfügung, die auf die Inhalte aller primären 8-Bit-Register oder direkt auf einen Speicherplatz wirken. Das Übertragsflag kann einbezogen oder einfach durch diese Verschiebebefehle gesetzt werden, um sowohl das Testen der Ergebnisse der Verschiebung als auch die Verbindung Register/Register oder Register/Speicher für Verschiebeoperationen zu ermöglichen. Die Reihe der 16-Bit-Arithmetikbefehle gestattet die unkomplizierte Berechnung von Speicheradressen insbesondere bei Tabellen und Datenblockverarbeitung.
Die Palette der Einzelbitbefehle ermöglicht die direkte und gezielte Beeinflussung (Setzen, Rücksetzen und Testen) von beliebigen Bitstellen in allen 8-Bit-Registern des Hauptregistersatzes und in allen Speicherplätzen. Die Bit-Testbefehle übernehmen das zu testende Bit in Komplementform in das Nullflag, so daß bitabhängige Programmverzweigungen sehr einfach realisierbar sind.

10.2. Beispiele für die Anwendung spezieller U880 Befehle

10.2.1. Blocktransportoperation

Es soll eine Datenkette, die im Speicher bei Adresse "DATA" beginnt, in einen anderen Speicherbereich überführt werden, der bei Adresse "BUFFER" beginnt. Die Kette soll 500 Bytes lang sein. Diese Aufgabe kann man wie folgt lösen:

```
LD    HL, DATA          ;Startadresse der Datenkette
LD    DE, BUFFER        ;Startadresse des Zielbereiches
LD    BC, 500           ;Länge der Datenkette
LDIR                    ;Transportiere die Kette vom Speicher-
                        ;platz, durch DE definiert,
                        ;Erhöhe HL und DE, Vermindere BC
                        ;wiederhole bis BC = 0.
```

Insgesamt sind 11 Bytes-Programm-Speicherplatz für die Operation erforderlich. Jedes Datenbyte wird innerhalb von 21 Takten transportiert.

10.2.2. Blocksuchoperation

Es soll eine Datenkette, die bei Adresse "DATA" beginnt, in einen anderen Speicherbereich transportiert werden, der bei "BUFFER" beginnt, bis das ASCII-Zeichen : gefunden wird (d. h. : ist Endekennung der Kette). Maximal kann die Kette eine Länge von 132 Zeichen haben. Die Operation kann wie folgt ausgeführt werden:

```
      LD    HL, DATA            ;Startadresse der Datenkette
      LD    DE, BUFFER          ;Startadresse des Zielbereiches
      LD    BC, 132             ;maximale Kettenlänge
      LD    A, ':'              ;Endekennung in Akkumulator
   LOOP: CP (HL)                ;Vergleich Speicherinhalt mit Ende-
                                ;kennung
      JRZ  END                  ;Sprung zu END, wenn Zeichen gleich
      LDI                       ;Zeichentransport (HL) zu (DE)
                                ;Erhöhen HL und DE, Vermindern BC
      JPPE  LOOP                ;Sprung zu LOOP, wenn noch Zeichen
                                ;zu transportieren sind, sonst gehe
   END:                         ;weiter. Beachte: P/V-Flag wird ver-
                                ;wendet, um zu signalisieren, daß
                                ;BC auf Null erniedrigt ist
```

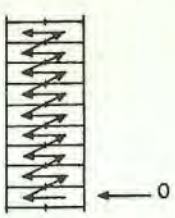
Insgesamt werden 19 Bytes Programmspeicherplatz für diese Operation benötigt.

10.2.3. BCD-Schieben im Speicher

Eine Dezimalzahl mit 16 Ziffern im gepackten BCD Format (zwei Ziffern je Byte) soll wie unten gezeigt verschoben werden, um die BCD Multiplikation oder Division zu vereinfachen.

```
      LD    HL, DATA            ;Adresse d. ersten Bytes
      LD    B, COUNT            ;Verschiebungszähler
      XOR   A                   ;AC löschen
ROTAT: RLD                      ;zykl. Versch., linksherum, der
                                ;unteren Zahl des AC mit Zahlen
                                ;in (HL)
      INC   HL                  ;Weiterschalten des Speicher-
                                ;zeigers
      DJNZ  ROTAT               ;Erniedrigen B, Sprung zu ROTAT
                                ;wenn B ≠ 0, sonst gehe weiter
```

Für diese Operation sind 11 Programmbytes erforderlich.



