SIEMENS

Datenbuch 1979/80

Mikrocomputer Bausteine

Mikroprozessor

System SAB 8080

SIEMENS

Mikrocomputer-Bausteine

Datenbuch 1979/80

Mikroprozessor System SAB 8080

SIEMENS AKTIENGESELLSCHAFT

Liebe Leser!

Technische Dokumentation geben wir mit dem Ziel heraus, Sie beim Einsatz unserer Produkte zu unterstützen. Bei der Erarbeitung von Form und Inhalt der benötigten Information sind wir jedoch auch auf Ihre Hilfe angewiesen.

Wertvolle Mitarbeit bei der Verbesserung unserer Produktinformation können Sie durch Hinweise zu folgenden Fragen leisten:

1. Welche Begriffe oder Beschreibungen sind unverständlich?
2. Welche Ergänzungen und Erweiterungen schlagen Sie vor?
3. Wo haben sich inhaltliche Fehler eingeschlichen?
4. Welche Druckfehler haben Sie gefunden?

Antworten und sonstige Anregungen richten Sie bitte an:

Siemens Aktiengesellschaft
Unternehmensbereich Bauelemente
Vertrieb/Produktinformation
Balanstraße 73
8000 München 80

Zugehörige Druckschriften

Benötigen Sie zur Ergänzung Ihrer Informationen weitere technische Unterlagen, so fordern Sie bitte die aktuelle Angebotsliste »Produktinformation zum Thema Mikrocomputer« an. Die halbjährlich neu erscheinende Angebotsliste mit anhängender Bestellkarte bekommen Sie bei Ihrer nächstgelegenen Siemens-Dienststelle (siehe Geschäftsstellenverzeichnis)

Herausgegeben von
Siemens AG, Bereich Bauelemente, Balanstraße 73, 8000 München 80.

Liefermöglichkeiten und technische Änderungen vorbehalten.

Fragen über Technik, Preise und Liefermöglichkeiten richten Sie bitte an unsere Zweigniederlassungen im Inland, Abteilung VB oder an unsere Landesgesellschaften im Ausland (siehe Geschäftsstellenverzeichnis).

Inhaltsverzeichnis

1. Einführung

1.1. Allgemeines

Seit ihrer Entstehung sind Digital-Computer ständig leistungsfähiger geworden und durch jede größere technische Verbesserung in neue Anwendungsbereiche vorgedrungen. Das Erscheinen des Minicomputers machte die Einbeziehung von Digital-Computern in verschiedene Prozeß-Kontrollsysteme als einen selbstverständlichen Bestandteil möglich. Leider schränken Größe und Kosten von Minicomputern ihren Einsatz für spezielle Anwendungen ein. Daher verwendete man in solchen Fällen kundenspezifische Systeme aus „festverdrahteter Logik", d.h. aus Gattern, Flip-Flops, Zählern, usw.

Der erhebliche Aufwand an Zeit und Kosten für die Entwicklung dieser Systeme, einschließlich der Fehlerbeseitigung, haben jedoch ihren Einsatz auf Anwendungen mit großen Stückzahlen begrenzt, bei denen die Entwicklungskosten über eine große Anzahl von Bausteinen verteilt werden konnten.

Heute wird dem Systementwickler eine neue Alternative angeboten: „der Mikrocomputer". Unter Ausnutzung der Erfahrungen und Technologien, die man in der Entwicklung von LSI-Speicherbauelementen sammeln konnte, wurde jetzt die Leistungsfähigkeit des Computers dem Anwender in Form von großintegrierten Computerbausteinen zugänglich gemacht. Mit dem n-Kanal Silizium Gate-MOS-Verfahren konnte der schnelle (2 µs-Zyklus) und leistungsfähige (72 Grundbefehle) Mikroprozessor SAB 8080A auf einem einzigen LSI-Baustein realisiert werden. Wird dieser Mikroprozessor mit Speicher- und Eingabe-Ausgabe (E/A)-Bausteinen kombiniert, ergibt sich ein vollständiger Computer.

Der Mikroprozessor SAB 8080A ist in einem „dual-in-line" Gehäuse (DIP) mit 40 Anschlüssen untergebracht. Der SAB 8080A hat einen 16-Bit-Adreßbus, einen 8-Bit-Zweiweg-Datenbus und voll dekodierte, TTL-kompatible Ausgänge. Zusätzlich zur Anschlußmöglichkeit bis zu 64 K-Bytes gemischter RAM und ROM, kann der SAB 8080A max. 256 Eingabe und Ausgabe-Bausteine adressieren. Der Befehlssatz SAB 8080A umfaßt folgende Anweisungen: Datentransferbefehle, arithmetische und logische Operationen, Befehle zur Programmverzweigung und Sonderbefehle. Der SAB 8080A Befehlssatz ist leistungsfähig genug, es mit vielen der erheblich teureren Minicomputer aufzunehmen.

Im Gegensatz zu festverdrahteten Logikschaltungen, die gewöhnlich funktionsparallel arbeiten, führt der Mikrocomputer seine Aufgaben mit einem sequentiellen Anwenderprogramm durch. Aus dieser sequentiellen Operationsweise ergibt sich, daß die Anzahl der Aufgaben, die ein Mikrocomputer in einer gegebenen Zeiteinheit durchführen kann, direkt proportional der Befehlsausführungsgeschwindigkeit des Mikroprozessors ist.

Der SAB 8080A enthält einen 16-Bit-Stackpointer (Kellerspeicherzeiger), der die Adressierung des externen RAM-residenten Stack steuert und einen beliebigen Teil des RAM-Speichers als „last-in/first-out"-Stack verwendet. Er ermöglicht damit eine nahezu unbegrenzte Verschachtelungstiefe von Unterprogrammen. Mit Hilfe des Stackpointers kann man den Inhalt des Befehlszählers, des Akkumulators, der Zustandskennbits oder irgendeines der Datenregister im Stack speichern oder von dort abrufen. Darüber hinaus ist eine mehrstufige Programmunterbrechung möglich. Die Zustandsinformation („status") des Prozessors kann in den Stack eingeschrieben werden, wenn man ein Unterbrechungssignal erhält und kann wieder aus dem Stack herausgenommen werden, wenn die Unterbrechungsbearbeitung ausgeführt worden ist. Diese Fähigkeit, den Inhalt der Register des Prozessors zu erhalten, ist auch dann gegeben, wenn ein Unterbrechungsprogramm selbst unterbrochen wird.

Vorteile beim Aufbau mit Mikrocomputern

Mikrocomputer vereinfachen nahezu jede Phase einer Produktentwicklung. Der erste Schritt ist bei jeder Produktentwicklung die Klarlegung der verschiedenen Funktionen, die das fertige System durchführen soll. Anstatt diese Funktionen mit Gattern und Flipflops zu realisieren, werden sie durch die Erstellung geeigneter Befehlsfolgen, des Programms, definiert und in Speicherelemente eingegeben. Daten werden im allgemeinen im RAM, das Anwenderprogramm in einem ROM gespeichert. Der Mikroprozessor führt alle Systemfunktionen durch, indem er die Befehle aus dem Speicher abruft, sie ausführt und die Ergebnisse über die Eingabe/Ausgabe Bausteine des Mikrocomputers weiterleitet. Ein Mikroprozessor SAB 8080A kann mit einem Programm, das in einem einzigen 2048 Byte ROM-Baustein gespeichert ist, die gleichen Funktionen ausführen, die bisher 1000 Gatter erforderten.

Die Vorteile beim Aufbau eines Systems mit einem Mikrocomputer gehen über eine vereinfachte Produktentwicklung hinaus. Darunter ist vor allem eine erhebliche Einsparung an Hardware-Kosten zu nennen. Neben den Kosten wird auch die Größe des Systems reduziert und die Produktionskosten vermindern sich durch die niedrigere Anzahl der Bauelemente. Weitere Einsparungen ergeben sich durch weniger Aufwand beim Durchführen von Änderungen (es braucht lediglich das Anwender-Programm, nicht die Schaltung geändert zu werden), niedrigere Ausfall- und Garantiekosten durch höhrere Zuverlässigkeit wegen der geringeren Anzahl der Bausteine und Lötverbindungen und durch Verringerung der Platinenkosten (geringere Abmessungen, Komplexität, weniger Anschlüsse, niedrigere Prüf- und Korrekturkosten).

Vorteile beim Aufbau mit Mikroprozessoren

	Konventionelles System	Programmierte Logik
Produktbeschreibung		Vereinfacht, da bestimmte Eigenschaften leicht integrierbar
Systemaufbau und logischer Aufbau	mit Logik-Diagrammen	Programmierhilfsmittel verfügbar (Compiler, Assembler, Editor-Programme)
Fehlerbeseitigung	mit konventionellen Laborgeräten	Software- und Hardware-Hilfsmittel verkürzen die benötigte Zeit
Platinen-Layout		weniger Platinen zu entwickeln
Dokumentation		weniger Hardware zu dokumentieren
Kühlungs- und Gehäuse-Fragen		geringere Systemgröße und verminderte Leistungsaufnahme
Stromversorgung		niedrigerer Leistungsverbrauch
System-Änderungen	mit Draht	Programm ändern ohne Hardware-Änderung

Mikrocomputer Entwicklungs-Hilfsmittel

Wenn Sie es gewohnt sind, logische Schaltungen zu entwickeln, wird Ihnen die Vorstellung einer programmierten Logik als ein zu radikaler Wechsel erscheinen. Seien Sie ohne Sorge: Wir haben bereits einen großen Teil der grundlegenden Arbeiten für Sie ausgeführt. Das SME (Siemens Mikrocomputer Entwicklungssystem) bietet flexible, kostengünstige und einfache Entwicklungsmethoden. Das beim SME mitgelieferte Standard-Software-Paket enthält neben einem umfassenden Betriebssystem komfortable Dienstprogramme und auch einen leistungsfähigen Macro-Assember für den SAB 8080A. Darüber hinaus werden verschiedene Compiler für höhere Programmiersprachen, wie PL/M, FORTRAN, BASIC usw., ablauffähig auf dem SME, angeboten. Für den effizienten Test von Soft- und Hardware steht ein Emulations- und Testadapter (ETA) zur Verfügung. SIEMENS steht Ihnen in jeder Phase Ihres Produktentwicklungsprojektes als Berater zur Seite.

Wir versehen Sie auch mit einer Dokumentation über sämtliche Hardware- und Software-Produkte.

Anwendungsbeispiele

Der SAB 8080A kann als Grundelement für eine Vielfalt von Rechen- und Steuersystemen verwendet werden. Der Systemaufbau wird sich bei einer Anwendung jeweils nach der Art der Peripheriegeräte richten und auch von der Menge und der Art

der verwendeten Speicherkapazität abhängen. Die Anwendungen und Lösungswege, die in diesem Kapitel besprochen werden, sollen nur Beispiele darstellen, die zeigen, wie Mikrocomputer zur Lösung von Entwicklungsproblemen verwendet werden können. Auf keinen Fall sollte der Eindruck entstehen, daß der SAB 8080A hinsichtlich des Anwendungsbereiches oder seiner Leistungsfähigkeit auf die hier beschriebenen Anwendungen begrenzt ist.

Das System SAB 8080 in einer automatischen preisrechnenden Waage

Das Grundsystem besteht aus 2 Eingabeelementen: Dem Wiegesystem und der Tastatur zur Auswahl der Funktionen und Eingabe des Grundpreises. Das Ausgabeelement ist eine Anzeige, die das Gesamtgewicht anzeigt; es kann auch ein Papierstreifendrucker als zusätzliches Ausgabegerät angeschlossen werden. Die Steuereinheit nimmt vom Wiegesystem Gewichtsinformationen auf, erhält Funktions- und Grundpreis-Angaben von der Tastatur und erzeugt die Daten für die Anzeige. Die einzige arithmetische Funktion, die durchzuführen ist, ist eine einfache Multiplikation des Gewichts mit dem Grundpreis.

Die Steuereinheit kann auch mit Standard TTL-Logikbausteinen realisiert werden. Es können Zustandsdiagramme für die verschiedenen Teilabschnitte erstellt und eine Multiplikationsschaltung entwickelt werden.

Man wird dann die Schaltung aufbauen, eine Auswahl von Standardbausteinen vornehmen und die Platine ausarbeiten. Eine derartige Entwicklung mit Bausteinen aus einer Standardlogikfamilie, wie in unserem Beispiel mit TTL, führt zu anwendungsspezifischen Schaltungen, bedingt durch die verwendeten Bausteine, als auch durch die Verdrahtung zu einer Logikschaltung. Wird ein Mikrocomputer zur Realisierung der Steuerschaltung eingesetzt, wie in Bild 1 dargestellt ist, dann ist die einzige anwendungsspezifische Logik die der Anpassungsschaltung. Diese Schaltung ist sehr einfach. Sie stellt lediglich Puffer für die Eingabe- und Ausgabe-Signale dar.

Der Systementwickler entwirft nun einen Datenflußplan, der zeigt, welche Eingangssignale gelesen werden müssen, welche Signale und Berechnungen benötigt werden und welche Ausgangssignale erstellt werden müssen.

Danach wird ein symbolischer Programmablaufplan erstellt, der anschließend in einer 8080-Sprache kodiert wird. Das so entstandene Programm wird maschinell in Bitmuster übersetzt, die in den Programmspeicher geladen werden. Das System erhält also seine anwendungsspezifischen Funktionseigenschaften durch den Inhalt seines Programmspeichers.

Für die automatische Waage wird das Programm in einen Festwertspeicher eingegeben, da der Mikrocomputer ein immer gleichbleibendes Programm für die Waagenfunktionen abzuwickeln hat. Der Prozessor wird ständig Tastatur und Waageeinrichtung abfragen und ggf. die Anzeigedaten erneuern. Die Schaltung erfordert nur wenig Speicherkapazität, da diese nur für die Speicherung des Grundpreises, der Zwischenergebnisse und der Anzeigedaten benötigt wird.

Wenn die Steuereinheit eines Gerätes mit einem Mikrocomputer-Bausteinsatz realisiert wird, können vorhandenen Funktionen durch eine Änderung des gespeicherten

Programms leicht neue Funktionen hinzugefügt werden. Bei einem System mit TTL Bausteinen erfordern Änderungen dagegen komplizierte Eingriffe in die Verdrahtung, die Platinen usw.

Der große Anwendungsbereich, der mit einem Mikrocomputer möglich ist, wird nur durch die Vorstellungskraft des Entwicklers begrenzt. Wir haben in der nachfolgenden Tabelle einige geeignete Anwendungen zusammengestellt und eine Gegenüberstellung mit den dafür normalerweise verwendeten Peripheriegeräten.

Bild 1

Mikrocomputer – Anwendung
Preisrechnende Waage

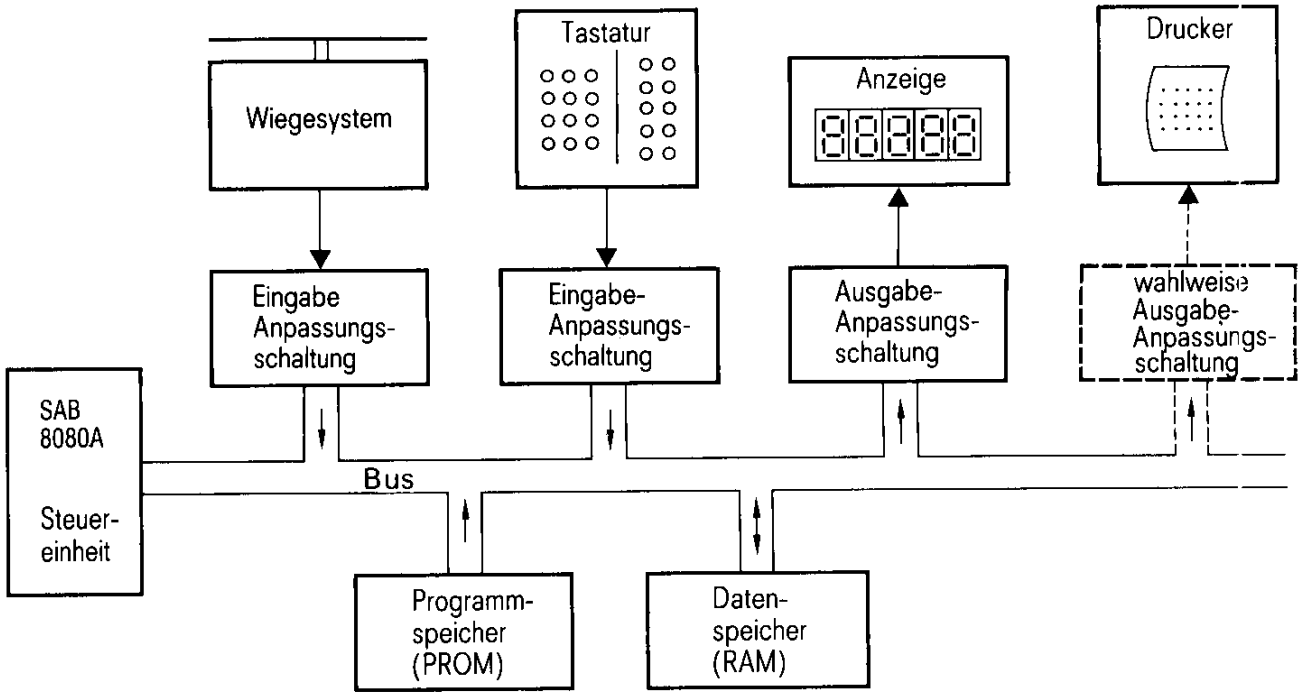


Tabelle über Anwendungen

Intelligente Terminals	Datensichtgeräte Drucker Synchrone und asynchrone Datenübermittlung Kassetten-Bandgeräte Tastaturen
Spielautomaten	Tastaturen, Druckkontakte und Schalter Verschiedene Anzeigen Münzen-Eingabe Münzen-Ausgabe
Registrierkassen	Tastatur oder Eingangsschalter – Anordnung Wechselgeld-Ausgabe Digitale Anzeige Papierstreifendrucker Magnetische Kartenleser Datenverkehr-Anpassungsschaltung
Buchhaltungs- und Belegmaschinen	Tastatur Drucker Kassette oder anderes Magnetbandgerät Floppy-Disks
Telefon-Schaltwerke	Telefonleitungs-Abfrage Anlage Schaltnetzwerke Wählscheibenregister Service-Klassifizierungseinheit
Numerisch gesteuerte Werkzeugmaschinen	Magnetband- oder Lochstreifenleser Schrittmotoren Optische Drehscheiben-Kodierer
Prozeß-Steuerung	Analog/Digital Wandler Digital/Analog Wandler Steuerschalter Anzeigen

1.2. Arbeitsweise eines Computers

Der folgende Abschnitt erläutert einige grundlegende Begriffe, die zum Verständnis eines Computers nötig sind. Er liefert die Hintergrundinformationen und die Begriffserläuterungen, die für die später folgenden Abschnitte benötigt werden.

Ein typisches Computer-System

Der Prozessor kann alle Operationen in einem Computer-System, wie dem SME, selbständig ausführen. Er kann aber nicht auf sich allein gestellt ein sinnvolles Endresultat produzieren, er muß vielmehr ständig mit anderen Systemkomponenten zusammenarbeiten. Dies sind zum Beispiel Speicher oder E/A-Schaltungen. Bei der Beschreibung irgendeines speziellen Bausteins muß man daher ständig auf alle anderen Bausteine des Systems Bezug nehmen. Somit ist von großer Bedeutung, zunächst die prinzipielle Funktionsweise eines Computers zu kennen, bevor man den Prozessor genauer betrachtet.

Die Bestandteile eines typischen Computer-Systems sind:

a) eine Zentraleinheit
b) der Speicher
c) die Ein/Ausgabekanäle (E/A-Kanäle)

Im Speicher werden **Befehle** und **Daten** gespeichert. Befehle sind codierte Informationen, die das Verhalten des Rechners steuern. Daten sind codierte Informationen, die von der Zentraleinheit verarbeitet werden. Eine Folge logisch zusammenhängender Befehle, die im Speicher abgelegt sind, heißt ein **Programm**. Die Zentraleinheit liest die Befehle in einer festgelegten Reihenfolge aus dem Speicher aus und benutzt sie dazu, Arbeitsgänge zu starten. Wenn die Programmfolge zusammenhängend und sinnvoll ist, kommt bei der Arbeit der Zentraleinheit ein sinnvolles und brauchbares Resultat heraus.