10.2.4. BCD-Subtraktion im Speicher

Es soll eine Zahl von einer anderen subtrahiert werden. Beide haben gepacktes BCD Format. Sie haben beide die gleiche aber variable Länge. Das Ergebnis soll auf dem Speicherplatz des Minuenden gespeichert werden:

```
      LD    HL, ARG 1           ;Adresse des Minuenden
      LD    DE, ARG 2           ;Adresse des Subtrahenden
      LD    B, LENGTH           ;Länge der beiden Argumente
      AND   A                   ;Nullsetzen des Übertragsflags
SUBDEC: LD  A, (DE)             ;Subtrahend nach AC
      SBC   (HL)                ;Subtraktion (HL) von AC
      DAA                       ;Korrektur des Ergebnisses auf
                                ;den dezimalkodierten Wert
```

```
        LD    (HL), A              ;Ergebnis speichern
        INC   HL                   ;Weiterschalten der Speicherzeiger
        INC   DE
        DJNZ  SUBDEC               ;Erniedrigen B und Sprung zu SUBDEC,
                                   ;wenn B ≠ 0, sonst gehe weiter
```

Für diese Operation werden 17 Bytes Programmspeicherplatz benötigt.

10.3. Beispiele von programmierten Aufgaben

10.3.1. Sortierprogramm

Das folgende Programm sortiert ein Feld von Zahlen, die im Bereich von 0 bis 255 liegen, in aufsteigender Reihenfolge unter Verwendung eines Standard-Austausch-Sortier-Algorithmus:

```
                    1   ;     STANDARD AUSTAUSCH SORTIER ROUTINE
                    2   ;
                    3   ; BEI BEGINN: IN HL ADRESSE DER DATEN
                    4   ;             IN C ZAHL DER ZU SORTIERENDEN ELEMENTE
                    5   ;             (1 < C < 256)
                    6   ;
                    7   ; BEI AUSGANG: DATEN IN AUFSTEIGENDER REIHENFOLGE
                    8   ;              SORTIERT
                    9   ; VERWENDUNG DER REGISTER
                   10   ;
                   11   ; REGISTER INHALTE
                   12   ;
                   13   ; A      ZEITW. SPEICHER FUER BERECHNUNGEN
                   14   ; B      ZAEHLER FUER DATENFELD
                   15   ; C      LAENGE DES DATENFELDES
                   16   ; D      1. ELEMENTE IM VERGLEICH
                   17   ; E      2. ELEMENT IM VERGLEICH
                   18   ; H      BIT 0 - ZEIGT AUSTAUSCH AN (AUSTAUSCHFLAG)
                   19   ; L      NICHT VERWENDET
                   20   ; IX     ZEIGER ZUM DATENFELD
                   21   ; IY     NICHT VERWENDET
                   22   ;
0000  222600       23   SORT:   LD   (DATA), HL    ;RETTEN DER DATENADRESSE
0003  CB84         24   LOOP:   RES  Ø, H          ;ANFWERT ZU AUSTAUSCHFLAG
0005  41           25           LD   B, C          ;ANFWERT ZU LAENGENZAEHLER
0006  05           26           DEC  B             ;EINSTELLEN ZUM TEST
0007  DD2A2600     27           LD   IX,(DATA)     ;ANFWERT ZU FELDZEIGER
000B  DD7E00       28   NEXT:   LD   A,(IX+0)      ;1. ELEMENT IM VERGLEICH
000E  57           29           LD   D,A           ;ZEITWERTSPEICHER F. ELEMENT
000F  DD5E01       30           LD   E,(IX+1)      ;2. ELEMENT IM VERGLEICH
0012  93           31           SUB  E             ;VERGLEICH 1. MIT 2.
0013  3008         32           JRNC NOEX          ;KEIN SPR., WENN 1. > 2.
0015  DD7300       33           LD   (IX),E        ;AUSTAUSCH V. FELDELEMENTEN
0018  DD7201       34           LD   (IX+1),D
001B  CBC4         35           SET  Ø,H           ;NOTIEREN: AUSTAUSCH ERFOLGT
001D  DD23         36   NOEX:   INC  IX           ;NAECHSTE DATENELEMENTADR.
001F  10EA         37           DJNZ NEXT          ;ZAEHLEN DER ANZAHL VON VER-
                   38                              ;GL.,WDH,WENN MEHR DATENPAARE
0021  CB44         39           BIT  Ø,H           ;ERMITTELN,OB VERTAU. ERFOLGT
0023  20DE         40           JRNZ LOOP          ;WEITER,WENN DATEN UNSORT.
0025  C9           41           RET                ;SONST ENDE
                   42           END
```