Der Speicher wird sowohl dazu benutzt, Daten zur Verarbeitung bereitzuhalten, wie auch die Befehle, die diese Verarbeitung steuern. Das Programm muß so organisiert sein, daß die Zentraleinheit stets einen Befehl und kein Datenwort aus dem Speicher liest, wenn sie einen Befehl erwartet. Die Zentraleinheit hat schnellen Zugriff auf alle Daten im Speicher, aber häufig ist der Speicher nicht groß genug, um die gesamten Daten zu speichern, die für einen bestimmten Zweck notwendig sind. Das Problem kann dadurch gelöst werden, daß man den Computer mit mehreren **Eingabekanälen** ausstattet. Die Zentraleinheit kann diese Kanäle adressieren und die in ihnen enthaltenen Eingabedaten übernehmen. Die Ausstattung mit Eingabekanälen macht es dem Computer möglich, Information von Peripheriegeräten zu empfangen (etwa einem Lochstreifenleser oder einer Floppy Disk). Die Eingabedaten können dabei sehr schnell und in großen Mengen übermittelt werden.

Ein Computer benötigt auch einen oder mehrere **Ausgabekanäle**, die es der Zentraleinheit ermöglichen, das Ergebnis der Berechnungen an die Außenwelt weiterzuleiten. Die Ausgabe kann über ein Bildschirmgerät dem Bedienungspersonal mitgeteilt werden oder sie kann an ein Peripheriegerät gerichtet werden, das sie auf einen Datenträger überträgt, wie das bei einem Matrixdrucker der Fall ist. Sie kann aber auch an einen peripheren Speicher, etwa an eine Floppy Disk übergeben werden. Schließlich kann der Computer Prozeß-Steuersignale ausgeben, die das

Verhalten eines anderen Systems beeinflussen, zum Beispiel eines automatisierten Fließbandes. Ausgabekanäle können wie die Eingabekanäle über Adressen angesteuert werden. Zusammen ermöglichen die Eingabe- und Ausgabekanäle die Kommunikation zwischen der Zentraleinheit und der Außenwelt.

Die **Zentraleinheit** koordiniert das ganze System und überwacht das Verhalten der übrigen Bausteine. Sie muß in der Lage sein, Befehle aus dem Speicher auszulesen, ihren binär verschlüsselten Inhalt zu decodieren und die Befehle auszuführen. Sie muß außerdem den Speicher und die E/A-Kanäle adressieren können, soweit dies bei der Ausführung der Befehle erforderlich wird. Auch sollte sie auf gewisse, von außen eingegebene Steuersignale reagieren können, zum Beispiel Unterbrechungs- (INTERRUPT-) und Warte- (WAIT-) Aufforderungen.

Nachstehend sind nun die funktionellen Untereinheiten der Zentraleinheit beschrieben, die ihr die Durchführung dieser Aufgaben ermöglichen.

1.3. Aufbau einer Zentraleinheit

Eine typische Zentraleinheit enthält folgende Funktionseinheiten:

- Register
- eine Arithmetik-Logik-Einheit (ALU, arithmetic/logic unit)
- eine Steuerlogik

Register sind Zwischenspeicher innerhalb der Zentraleinheit. Einige Register, z.B. der Befehlszähler und das Befehlsregister, haben spezielle Aufgaben. Andere Register, z.B. der Akkumulator, dienen mehr allgemeinen Zwecken.

Akkumulator

Der Akkumulator speichert im allgemeinen einen der Operanden, die von der ALU verarbeitet werden. Ein typischer Befehl kann zum Beispiel veranlassen, daß die ALU den Inhalt eines anderen Registers auf den Inhalt des Akkumulators aufaddiert und das Ergebnis wieder im Akkumulator abspeichert.

Im allgemeinen kann also der Akkumulator sowohl Operanden- als auch Resultatregister einer Operation sein.

Oft enthält die Zentraleinheit eine Anzahl zusätzlicher Register für allgemeine Zwecke. Diese können dazu benutzt werden, Zwischenresultate oder Operanden zu speichern. Wenn solche zusätzlichen Register vorhanden sind, entfällt die Notwendigkeit, Zwischenresultate laufend zwischen dem Speicher und dem Akkumulator hin- und herzuschieben. Dadurch steigt die Arbeitsgeschwindigkeit und die Gesamteffektivität.

Befehlszähler (Sprünge, Unterprogrammaufrufe und Stack)

Die Befehle, aus denen ein Programm besteht, sind im Speicher des Computers abgespeichert. Die Zentraleinheit holt sich diese Befehle aus dem Speicher und stellt so fest, welche Operationen sie ausführen muß. Das bedeutet, daß der Prozessor immer wissen muß, auf welcher Adresse der nächste Befehl zu finden ist, damit der logisch vorgeschriebene Programmablauf gesichert ist.

Jeder Speicherplatz ist numeriert und unterscheidet sich durch seine Nummer von allen anderen Speicherplätzen. Diese Nummer, die einen Speicherplatz charakteris ert, nennt man seine **Adresse**.

Der Prozessor enthält nun ein Zählregister, das immer die Adresse des nächsten Programmbefehls enthält. Es wird **Befehlszähler** genannt. Jedesmal, wenn ein Befehl geholt wird, bringt der Prozessor den Befehlszähler auf den aktuellen Stand, in dem er ihn um 1 erhöht.

Der Programmierer muß seine Befehle daher in Speicherplätzen mit aufeinanderfolgenden Adressen ablegen. Die niedrigeren Adressen müssen den zuerst auszuführenden Befehlen und die höheren den später auszuführenden Befehlen entsprechen. Diese Reihenfolge darf nur dann mißachtet werden, wenn in einem Speicherbereich ein **Sprungbefehl** zu einem anderen Speicherbereich abgelegt wird. Ein solcher Sprungbefehl enthält die Adresse des Befehls, der im Anschluß an ihn ausgeführt werden soll. Dieser nächste Befehl kann sich in einer beliebigen Speicherzelle befinden, solange der programmierte Sprungbefehl seine richtige Adresse angibt. Bei der Ausführung eines Sprungbefehls ersetzt der Prozessor den Inhalt des Befehlszählers durch die Adresse, die der Sprungbefehl angibt. Dadurch wird die korrekte Abarbeitung einer Befehlsfolge gesichert.

Ein Spezialfall des Sprungbefehls ist der **Unterprogrammaufruf**. Dabei merkt sich der Prozessor den Inhalt des Befehlszählers zu dem Zeitpunkt, in dem der Sprungbefehl interpretiert wird. Das gibt dem Prozessor die Möglichkeit, die Ausführung des Hauptprogramms fortzusetzen, wenn der letzte Befehl des Unterprogramms ausgeführt ist.

Ein **Unterprogramm** ist ein Programm innerhalb eines anderen Programms. Im allgemeinen ist es eine Folge von Befehlen, die im Verlauf des Hauptprogramms mehrfach ausgeführt werden muß. Gute Beispiele für Unterprogramme sind Routinen, die das Quadrat, den Sinus oder den Logarithmus einer Programmvariablen berechnen. Weitere Beispiele sind Programme, die für die Datenein- oder -ausgabe mit bestimmten Peripheriegeräten geschrieben wurden.

Um eine korrekte Rückkehr zum Hauptprogramm zu gewährleisten, benutzt der Prozessor eine spezielle Organisationsform. Bei der Ausführung eines Unterprogramm-Aufrufbefehls erhöht er den Befehlszähler und legt seinen Inhalt in einem speziellen Speicherbereich ab, den man **Stack** (Stapelspeicher) nennt. Der Stack bewahrt also die Adresse des Befehls auf, der nach der Ausführung des Unterprogramms verarbeitet werden muß. Anschließend lädt der Prozessor die Adresse, die durch den Unterprogramm-Aufruf angegeben wird, in den Befehlszähler. Der als nächster vom Speicher geholte Befehl ist daher der erste im Unterprogramm.

Die letzte Anweisung in jedem Unterprogramm ist ein **Rückkehrbefehl**. Ein solcher Befehl braucht keine Adresse anzugeben. Wenn der Prozessor einen Rückkehrbefehl ausführt, ersetzt er einfach den momentanen Inhalt des Befehlszählers durch die Adresse an der Spitze des Stack. Das führt dazu, daß der Prozessor die Ausführung des aufrufenden Programms fortsetzt und zwar an der Stelle, die unmittelbar dem ursprünglichen Unterprogrammaufruf folgt.

Unterprogramme sind häufig **verschachtelt**, das heißt, ein Unterprogramm kann manchmal ein weiteres aufrufen. Das zweite kann wieder ein drittes aufrufen und so fort. Das ist in Ordnung, solange die Kapazität des Rechners ausreicht, um die notwendigen Rückkehradressen zu speichern und die Aufrufe auch von der

Programmlogik her richtig sind. Mit anderen Worten heißt das, daß die maximale Verschachtelungstiefe für Unterprogramme nur durch die Kapazität des Stack begrenzt wird. Wenn der Stack z.B. drei Rückkehradressen aufnehmen kann, können drei Unterprogrammebenen verarbeitet werden.

Die Stackorganisation kann sich bei den einzelnen Prozessoren unterscheiden. Bei einigen kann die Rückkehradresse im Prozessor selbst gespeichert werden. Andere benutzen einen reservierten Bereich des externen Speichers als Stack und verwenden einfach ein **Zeiger-Register** (Pointer-Register), das die Adresse des obersten Stackelementes angibt. Dieser externe Stack erlaubt praktisch unbeschränkte Unterprogrammverschachtelung. Wenn der Prozessor außerdem noch Befehle besitzt, mit denen er den Inhalt des Akkumulators oder anderer Register auf dem Stack abspeichern (PUSH-Operationen) oder von ihm herunterlesen (POP-Operationen) kann, dann werden mehrstufige Unterbrechungen möglich. Der Zustand des Prozessors, das heißt die Inhalte seiner sämtlichen Register können dann nämlich im Stack festgehalten werden, wenn eine Unterbrechung erfolgt. Im Anschluß an diese stehen sie dann wieder zur Verfügung. Diese Möglichkeit, den Zustand des Prozessors jederzeit konservieren zu können, bleibt auch erhalten, wenn ein Programm zur Behandlung einer Unterbrechungsaufforderung selbst noch einmal unterbrochen wird.

Befehlsregister und Befehlsdekodierer

Jeder Computer besitzt eine für ihn charakteristische **Wortlänge**. Die Wortlänge eines Computers wird gewöhnlich durch die Größe seiner internen Speichereinheiten (der Register) bestimmt, sowie durch seine Verbindungswege (Busse genannt). Ein Computer, dessen Register und Busse acht Bit an Information speichern bzw. übertragen können, hat eine charakteristische Wortlänge von acht Bit. Für einen solchen Rechner ist es am effektivsten, acht Bit lange Informationseinheiten zu bearbeiten. Der Speicher muß in diesem Fall so organisiert sein, daß in jeder adressierbaren Speicherzelle acht Bit gespeichert werden können. Daten und Befehle sind dann im Speicher als acht Bit lange Binärzahlen abgelegt oder als Zahlen, deren Länge ein ganzzahliges Vielfaches von acht ist: 16 Bit, 24 Bit, usw.

Ein solches charakteristisches 8-Bit-Wort wird häufig auch **Byte** genannt.

Jede Operation, die der Prozessor ausführen kann, wird eindeutig durch ein Byte gekennzeichnet, das **Befehlscode** oder **Operationscode** genannt wird. Mit 8-Bit-Worten lassen sich 256 verschiedene Aktionen charakterisieren, mehr sind für die meisten Computer nicht notwendig.

Der Prozessor holt einen Befehl in zwei aufeinanderfolgenden Schritten. Im ersten übergibt er die Adresse, die in seinem Befehlszähler steht an den Speicher, im zweiten übergibt der Speicher das angeforderte Byte an den Prozessor. Die Zentraleinheit speichert dieses Befehlswort in einem Register, dem **Befehlsregister**, und macht von ihm die nachfolgende Befehlsausführung abhängig.

Der Mechanismus, mit dessen Hilfe der Computer einen Befehlscode in entsprechende Operationen umsetzt, erfordert mehr Eindringen ins Deteil, als an dieser Stelle möglich ist. Das Grundkonzept sollte jedoch intuitiv jedem klar sein, der Logikschaltungen entwirft. Die acht Bit im Befehlsregister werden decodiert und in Abhängigkeit davon wird eine Auswahl der Ausgangsleitungen aktiviert. Im Fall

des 8-Bit-Prozessors können es bis zu 256 Ausgänge sein. Jedem Ausgang ist eine Anzahl von Operationen zugeordnet, die zur Ausführung eines bestimmten Befehlscodes nötig sind. Gleichzeitig können die aktivierten Ausgänge mit bestimmten Taktimpulsen belegt werden. Dadurch werden elektrische Signale erzeugt, mit deren Hilfe man bestimmte Operationen starten kann. Diese Übersetzung des Befehlscodes in Operationen wird vom **Befehlsdecodierer** und der zugehörigen Steuerlogik vorgenommen.

Ein acht Bit langes Codewort reicht häufig aus, um eine bestimmte Operation festzulegen. Manchmal erfordert die Ausführung eines Befehls aber mehr Information als sich mit einem 8-Bit-Wort übermitteln läßt.

Ein Beispiel für diesen Fall sind Befehle, die Speicherzellen adressieren. Der Operationscode gibt in diesem Fall nur die Operation selbst an. Er kann aber nicht auch noch die Adresse enthalten. In einem solchen Fall muß der Befehl zwei oder drei Bytes umfassen. Aufeinanderfolgende Bytes eines Befehls sind in aufeinanderfolgenden Speicherzellen abgespeichert. Der Prozessor muß dann zwei oder drei Lesezyklen durchlaufen, um den Befehl vollständig aus dem Speicher zu holen. Das erste Byte wird im Befehlsregister abgelegt und die nachfolgenden in Hilfsspeicherzellen. Erst anschließend beginnt der Prozessor, den Befehl auszuführen.

Adreßregister

Eine Zentraleinheit kann ein Register oder ein Registerpaar dazu benutzen, die Adresse einer Speicherzelle zu speichern, auf deren Daten zugegriffen werden soll. Wenn das Adreßregister **programmierbar** ist, d.h. wenn es Befehle gibt, die es dam Programmierer gestatten, den Inhalt des Adreßregisters zu verändern, kann sich das Programm die Adresse für einen **Speicherbefehl** selbst erzeugen, d.h. für einen Befehl, der Daten aus dem Speicher liest, sie dort einschreibt oder ändert.

Arithmetik-Logik-Einheit (ALU)

Alle Prozessoren besitzen eine Arithmetik-Logik-Einheit, die ALU. Wie schon ihr Name sagt, ist die ALU der Teil der Zentraleinheit, der arithmetische und logische Operationen mit binären Daten ausführt.

Die ALU muß ein **Addierwerk** besitzen, das in der Lage ist, die Inhalte zweier Register nach den Regeln der binären Arithmetik zu addieren. Dadurch kann der Prozessor arithmetische Operationen mit binären Daten durchführen, die er aus dem Speicher oder von anderen Eingängen her geliefert bekommt.

Ein Programmierer kann unter ausschließlicher Verwendung des Addierwerks Programme schreiben, die subtrahieren, multiplizieren und dividieren können. Dadurch wird das System in die Lage vesetzt, alle arithmetischen Operationen durchzuführen. In der Praxis besitzen die meisten ALUs weitere hardwaremäßig realisierte Funktionen, etwa Subtraktion, Operationen der Booleschen Logik oder Shiftoperationen.

Die ALU enthält **Merkbits**, die bestimmte Bedingungen anzeigen, die im Verlauf arithmetischer und logischer Operationen auftreten. Die Merkbits zeigen im allgemeinen das Auftreten eines **Übertrags** oder einer **Null** an, das **Vorzeichen** und die **Parität** des Resultats. Es ist möglich, Sprungbefehle zu geben, die vom Inhalt eines oder mehrerer Merkbits abhängen. So kann man zum Beispiel ein

bestimmtes Unterprogramm aufrufen, wenn im Gefolge einer Additionsoperation das Überlauf-Bit gesetzt worden ist.

Steuerlogik

Die Steuerlogik ist der wichtigste Teil einer Zentraleinheit überhaupt. Mit Hilfe von Taktsignalen steuert sie die Folge von Ereignissen, die zur Ausführung der einzelnen Befehle notwendig ist. Nachdem ein Befehl aus dem Speicher geholt und decodiert wurde, gibt die Steuerlogik die entsprechenden Signale aus (sowohl an interne als auch an externe Einheiten), die die geforderten Aktionen starten. Häufig kann die Steuerlogik auch auf Signale von außen reagieren, z.B. auf Unterbrechungssignale (INTERRUPTs) oder Wartesignale (WAIT-Signale). Ein **Unterbrechungssignal** veranlaßt die Zentraleinheit zu einer vorübergehenden Programmunterbrechung. Sie führt dann ein besonderes Unterprogramm aus, um die Forderungen des unterbrechenden Gerätes zu erfüllen. Anschließend kehrt sie automatisch zum Hauptprogramm zurück. Ein **Wartesignal** wird häufig von einem Speicher oder einem E/A-Gerät abgegeben, das langsamer als die Zentraleinheit ist. Die Steuerlogik hält dann die Zentraleinheit so lange an, bis der Speicher oder das E/A-Gerät die Daten bereit hat.

1.4. Elementaroperationen eines Computers

Es gibt einige Elementaroperationen, die bei Computern gleich sind. Ein gründliches Verständnis dieser Operationen ist die notwendige Voraussetzung, um im Einzelfall die Funktionsweise eines speziellen Computers zu verstehen.

Taktsteuerung

Die Operationen einer Zentraleinheit sind zyklisch. Der Prozessor holt einen Befehl aus dem Speicher, führt die verlangten Operationen aus, holt anschließend den nächsten Befehl usw. Eine geordnete Folge derartiger Operationen erfordert eine Taktsteuerung. Die Zentraleinheit braucht daher einen unabhängigen Taktgenerator, der den Basistakt für alle ihre Operationen liefert. Als **Befehlszyklus** bezeichnen wir das Lesen und die Ausführung eines Befehls. Die Phasen eines Befehlszyklus, in denen wohldefinierte elementare Ereignisse ablaufen, nennen wir **Zustände**, und die Zeiten zwischen den Impulsen des Taktgenerators heißen **Taktperioden**. Im allgemeinen dauert ein Zustand eine oder mehrere Taktperioden und ein Befehlszyklus umfaßt mehrere Zustände.

Befehlsabruf

Die ersten Zustände (evtl. nur einer) eines Befehlszyklus dienen dem Abruf des nächsten Befehls aus dem Speicher. Die Zentraleinheit gibt ein Lesesignal und den Inhalt des Befehlszählers an den Speicher, der daraufhin das nächste Befehlswort liefert. Das erste Byte eines Befehls wird im Befehlsregister abgelegt. Wenn der Befehl mehrere Bytes umfaßt, werden noch weitere Zustände benötigt, um diese aus dem Speicher auszulesen. Sobald die Zentraleinheit den Befehl vollständig gelesen hat, wird der Befehlszähler erhöht, um so das Auslesen des nächsten

Befehls vorzubereiten. Dann wird der Befehl decodiert. In den nachfolgenden Zuständen erfolgt die Operation, die durch den Befehl vorgeschrieben wird. Das kann eine Ein- oder Ausgabe sein, ein Lese- oder Schreibvorgang im Speicher oder eine interne Prozessor-Operation, wie z.B. eine Übertragung von Daten von einem Register zum andern oder eine Addition von Registerinhalten.

Auslesen aus dem Speicher

Ein Befehlsabruf ist nur eine spezielle Leseoperation, die einen Befehl vom Speicher in das Befehlsregister der Zentraleinheit bringt. Dieser Befehl kann dann selbst wieder veranlassen, daß Daten vom Speicher ausgelesen werden. Dazu erzeugt die Zentraleinheit ein weiteres Lesesignal und gibt die erforderliche Speicheradresse aus. Der Speicher liefert dann das angeforderte Datenwort. Dieses wird von der Zentraleinheit im Akkumulator oder in einem der Register für allgemeine Zwecke abgelegt (nicht aber im Befehlsregister).

Einschreiben in den Speicher

Eine Schreiboperation ist einer Leseoperation ähnlich, mit Ausnahme der Richtung der Datenübertragung. Die Zentraleinheit erzeugt ein Schreibsignal, gibt die Zieladresse an den Speicher und schickt dann das in den Speicher einzuschreibende Wort ab.