10.3.2. 16-Bit-Multiplikation

Das folgende Programm multipliziert zwei 16-Bit-Binärzählen ohne Vorzeichen und legt das Ergebnis in das HL-Registerpaar.

```
0000                1   MULT::  GANZZAHLIGE 16-BITMULTIPLIKATION OHNE VORZ.
                    2   ;       BEI EINGANG: FAKTOR 1 IN HL
                    3   ;                    FAKTOR 2 IN DE
                    4   ;
                    5   ;       BEI AUSGANG: ERGEBNIS IN HL
                    6   ;
                    7   ;       VERWENDUNG DER REGISTER
                    8   ;
                    9   ;
                   10   ;       H OBERER ERGEBNISTEIL
                   11   ;       L UNTERER ERGEBNISTEIL
                   12   ;       D OBERER TEIL DES FAKTORS 2
                   13   ;       E UNTERER TEIL DES FAKTORS 2
                   14   ;       B ZAELER FUER ZAHL DER VERSCHIEBUNGEN
                   15   ;       C OBERE BITS DES FAKTORS 1
                   16   ;       A UNTERE BITS DES FAKTORS 1
                   17   ;
```

```
0000  0610      18              LD   B,16;       ANFANGSWERT DER BITANZAHL
0002  4A        19              LD   C,D;        TRANSPORT DES FAKTORS 1
0003  7B        20              LD   A,E;
0004  EB        21              EX   DE,HL;      TRANSPORT DES FAKTORS 2
0005  210000    22              LD   HL,0;       LOESCHEN DES ERGEBNISTEILES
0008  CB39      23  MLOOP:      SRL  C;          VERSCHIEBEN FAKTOR 1 N.RECHTS
000A  1F        24              RR   A;          LETZTES SIGNIFIKANTES BIT
                25  ;                            STEHT IM UEBERTRAG
000B  3001      26              JRNC NOADD;      WENN KEIN UEBERTRAG KEINE AD.
000D  19        27              ADD  HL,DE;      SONST ADDITION DES FAKTORS 2
                28  ;                            ZUM ERGEBNISTEIL
000E  EB        29  NOADD:      EX   DE,HL       LINKSVERSCH. DES FAKTORS 2
000F  29        30              ADD  HL,HL;      DURCH MULT. MIT 2
0010  EB        31              EX   DE,HL;
0011  10F5      32              DJNZ MLOOP;      WDH,BIS KEINE WEITEREN BITS
0013  C9        33              RET;
                34              END;
```

11. Technische Daten

Gesetzliche Grundlage für alle technischen Daten des Bauelementes U880 D bildet der Fachbereichstandard TGL 26 176.
Abnahmeregeln und Prüfverfahren dazu sind in der TGL 24 951 festgelegt.

11.1. Zuverlässigkeitswerte

Prüfzuverlässigkeit

Prüfausfallrate: $\lambda_{PO,\ 6} \leqq 5 \cdot 10^{-5} h^{-1}$

Betriebszuverlässigkeit (garantiert)

Bei mittlerer elektrischer Belastung (Betriebsspannung U_{CC} = 4,75 V bis 5,25 V und Umgebungstemperatur $\vartheta_a \leqq 50\ ^{o}C$), normaler klimatischer Beanspruchung (TGL 24 951) und vernachlässigbarer mechanischer Beanspruchung wird eine Betriebsausfallrate (garantierte Betriebsausfallrate) von

$$\lambda_{BG} \leqq 5 \cdot 10^{-6}\ h^{-1}$$

bezogen auf die durch den Schaltkreis verursachten Funktionsausfälle von Geräten und Anlagen, vom Hersteller garantiert.

Betriebszuverlässigkeit (erwartet)

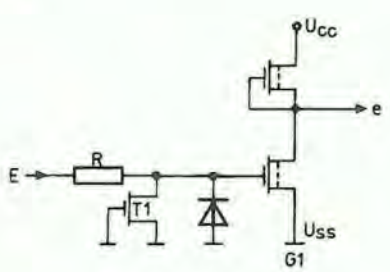
Unter den gleichen Einsatzbedingungen wie sie bei der garantierten Betriebsausfallrate gefordert werden, kann eine Betriebsausfallrate von

$$\lambda_{BE} \leqq 2 \cdot 10^{-6}\ h^{-1}$$

bezogen auf die durch den Schaltkreis verursachten Funktionsausfälle von Geräten und Anlagen, erwartet werden.

11.2. Eingänge und Ausgänge

11.2.1. Eingänge



Reine Eingänge sind die Anschlüsse $\overline{WAIT}$, $\overline{INT}$, $\overline{NMI}$ $\overline{RESET}$, $\overline{BUSRQ}$ und der Takteingang C.
Die Schaltung der Eingänge zeigt Bild 23.

Der Transistor T1 ist ein Enhancementtransistor, der wegen U_{GS} = 0V stets gesperrt ist und mit seinem Durchbruch das Gate des Basistransistors von G1 vor Zerstörung schützt.

Bild 23: Schaltung der Eingänge $\overline{WAIT}$, $\overline{INT}$, $\overline{NMI}$, $\overline{RESET}$, $\overline{BUSRQ}$ und C

11.2.2. Ausgänge

Reine Ausgänge sind die Anschlüsse $\overline{M1}$, $\overline{RFSH}$, $\overline{HALT}$ und $\overline{BUSAK}$. Die Schaltung der Ausgänge zeigt Bild 24.

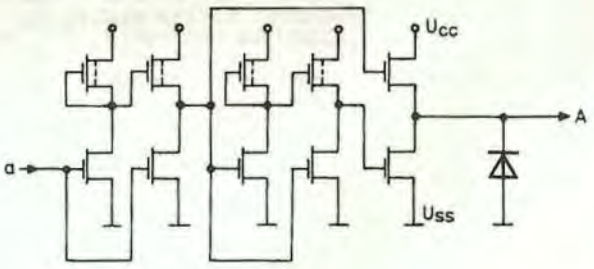


Bild 24: Schaltung der Ausgänge $\overline{M1}$, $\overline{RFSH}$, $\overline{HALT}$ und $\overline{BUSAK}$

11.2.3. Drei-Zustands-Ausgänge

Drei-Zustands-Ausgänge sind die Anschlüsse A0 bis A15, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ und $\overline{WR}$. Die Schaltung zeigt Bild 25. Mit dem Signal HOLD auf log. 1 (H-Pegel) werden die Ausgangstreibertransistoren T1 und T2 in den hochohmigen Zustand gebracht.

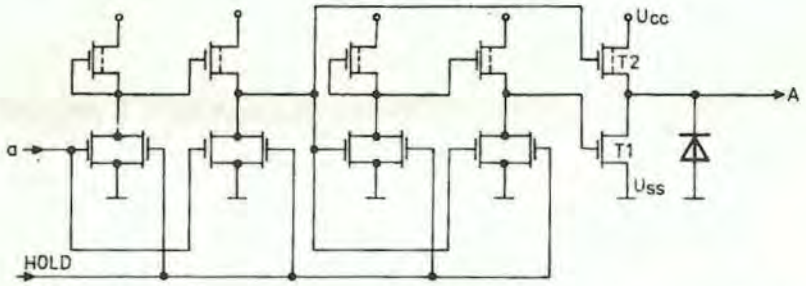


Bild 25: Schaltung der Drei-Zustands-Ausgänge A0...A15, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ und $\overline{WR}$

11.2.4. Drei-Zustands-Ein-und-Ausgänge

Drei-Zustands-Ein-und-Ausgänge sind die Anschlüsse D0 bis D7. Die zugehörige Schaltung zeigt Bild 26.

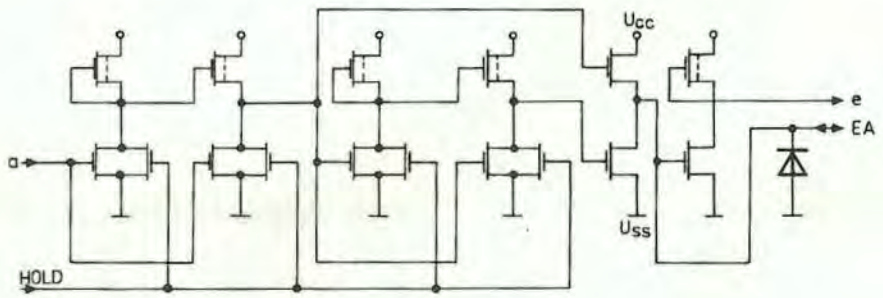


Bild 26: Schaltung der Drei-Zustands-Ein-und Ausgänge D0...D7

11.3. Elektrische Kennwerte

Grenzwerte

bei ϑ_a = Ø bis 70 °C, alle Spannungen bezogen auf U_{ss} = 0V

Kenngröße	Kurzzeichen	Einheit	Kleinstwert	Größtwert	Bemerkung
Betriebsspannung	U_{CC}	V	-0,5	7	
Eingangsspannung	U_I		-0,5	7	
Betriebstemperaturbereich	ϑ_a	°C	0 bis	70	
Lagerungstemperaturbereich	ϑ_{stg}	°C	-55 bis	125	
Verlustleistung	P	W	-	1,1	bei ϑ_a = 25 °C