Wartezustand (Speichersynchronisation)

Wie schon vorher festgestellt wurde, werden die Aktionen eines Prozessors von einem Taktgenerator gesteuert. Die Taktperiode bestimmt die zeitliche Abfolge aller Einzelaktionen.

Die Länge des Befehlszyklus hängt von der **Speicherzugriffszeit** ab. Wenn der Prozessor eine Leseadresse an den Speicher abgeschickt hat, muß er so lange warten, bis der Speicher antworten kann. Viele Speicher antworten schneller als der Befehlszyklus es erfordert. Einige können die angeforderten Daten jedoch nicht in der Zeit bereitstellen, die vom Taktgenerator des Prozessors vorgeschrieben wird.

Daher sollte ein Prozessor eine Synchronisationseinrichtung besitzen, die es dem Speicher erlaubt, den Prozessor in den **Wartezustand** zu versetzen. Wenn der Speicher ein Lese- oder Schreibsignal empfängt, gibt er ein Wartesignal auf die READY-Leitung des Prozessors aus. Dadurch wird die Zentraleinheit vorübergehend angehalten. Wenn der Speicher antwortbereit ist, gibt er die READY-Leitung frei und der Befehlszyklus läuft weiter.

Ein/Ausgabe

Ein- und Ausgabeoperationen gleichen den Schreib- und Leseoperationen mit dem Unterschied, daß an Stelle einer Speicheradresse ein E/A-Gerät adressiert wird. Die Zentraleinheit erzeugt das erforderliche Ein- oder Ausgabesignal und gibt die Adresse des Ein- oder Ausgabegerätes aus. Anschließend nimmt sie die eingegebenen Daten auf oder schickt die Ausgabedaten ab.

Daten können seriell oder parallel ein- und ausgegeben werden. Alle Daten im Computer sind als Binärworte codiert. Ein Binärwort besteht aus mehreren Bits und jedes dieser Bits hat einen der Werte Ø oder 1. Bei **paralleler** Ein- oder Ausgabe werden alle Bits des Datenworts simultan übertragen, jedes Bit auf einer eigenen Leitung. Bei **serieller** Ein- oder Ausgabe wird ein Bit nach dem anderen übertragen und zwar alle auf der gleichen Leitung. Natürlich ist serielle E/A langsamer als parallele, aber es ist viel weniger Schaltungsaufwand dafür erforderlich.

Programmunterbrechungen

Zur Erhöhung ihres Wirkungsgrades besitzen viele Prozessoren eine Einrichtung, die Unterbrechungen von außen erlaubt. Man denke zum Beispiel an einen Computer, der große Datenmengen verarbeitet und diese auf einem Schnelldrucker ausgeben soll. Die Zentraleinheit kann pro Befehlszyklus ein Datenwort ausgeben, aber es kann sehr viele Befehlszyklen dauern, bis der Drucker das entsprechende Zeichen auch gedruckt hat. Die Zentraleinheit könnte dann warten bis der Drucker so weit ist, daß er das nächste Datenwort verarbeiten kann. Wenn der Prozessor jedoch Unterbrechungssignale zuläßt, kann er ein Datenwort ausgeben und anschließend mit der Verarbeitung der Daten fortfahren. Wenn der Drucker das nächste Datenwort aufnehmen kann, fordert er eine Unterbrechung. Sobald der Prozessor die Aufforderung angenommen hat, unterbricht er das laufende Programm und springt automatisch zu einem Unterprogramm, welches das nächste Datenbyte ausgibt. Anschließend setzt die Zentraleinheit die Bearbeitung des Hauptprogramms fort. Man beachte, daß dies ganz ähnlich wie ein normaler Unterprogrammaufruf ist, mit dem Unterschied, daß der Sprung von außen und nicht durch das Programm verursacht wird. Kompliziertere Unterbrechungsverfahren sind möglich, etwa in der Weise, daß mehrere Geräte Unterbrechungssignale an den gleichen Prozessor abgeben dürfen, jedoch auf verschiedenen Vorrangebenen. Unterbrechungen sind ein wichtiges Mittel zur maximalen Ausnutzung eines Prozessors.

Direkter Speicherzugriff (DMA)

Ein weiteres wichtiges Mittel zur besseren Ausnutzung eines Prozessors stellt der **direkte Speicherzugriff** (DMA) dar.

Bei gewöhnlichen E/A-Operationen überwacht der Prozessor selbst den gesamten Datenverkehr. Die Information, die im Speicher abgelegt werden soll, wird vom Eingabegerät an den Prozessor übergeben und anschließend zur vorgesehenen Speicherzelle. In ähnlicher Weise macht die Information bei der Ausgabe einen Umweg über den Prozessor.
Einige Peripheriegeräte können Informationen jedoch wesentlich schneller liefern bzw. aufnehmen als der Prozessor. Falls nun eine größere Datenmenge von einem solchen Gerät zum Speicher oder in umgekehrter Richtung übertragen werden muß, kann man die Übertragungsgeschwindigkeit dadurch steigern, daß man eine direkte Datenübertragung zwischen dem Speicher und der Peripherie zuläßt. Der Prozessor muß dann aber vorübergehend warten bis die Übertragung beendet ist, damit es keine Konflikte dadurch gibt, daß der Prozessor und das Peripheriegerät den Systembus gleichzeitig belegen wollen.

2. Mikroprozessor SAB 8080A

2.1. Allgemeines

Der SAB 8080A stellt eine vollständige Zentraleinheit (central processing unit) zum Einsatz in Digital-Computern dar. Er ist als ein LSI-Baustein, im n-Kanal-Silizium-MOS-Verfahren hergestellt. Der SAB 8080A überträgt Daten und Information über seinen Zustand auf einem Zweiweg-Tri-State-Datenbus, der acht Bit parallel übertragen kann. ($D_\emptyset \ldots D_7$). Adressen von Speicherzellen und Peripheriegeräten werden über einen eigenen Tri-State-Adreßbus ($A_\emptyset \ldots A_{15}$), der 16 Bit parallel überträgt, ausgegeben. Außerdem besitzt der SAB 8080 sechs Takt- und Steuerausgänge, (SYNC, DBIN, WAIT, $\overline{WR}$, HLDA und INTE) sowie vier Steuerleitungen (READY, HOLD, INT und RESET), vier Spannungseingänge (+12, +5, −5 und GND) und zwei Takteingänge (Φ_1 und Φ_2).

2.2. Aufbau

Die Zentraleinheit SAB 8080A besteht aus folgenden vier Funktionseinheiten

- dem Registerfeld und der Adressenlogik
- der Arithmetik-Logik-Einheit (ALU)
- dem Befehlsregister und der Steuerlogik
- dem Zweiweg-Datenbus-Puffer mit 3 Ausgangszuständen (Tri-state)

Bild 2 zeigt die Funktionseinheiten der Zentraleinheit.

Register

Der Registerblock ist ein statisches RAM-Speicherfeld, das in sechs 16-Bit-Register unterteilt ist:

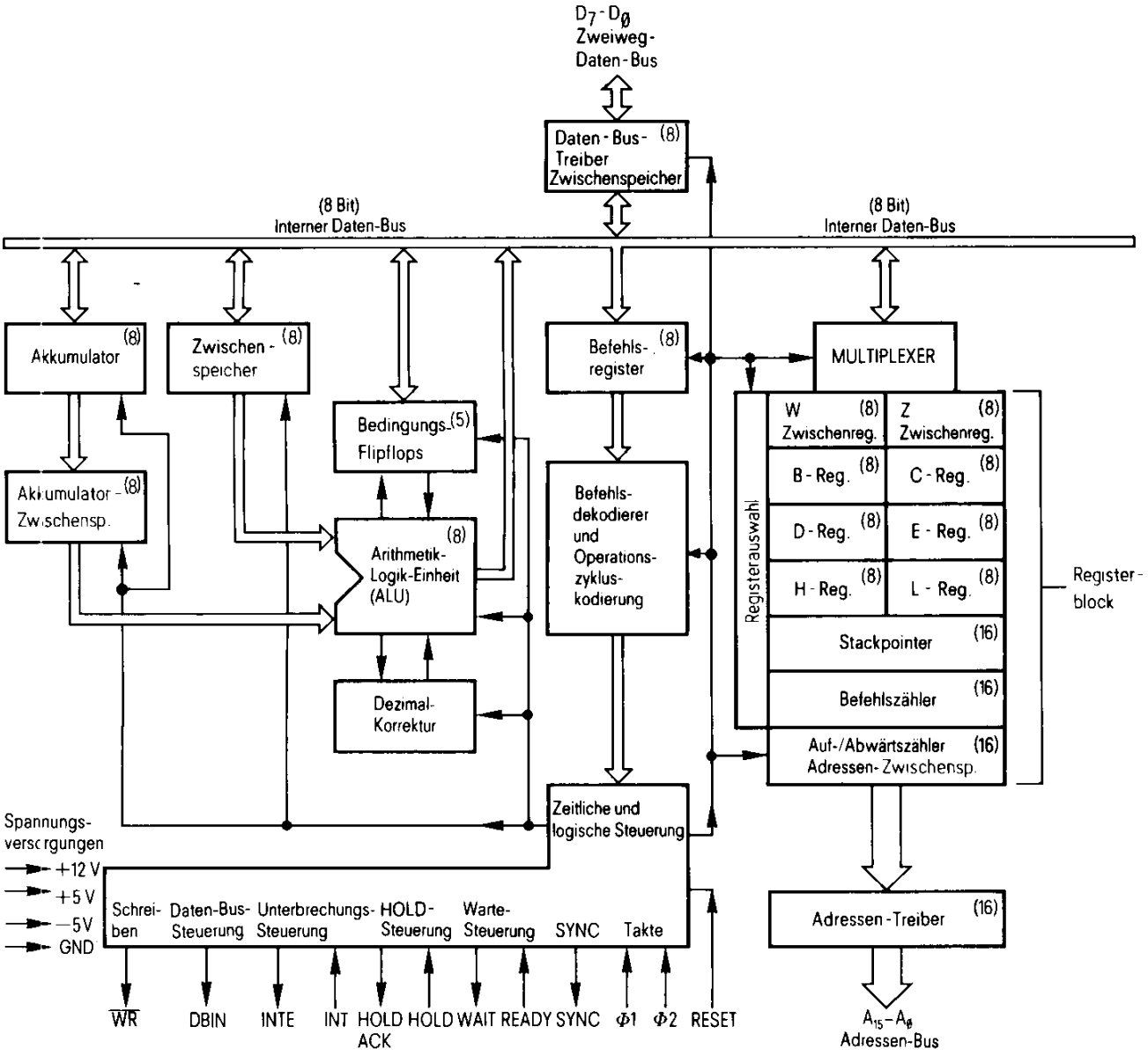
- den Befehlszähler (PC, program counter)
- den Stackpointer (SP, Kellerspeicherzeiger)
- sechs paarweise angeordnete Mehrzweckregister mit je 8 Bit mit den Bezeichnungen B, C; D, E; H, L
- ein Zwischenspeicher-Registerpaar mit der Bezeichnung W, Z

Der **Befehlszähler** enthält die Adresse des nächsten auszuführenden Befehls und wird während des Befehlsabruf-Zyklus automatisch erhöht (inkrementiert[1])).