Statische Kennwerte

Kenngröße	Kurzzeichen	Einheit	Kleinstwert	Größtwert	Bemerkung
Betriebsspannung	U_{CC}		4,75	5,25	ϑ_a = 0 °C...70 °C
Eingangsspannung	U_{IL}		-0,5	0,8	ϑ_a = 0 °C...70 °C
	U_{IH}	V	2	U_{CC}	ϑ_a = 0 °C...70 °C
Takteingangsspannung	U_{ILC}		-0,5	0,45	ϑ_a = 0 °C...70 °C
	U_{IHC}		U_{CC}-0,2	U_{CC}	
Ausgangsspannung	U_{OL}		-	0,4	bei I_{OL} = 1,8 mA, ϑ_a = 0 °C...70 °C
	U_{OH}		2,4	-	bei I_{OH} = -0,25 mA, ϑ_a = 0 °C...70 °C
Eingangsreststrom	I_{LI}		-	10	
Reststrom des Tri-State-Ausgangs im hochohmigen Zustand	I_{LO}		-	10	bei U_I = 0 V und 5,25 V
Reststrom des Datenbusses bei Eingabe	I_{LD}	µA	-	10	ϑ_a = 0 °C und 70 °C
Taktkapazität	C_{CP}		-	50	bei ϑ_a = 25 °C
Eingangskapazität	C_I	pF	-	5	und f=0,5...2 MHz
Ausgangskapazität	C_O		-	10	
Stromaufnahme	I_{CC}	mA	-	200	bei U_{CC} = 5,25 V und ϑ_a = 25 °C

Dynamische Kennwerte

Kenngröße	Kurzzeichen	Einheit	Kleinstwert	Größtwert
Taktperiode	t_c		400	1)
High-Breite des Taktes	$t_{w(CH)}$		180	2000
Low-Breite des Taktes	$t_{w(CL)}$		180	2000
Anstiegs-, Abfallzeiten des Taktes	t_r, t_f		-	30
Bereitstellzeit des $\overline{WAIT}$ vor H/L-Flanke des Taktes	$t_{s(WT)}$		70	-
Bereitstellzeit des $\overline{RESET}$ vor L/H-Flanke des Taktes	$t_{s(RS)}$	ns	90	-
Bereitstellzeit des $\overline{INT}$ vor L/H-Flanke des Taktes	$t_{s(IT)}$		80	-
Impulsbreite von NMI-Low	$t_{w(\overline{NMI})}$		80	-
Bereitstellzeit des $\overline{BUSRQ}$ vor L/H-Flanke des Taktes	$t_{s(BQ)}$		80	-
Datenbereitstellzeit bis zur L/H-Flanke des Taktes im M1-Zyklus (Signal:D0-D7)	$t_{sc(D)}$		50	-
Datenbereitstellzeit bis zur H/L-Flanke des Taktes von M2 bis M5 (Signal:D0-D7)	$t_{s\bar{c}}$		60	-
alle Nachwirkzeiten	t_H		0	-

1) $t_c = t_{w(CH)} + t_{w(CL)} + t_r + t_f$

Verzögerungszeiten

bei: U_{CC} = 4,75 V; U_{IL} = 0,8 V; U_{IH} = 2 V; U_{ILC} = 0,45 V;
U_{IHC}= 4,55 V; C_L = 100 pF und ϑ_a = 70 °C