Der Stackpointer (Kellerspeicherzeiger) enthält die Adresse des letzten belegten Platzes im Stack. Der Stackpointer kann mit einer beliebigen Adresse vorbesetzt werden und so ein frei wählbarer RAM-Speicherbereich als Stack benutzt werden. Wenn Daten im Stack abgelegt werden, wird der Stackpointer automatisch erniedrigt (PUSH-Operation). Wenn Daten vom Stack ausgelesen werden, wird der Stackpointer automatisch erhöht (POP-Operation). Der Stack wird also von höheren zu niedrigeren Adressen hin aufgefüllt.

[1]) inkrementieren = erhöhen um eins.

Bild 2
Funktions-Blockschaltbild des SAB 8080A



Die **sechs Mehrzweckregister** können sowohl als Einzelregister mit 8 Bit als auch als Registerpaare mit 16 Bit verwendet werden. Das Zwischenspeicher-Registerpaar W, Z, ist nicht vom Programm her adressierbar und wird intern bei der Befehlsausführung benutzt.

8 Bit lange Datenbytes können mit Hilfe des Registerauswahl-Multiplexer zwischen dem internen Bus und dem Registerfeld übertragen werden. Übertragungen von 16-Bit-Worten können zwischen dem Registerfeld und dem Adreßzwischenspeicher bzw. dem Auf-/Abwärtszähler stattfinden. Der Adreßzwischenspeicher erhält von einem der drei Registerpaare Daten und steuert die 16 Adressentreiber ($A_{\emptyset} \ldots A_{15}$) sowie den Auf-/Abwärtszähler an. Der Auf-/Abwärtszähler erhält Daten vom Adreßzwischenspeicher und reicht sie an das Registerfeld weiter. Die 16 Bit langen Datenworte können dabei inkrementiert oder dekrementiert oder auch unverändert weitergegeben werden.

Arithmetik-Logik-Einheit (ALU)

Zur ALU gehören folgende Register:

- einen Akkumulator mit 8 Bit
- einen Hilfsakkumulator mit 8 Bit (Akkumulator-Zwischenspeicher)
- ein Kennzeichen-Register (Bedingungs-Flipflops) mit 5 Bit für Null (zero), Übertrag (carry), Vorzeichen (sign), Parität (parity) und Hilfsübertrag (auxiliary carry)
- ein Zwischenspeicher-Register mit 8 Bit

Die ALU führt arithmetische, logische und Schiebeoperationen aus. Sie wird vom Zwischenspeicher-Register, dem Hilfsakkumulator und dem Übertrag-Flipflop mit Eingabedaten versorgt. Das Resultat der Operation kann auf den internen Bus oder in den Akkumulator übertragen werden. Außerdem wird von der ALU das Kennzeichen-Register mit Daten versorgt.

Das Zwischenspeicher-Register bekommt Information vom internen Bus und kann sie vollständig oder auszugsweise an die ALU, an das Kennzeichen-Register oder an den internen Bus weitergeben.

Der Akkumulator kann von der ALU her oder vom internen Bus Daten bekommen und sie an den Hilfsakkumulator und an den internen Bus weitergeben. Bei der Ausführung des DAA-Befehls (decimal adjust accumulator) wird auf den Inhalt des Akkumulators unter Auswertung des Übertrag- und Hilfsübertrag-Flipflops eine Dezimalkorrektur durchgeführt (Verarbeitung von BCD-Ziffern).

Befehlsregister und Steuerlogik

Während des Befehlsabrufs wird das erste Byte eines Befehls, also der Operationscode vom internen Bus in das 8 Bit Befehlsregister geladen.

Der Inhalt des Befehlsregisters wiederum, wird an den Befehlsdecodierer weitergegeben. Dessen Ausgabe liefert zusammen mit verschiedenen Taktimpulsen, die Steuersignale für den Registerblock, die ALU und die Pufferspeicher. Außerdem gehen die Ausgänge des Befehlsdecodierers und die Steuersignale von außen an die Steuerlogik, die die Zustands- und Zyklustaktsignale erzeugt.

Datenbus-Puffer

Der 8 Bit Zweiweg-Tri-State-Datenbus-Puffer wird dazu eingesetzt, den internen Mikroprozessorbus vom externen Datenbus ($D_0 \ldots D_7$) zu trennen. Im Ausgabezustand wird der Inhalt vom internen Datenbus in einen 8 Bit fassenden Zwischenspeicher geladen, der seinerseits die Datenbus-Ausgangstreiber steuert. Während der Eingabeoperationen und anderen Operationen, die keine Datenübergabe erfordern, sind die Ausgabetreiber abgeschaltet.

Im Eingabezustand werden Daten vom externen Datenbus auf den internen übertragen. Zu Beginn eines jeden internen Zustandes, mit Ausnahme des Übertragungszustandes, wird der interne Bus vorbesetzt. (Dies ist der Zustand T3, der später in diesem Kapitel noch beschrieben wird.)

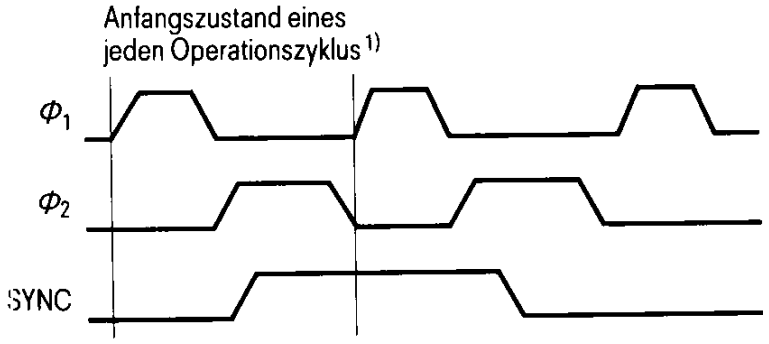
2.3. Arbeitszyklen

Ein **Befehlszyklus** ist die Zeit, die zum Abruf und zur Ausführung eines Befehls benötigt wird. Beim Befehlsabruf wird ein Befehl mit einem, zwei oder drei Bytes aus dem Speicher ausgelesen und im Befehlsregister des Mikroprozessors abgelegt. Bei der Ausführung des Befehls wird der Befehl decodiert und in bestimmte Aktionen umgesetzt.

Jeder Befehlszyklus besteht aus einem, zwei, drei, vier oder fünf **Operationszyklen**. Für jeden Speicherzugriff und jeden Zugriff auf einen E/A-Kanal wird ein Operationszyklus benötigt. Der Befehlsabruf benötigt für jedes Byte des Befehls einen Operationszyklus. Die Dauer der Befehlsausführung hängt von der speziellen Art des Befehls ab. Manche Befehle benötigen außer den Zyklen zum Befehlsabruf keine weiteren. Andere Befehle benötigen zusätzliche Operationszyklen für Speicherzugriffe oder E/A-Operationen. Der DAD-Befehl (double add) benötigt ausnahmsweise zwei zusätzliche Operationszyklen zur Addition eines Registerpaares.

Jeder **Operationszyklus** besteht aus drei, vier oder fünf Zuständen. Ein **Zustand** ist die kleinste Aktionseinheit und ist definiert als der Zeitraum zwischen zwei ansteigenden Flanken des Φ_1-Taktes. Der SAB 8080A wird von einem zweiphasigen Taktgenerator gesteuert. Alle seine Aktivitäten sind auf diesen Takt bezogen. Die beiden Taktimpulse, die sich nicht überlappen und mit Φ_1 und Φ_2 bezeichnet werden, kommen von außen in den Prozessor. Der Φ_1-Takt dient dazu, den Operationszyklus in Zustände zu unterteilen. Die Logik für die Ablaufsteuerung im SAB 8080A erzeugt mit Hilfe der eingegebenen Taktsignale einen Synchronisationsimpuls (SYNC), der den Anfang eines jeden Operationszyklus markiert. Der SYNC-Impuls wird durch die ansteigende Flanke des Φ_2-Taktes ausgelöst, wie dies in Bild 3 gezeigt wird.

Bild 3
Impulsdiagramm der Signale Φ_1, Φ_2 und SYNC



Durch Φ_1 ist die Dauer der Zustände festgelegt bis auf drei Ausnahmen, und zwar den Wartezustand (WAIT-Zustand) und die Haltezustände HLDA und HLTA, die später beschrieben werden. Da die Dauer dieser Zustände von äußeren Signalen abhängt, haben sie von vornherein keine feste Länge. Aber auch diese Sonderzustände müssen mit den Taktimpulsen synchronisiert werden. Daher ist die Zeitdauer eines beliebigen Zustandes ein ganzzahliges Vielfaches der Dauer eines Taktes.

[1]) Das SYNC-Signal kommt im zweiten und dritten Operationszyklus des DAD-Befehls nicht vor, da diese nur für eine interne Addition eines Registerpaares benötigt werden.

Zusammenfassend halten wir fest:

Jede **Taktperiode** definiert einen **Zustand**,
drei bis fünf **Zustände** ergeben zusammen einen **Operationszyklus** und
ein bis fünf **Operationszyklen** ergeben zusammen einen **Befehlszyklus**.