Kenngröße	Kurzzeichen	Einheit	Kleinstwert	Größtwert
Verzögerungszeit $\overline{MREQ}$ von H/L-Flanke des Taktes bis $\overline{MREQ}$ = H	$t_{DHC(MR)}$		-	110
Verzögerungszeit $\overline{MREQ}$ von L/H-Flanke des Taktes bis $\overline{MREQ}$ = H	$t_{DH\bar{C}(MR)}$		-	110
Verzögerungszeit $\overline{IORQ}$ von L/H-Flanke des Taktes bis $\overline{IORQ}$ = L	$t_{DLC(IR)}$		-	100
Verzögerungszeit $\overline{IORQ}$ von H/L-Flanke des Taktes bis $\overline{IORQ}$ = L	$t_{DL\bar{C}(IR)}$	ns	-	120
Verzögerungszeit $\overline{IORQ}$ von L/H-Flanke des Taktes bis $\overline{IORQ}$ = H	$t_{DHC(IR)}$		-	110
Verzögerungszeit $\overline{IORQ}$ von H/L-Flanke des Taktes bis $\overline{IORQ}$ = H	$t_{DH\bar{C}(IR)}$		-	120
Verzögerungszeit $\overline{RD}$ von L/H-Flanke des Taktes bis $\overline{RD}$ = L	$t_{DLC(RD)}$		-	110
Verzögerungszeit $\overline{RD}$ von H/L-Flanke des Taktes bis $\overline{RD}$ = L	$t_{DL\bar{C}(RD)}$		-	140
Adressenausgangsverzögerungszeit	$t_{D(AD)}$		-	160
Verzögerungszeit bis Floaten	$t_{F(AD)}$		-	110

Kenngröße	Kurzzeichen	Einheit	Kleinstwert	Größtwert
Ausgangsverzögerungszeiten: Daten	$t_{D(D)}$	ns	-	260
Verzögerungszeit bis Floaten bei Schreibzyklus	$t_{F(D)}$		-	90
Verzögerungszeit $\overline{MREQ}$ von H/L-Flanke des Taktes bis $\overline{MREQ}$ = L	$t_{DL\overline{C}(MR)}$		-	110
Verzögerungszeit $\overline{RFSH}$ von L/H-Flanke des Taktes bis $\overline{RFSH}$ = H	$t_{DH(RF)}$		-	160
Verzögerungszeit $\overline{HALT}$ von H/L-Flanke	$t_{D(HT)}$		-	310
Verzögerungszeit $\overline{BUSAK}$ von L/H-Flanke des Taktes bis $\overline{BUSAK}$ = L	$t_{DL(BA)}$	ns	-	130
Verzögerungszeit $\overline{BUSAK}$ von H/L-Flanke des Taktes bis $\overline{BUSAK}$ = H	$t_{DH(BA)}$		-	120
Verzögerungszeit $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$ bis Floaten	$t_{F(C)}$		-	110
Verzögerungszeit $\overline{RD}$ von L/H-Flanke des Taktes bis $\overline{RD}$ = H	$t_{DHC(RD)}$		-	110
Verzögerungszeit $\overline{RD}$ von H/L-Flanke des Taktes bis $\overline{RD}$ = H	$t_{DH\overline{C}(RD)}$		-	120
Verzögerungszeit $\overline{WR}$ von L/H-Flanke des Taktes bis $\overline{WR}$ = L	$t_{DLC(WR)}$		-	90
Verzögerungszeit $\overline{WR}$ von H/L-Flanke des Taktes bis $\overline{WR}$ = L	$t_{DL\overline{C}(WR)}$		-	100
Verzögerungszeit $\overline{WR}$ von H/L-Flanke des Taktes bis $\overline{WR}$ = H	$t_{DH\overline{C}(WR)}$	ns	-	110
Verzögerungszeit $\overline{M1}$ von L/H-Flanke des Taktes bis $\overline{M1}$ = L	$t_{DL(M1)}$		-	145
Verzögerungszeit $\overline{M1}$ von H/L-Flanke des Taktes bis $\overline{M1}$ = H	$t_{DH(M1)}$		-	145
Verzögerungszeit $\overline{RFSH}$ von L/H-Flanke des Taktes bis $\overline{RFSH}$ = L	$t_{DL(RF)}$		-	195

Zusätzliche Zeitangaben

Adresse vor MREQ stabil, Speicher-Zyklus	t_{acm}	$= t_{w(CH)} + t_f - 75$ ns
Adresse vor $\overline{IORQ}$, $\overline{RD}$ oder $\overline{WR}$ stabil, E/A-Zyklus	t_{aci}	$= t_c - 80$ ns
Adresse nach $\overline{RD}$ oder $\overline{WR}$	t_{ca}	$= t_{w(CL)} + t_r - 40$ ns
Adresse nach $\overline{RD}$ oder $\overline{WR}$ beim Floaten stabil	t_{caf}	$= t_{w(CL)} + t_r - 60$ ns
Daten vor $\overline{WR}$ stabil, Speicher-Zyklus	t_{dcm}	$= t_c - 180$ ns
Daten vor $\overline{WR}$ stabil, E/A-Zyklus	t_{dci}	$= t_{w(CL)} + t_r - 180$ ns
Daten nach $\overline{WR}$ stabil	t_{cdf}	$= t_{w(CL)} + t_r - 50$ ns
Impulsbreite von $\overline{MREQ}$-low	$t_{w(\overline{MRL})}$	$= t_c - 40$ ns
Impulsbreite von $\overline{MREQ}$-high	$t_{w(\overline{MRH})}$	$= t_{w(CH)} + t_f - 30$ ns
Impulsbreite von $\overline{WR}$-low	$t_{w(WRL)}$	$= t_c - 40$ ns
M1 vor IORQ stabil (Interrupt-Annahme)	t_{m1}	$= 2\,t_c + t_{w(CH)} + t_f - 80$ ns