Beschreibung der Operationszyklen

Mit Ausnahme des DAD-Befehls hängt es nur von einer Bedingung ab, wie viele Operationszyklen für einen Befehlszyklus erforderlich sind: nämlich von der Anzahl der Zugriffe auf den Speicher oder ein peripheres Gerät, die für den Abruf und die Ausführung des Befehls nötig sind. Wie viele andere Prozessoren ist auch der SAB 8080A so konzipiert, daß in jedem Operationszyklus nur eine Adresse übertragen werden kann. Wenn also für den Abruf und die Ausführung eines Befehls zwei Speicherzugriffe nötig sind, umfaßt der Befehlszyklus dieses Befehls zwei Operationszyklen. Wenn fünf Speicherzugriffe nötig sind, dann sind für den zugehörigen Befehlszyklus fünf Operationszyklen nötig.

In jedem Befehlszyklus kommt mindestens ein Speicherzugriff vor, durch den der Befehl geholt wird. Ein solcher Befehlsabruf ist auch dann notwendig, wenn der Befehl keine weiteren Speicherzugriffe mehr verlangt. Daher ist der erste Operationszyklus eines Befehls ein Abruf (FETCH). Darüber hinaus kann man keine einfachen Regeln für den weiteren Ablauf angeben, da dieser vom Befehlstyp abhängt.

Wir betrachten einige Beispiele. Der Befehl zur Addition von Registerinhalten (ADD r) benötigt nur einen Operationszyklus und zwar einen Befehlsabruf. In diesem Befehl, der nur ein Byte lang ist, wird der Inhalt des Akkumulators zum Inhalt eines der Mehrzweckregister addiert. Da die Information, die zur Ausführung benötigt wird, in den acht Bit des Operationscodes steckt, ist nur ein Speicherzugriff erforderlich, der Befehlsabruf. Drei Zustände sind notwendig, um den Befehl aus dem Speicher abzurufen und in einem weiteren Zustand wird die verlangte Addition ausgeführt. Für den Befehl ist also nur ein Operationszyklus notwendig, der aus vier Zuständen oder vier Taktperioden besteht.

Wir nehmen nun an, daß der Inhalt einer Speicherzelle auf den vorgegebenen Akkumulatorinhalt addiert werden soll (ADD M-Befehl). Obwohl dieses Beispiel dem eben erläuterten ganz ähnlich ist, werden einige zusätzliche Schritte benötigt. Ein zusätzlicher Operationszyklus wird gebraucht, um den Inhalt der Speicherzelle zu lesen.

Der genaue Ablauf sieht folgendermaßen aus: Zunächst wird das ein Byte umfassende Befehlswort mit Hilfe der im Befehlszähler gespeicherten Adresse vom Speicher abgerufen. Dazu sind drei Zustände notwendig. Das acht Bit umfassende Wort, das beim Befehlsabruf vom Speicher geholt worden ist, wird im Befehlsregister des Prozessors abgelegt und dazu benutzt, den weiteren Ablauf des Befehlszyklus zu steuern. Anschließend wird der Inhalt der Register H und L auf den Adreßleitungen ausgegeben. Das Datenwort von 8 Bit Länge, das bei diesem **Lesezyklus** (MEMORY READ) an den Prozessor vom Speicher geliefert wird, wird in einem Zwischenspeicher-Register im Prozessor abgelegt. Dies dauert drei weitere Taktperioden (Zustände). Im siebten und letzten Zustand wird der Inhalt des Zwischenspeicher-Registers auf den Inhalt des Akkumulators addiert. Folglich benötigt der ADD M-Befehl zwei Operationszyklen mit insgesamt sieben Zuständen.

Das andere Extrem stellt der SHLD-Befehl dar, der den Inhalt der Register H und L im Speicher ablegt und dafür fünf Operationszyklen benötigt. Wahrend eines SHLD-Befehlszyklus werden die Inhalte der Register H und L in zwei aufeinanderfolgenden Speicherplätzen abgelegt. Die Zieladresse steht in den beiden Speicherplätzen, die unmittelbar auf den Operationscode folgen. Die nachstehenden Ereignisse laufen nacheinander ab:

1. **Ein Lesezyklus** (FETCH), der aus vier Zuständen besteht. In den ersten drei Zuständen ruft der Prozessor mit Hilfe der Adresse im Befehlszähler den Operationscode vom Speicher ab. Anschließend wird der Befehlszähler erhöht. Der vierte Zustand wird für die interne Befehlsdecodierung benötigt.
2. **Ein Speicher-Lesezugriff** (MEMORY READ), der drei Zustände umfaßt. In diesem Operationszyklus wird das Byte, das durch die Adresse im Befehlszähler adressiert ist, vom Speicher ausgelesen und im Z-Register abgelegt. Anschließend wird der Befehlszähler noch einmal erhöht.
3. **Ein weiterer Speicher-Lesezugriff** mit drei Zuständen, der das durch den Befehlszähler angegebene Byte vom Speicher in das W-Register bringt. Dann wird der Befehlszähler wieder erhöht, um den nächsten Befehlsabruf vorzubereiten.
4. **Ein Speicher-Schreibzugriff** (MEMORY WRITE) mit drei Zuständen, in dem der Inhalt des L-Registers auf den Speicherplatz gebracht wird, der durch den momentanen Inhalt des Registerpaares W und Z bezeichnet ist. Der Zustand, der auf die Datenübergabe folgt, wird dazu benutzt, die Adresse in W und Z zu inkrementieren, um so jenen Speicherplatz zu definieren, der Daten aufnehmen soll.
5. **Ein Speicher-Schreibzugriff** mit drei Zuständen, durch den der Inhalt von H in die Speicherzelle eingeschrieben wird, die durch W und Z adressiert ist.

Somit umfaßt der SHLD-Befehl fünf Operationszyklen mit insgesamt 16 Zuständen.

Die meisten Befehle liegen in der Zahl ihrer Zustände zwischen den beiden Extremen, dem ADD r-Befehl und dem SHDL-Befehl. So benötigen der Eingabebefehl (IN) und der Ausgabebefehl (OUT) drei Operationszyklen:

einen Befehlsabrufzyklus, um den Befehl aus dem Speicher zu holen,
einen Speicher-Lesezugriff, um die Adresse des Peripheriegerätes zu holen
einen Ein- oder Ausgabezyklus, um die Datenübertragung auszuführen.

Es gibt in den Befehlszyklen insgesamt neun verschiedene Typen von Operationszyklen. (Ein einzelner Operationszyklus umfaßt aber nie mehr als fünf Zustände):

a) Befehlsabruf (FETCH)
b) Speicher-Lesezugriff (MEMORY READ)
c) Speicher-Schreibzugriff (MEMORY WRITE)
d) Auslesen aus dem Stack (STACK READ)
e) Einschreiben in den Stack (STACK WRITE)
f) Eingabe (INPUT)
g) Ausgabe (OUTPUT)
h) Unterbrechung (INTERRUPT)
i) Haltzustand (HALT)

Welche Operationszyklen in einem bestimmten Befehlszyklus vorkommen, hängt von der Art des Befehls ab. Generell ist es aber so, daß der erste Operationszyklus immer ein Befehlsabruf ist (FETCH).

Der Prozessor gibt an, welcher Operationszyklus gerade abläuft, indem er im ersten Zustand eines jeden Operationszyklus ein Zustandssignal von acht Bit Länge ausgibt. Die Information über den Zustand wird während des SYNC-Impulses auf den SAB 8080A Datenleitungen ausgegeben ($D_\emptyset \ldots D_7$). Diese Daten werden in Zwischenspeichern aufbewahrt, decodiert und zur Erzeugung von Steuersignalen für die externe Logik verwendet. Tabelle 2-1 zeigt, wie die Zustandsinformation auf den Datenleitungen des Prozessors ausgegeben wird. Dabei entsprechen positive Signale einer erfüllten Bedingung.

Die Zustandsinformation dient vor allem zur Steuerung der externen Logik. Die Festlegung der einzelnen Zustandsbits wird mehr durch Zweckmäßigkeitsüberlegungen als durch den Typ des Operationszyklus bestimmt. Man sieht daher, daß manche Operationszyklen bereits durch ein einziges Zustandsbit eindeutig gekennzeichnet sind. Das Zustandsbit für M_1, das auf D_5 liegt, charakterisiert bereits eindeutig

Tabelle 2-1
Festlegung der Zustandsbits des SAB 8080A

Symbole	Datenbus-Leitung	Beschreibung
HLTA	D_3	Quittungssignal für den HALT-Befehl
INTA	$D_\emptyset$	Quittungssignal für eine Unterbrechungsanforderung. Das Signal sollte verwendet werden, um einen Wiederstartbefehl (restart) auf den Datenbus zu legen, wenn das DBIN-Signal aktiv ist.
INP	D_6	Zeigt an, daß auf dem Adreßbus die Adresse eines Eingabegerätes ansteht, und daß die Eingabedaten auf den Datenbus gelegt werden sollten, wenn DBIN aktiv ist.
OUT	D_4	Zeigt an, daß auf dem Adreßbus die Adresse eines Ausgabegerätes ansteht und daß der Datenbus die Ausgabedaten enthält, wenn $\overline{WR}$ aktiv ist.
MEMR	D_7	Gibt an, daß der Datenbus aus dem Speicher ausgelesene Daten enthält.
M_1	D_5	Gibt an, daß der Prozessor das erste Byte eines Befehls abruft.
STACK	D_2	Gibt an, daß auf dem Adreßbus der Inhalt des Stackpointers ansteht.
$\overline{WO}$	D_1	Gibt an, daß der momentane Operationszyklus ein Einschreiben in den Speicher oder eine Ausgabe ist ($\overline{WO}=\emptyset$). Andernfalls handelt es sich um Auslesen aus dem Speicher oder um eine Eingabe.

den Befehlsabrufzyklus. Ein Lesevorgang im Stack andererseits wird durch das gleichzeitige Vorkommen der Signale STACK und MEMR angegeben. Die Kenntnis der Zustandssignale kann auch bei der Systementwicklung und für die Fehlersuche nützlich sein. Tabelle 2-2 gibt die Zustandsbits für jeden Operationszyklus an.