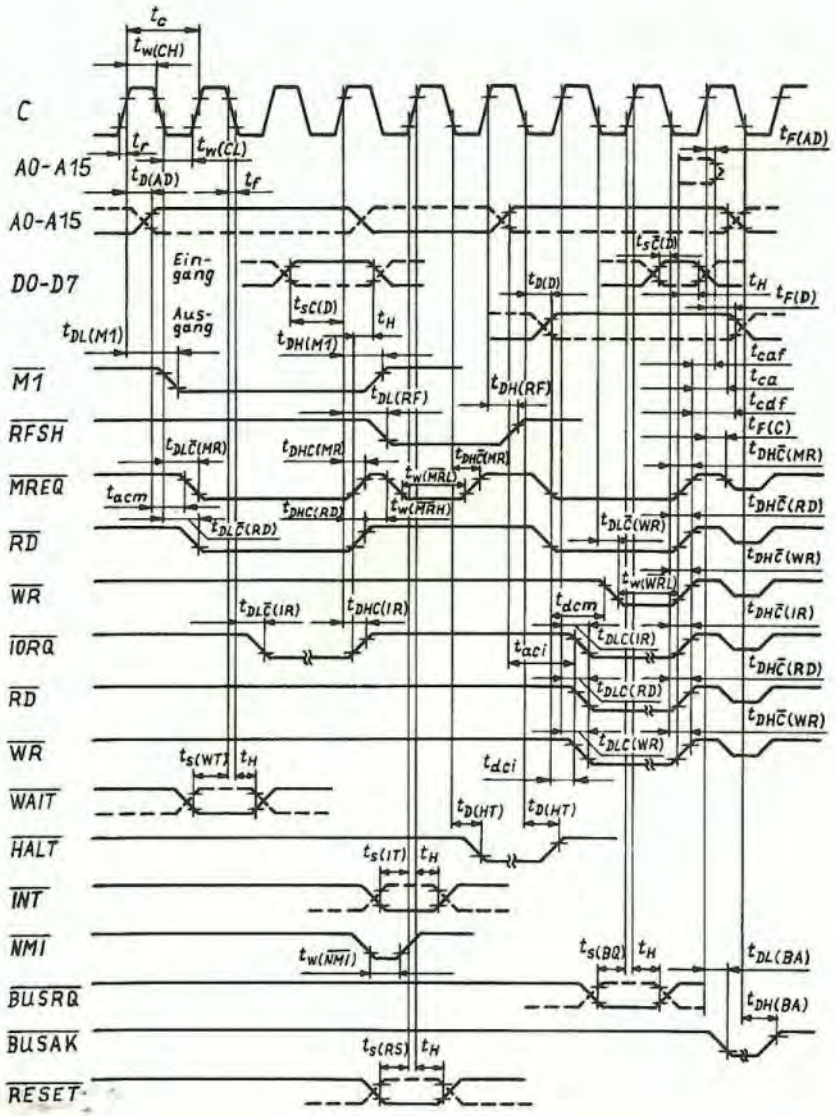


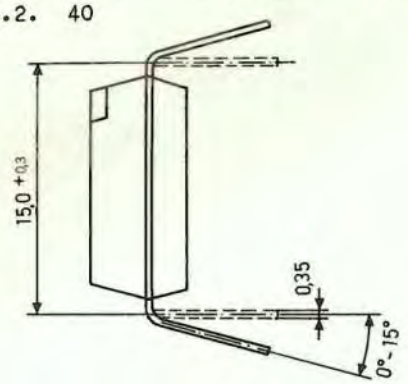
Bild 27: Zeitverhalten - Impulsformen

11.4. Gehäuse

Der Schaltkreis wird in einem 40poligen Plastgehäuse nach TGL 26 713 ausgeliefert

Bauform: 21.2.3.2. 40

Masse: ≈ 5,4 g



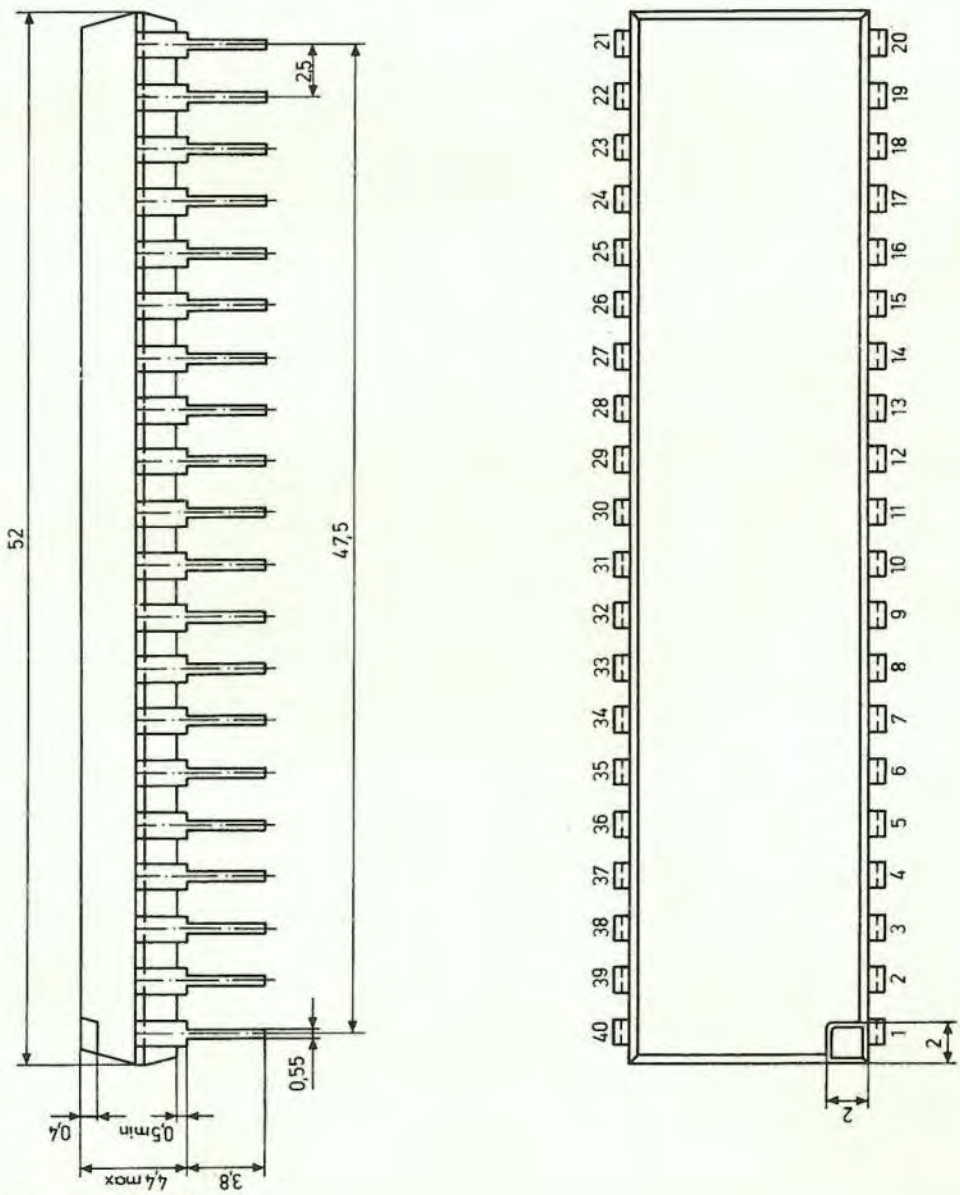


Bild 28: Gehäuseabmessungen U880 D

Rs 01.82–53 V 7 1 2354 KO

elektronik
export·import
VOLKSEIGENER AUSSENHANDELSBETRIEB DER
DEUTSCHEN DEMOKRATISCHEN REPUBLIK
DDR 1026 BERLIN ALEXANDERPLATZ
HAUS DER ELEKTROINDUSTRIE

VEB FUNKWERK ERFURT
im VEB Kombinat Mikroelektronik
DDR - 5010 Erfurt, Rudolfstraße 47
Telefon: 5 80
Telex: 061 306
Kabel: FUNKWERK ERFURT

Rs 01-82-51 W-V-2-1