Tabelle 2-2
Decodierung der Zustandsbits

Operationszyklus	Zustandsbits							
	D_0 INTA	D_1 $\overline{WO}$	D_2 STACK	D_3 HLTA	D_4 OUT	D_5 M1	D_6 INP	D_7 MEMR
Befehlsabruf	Ø	1	Ø	Ø	Ø	1	Ø	1
Auslesen vom Speicher	Ø	1	Ø	Ø	Ø	Ø	Ø	1
Einschreiben in den Speicher	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Auslesen vom Stack	Ø	1	1	Ø	Ø	Ø	Ø	1
Einschreiben in den Stack	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø
Eingabe	Ø	1	Ø	Ø	Ø	Ø	1	Ø
Ausgabe	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø
Unterbrechung	1	1	Ø	Ø	Ø	1	Ø	Ø
Halt	Ø	1	Ø	1	Ø	Ø	Ø	Ø

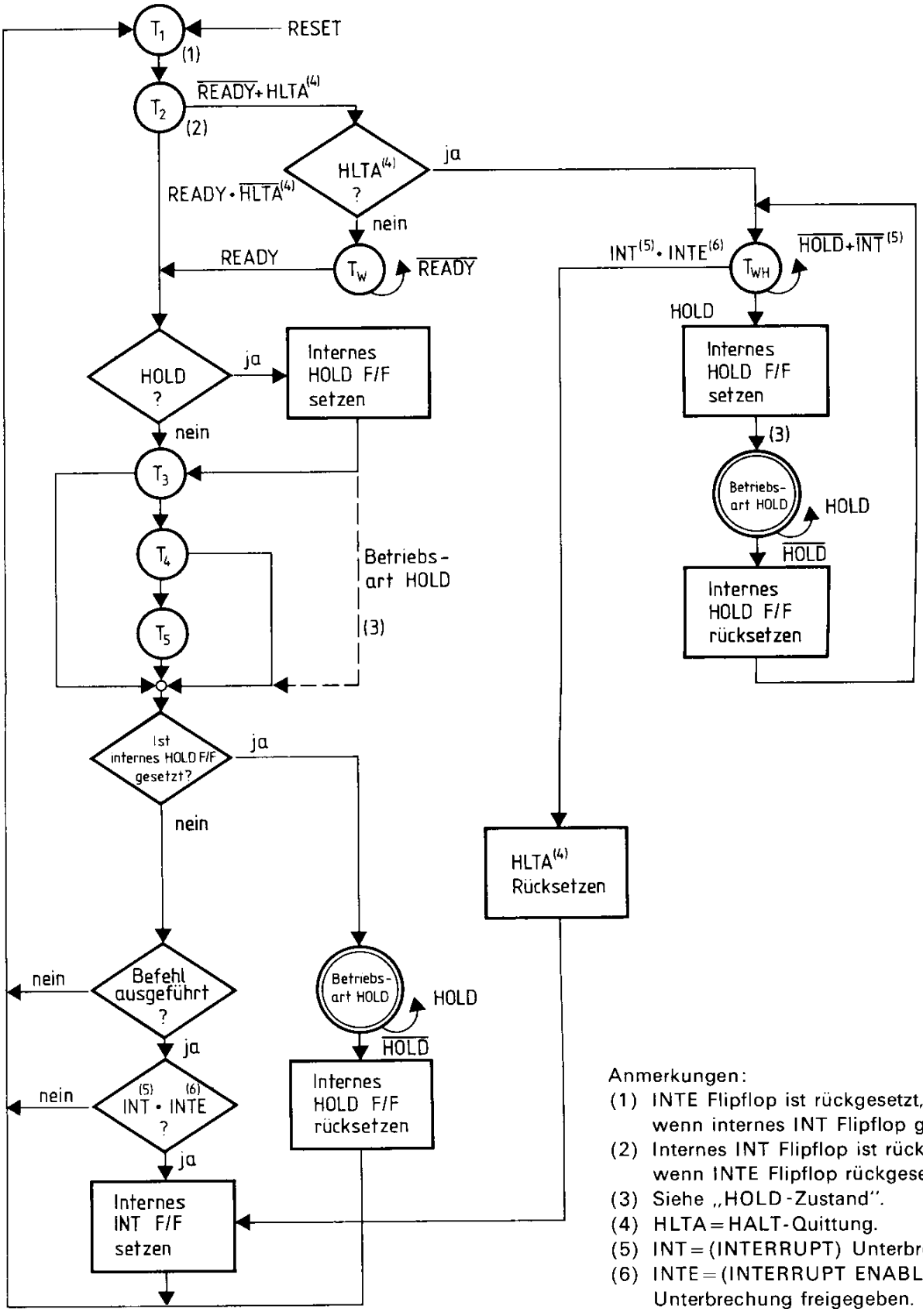
Bemerkung: 1 = H-Pegel, Ø = L-Pegel
Man beachte, daß für das Bit $\overline{WO}$ (Schreiben und Ausgabe) negative Logik gilt (L-Pegel entspricht aktiv).

Zustandsübergänge

Jeder Operationszyklus innerhalb eines Befehlszyklus besteht aus drei bis fünf Zuständen, die mit T_1, T_2, T_3, T_4, T_5, oder T_W bezeichnet werden. Die jeweilige Anzahl von Zuständen hängt vom Befehlstyp ab und des jeweiligen Operationszyklus. Das Diagramm der Zustandsübergänge in Bild 4 zeigt, wie der SAB 8080A im Verlauf eines Operationszyklus von einem Zustand zum nächsten übergeht. Das Diagramm zeigt auch, wie die Signale auf den READY-, HOLD- und INTERRUPT-Eingängen, während des Operationszyklus abgefragt werden, und wie sie die Reihenfolge der einzelnen Zustände beeinflussen. Im Augenblick befassen wir uns jedoch nur mit der prinzipiellen Reihenfolge und mit der Wirkung des READY-Signals. Die Auswirkungen der HOLD- und INTERRUPT-Signale werden später besprochen.

Der Mikroprozessor SAB 8080A zeigt seinen Zustand nicht unmittelbar durch die Ausgabe von Zustandssignalen in jedem einzelnen Zustand an. Er gibt stattdessen direkte Steuersignale nach außen ab (INTE, HLDA, DBIN, $\overline{WR}$ und WAIT), die von der peripheren Logik weiterverarbeitet werden.

Bild 4
Flußdiagramm der Operationsschritte des SAB 8080A



Anmerkungen:
(1) INTE Flipflop ist rückgesetzt, wenn internes INT Flipflop gesetzt ist.
(2) Internes INT Flipflop ist rückgesetzt, wenn INTE Flipflop rückgesetzt ist.
(3) Siehe „HOLD-Zustand".
(4) HLTA = HALT-Quittung.
(5) INT = (INTERRUPT) Unterbrechung.
(6) INTE = (INTERRUPT ENABLE) Unterbrechung freigegeben.

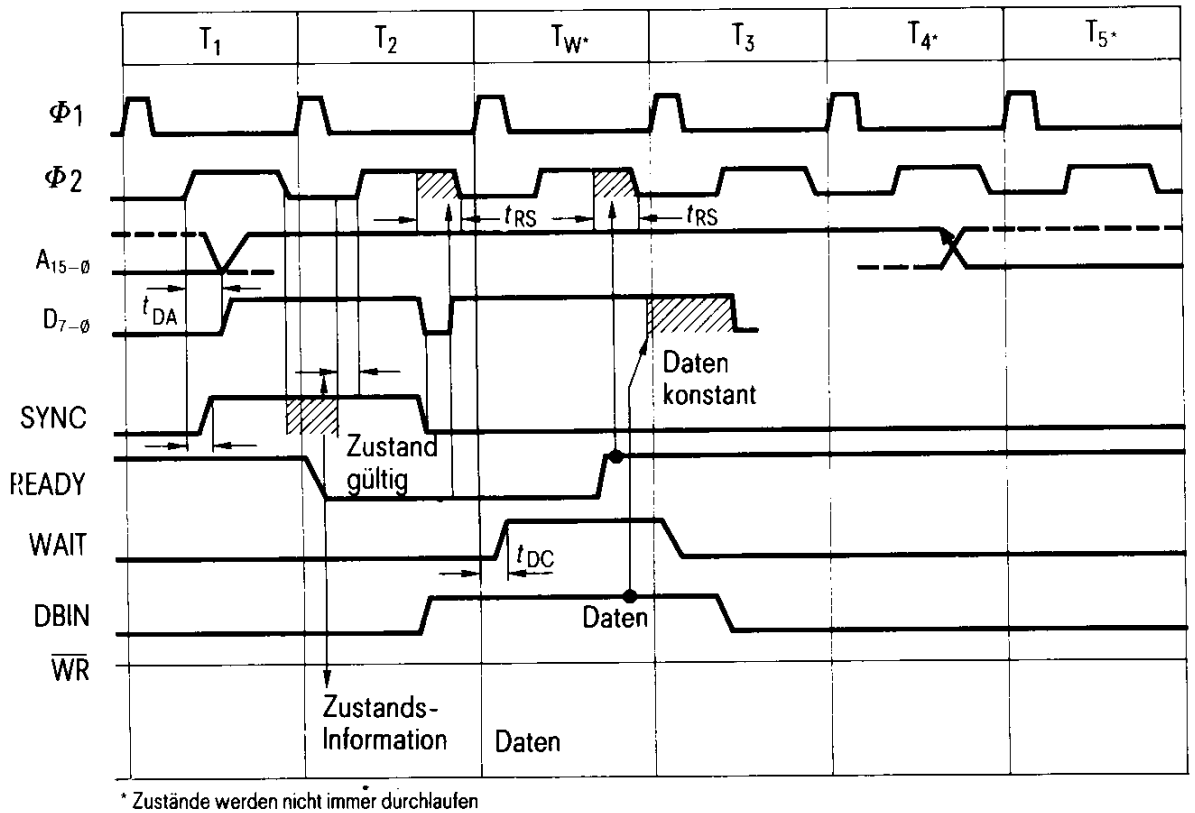
Wir erinnern noch einmal daran, daß der SAB 8080A in jedem Operationszyklus mindestens drei Zustände durchläuft. Jeder dieser Zustände wird durch eine ansteigende Flanke des Φ_1-Taktes eingeleitet. Bild 5 zeigt den Verlauf eines typischen Abrufzyklus. Ereignisse, die in jedem Zustand auftreten, werden durch Impulsflanken des Φ_1- und des Φ_2-Taktes ausgelöst.

Der erste Zustand (T_1) eines jeden Operationszyklus wird durch das SYNC-Signal gekennzeichnet. Wie in Bild 5 gezeigt wird, ist das Signal an die ansteigende Flanke des Φ_2-Taktes gebunden. Während der Dauer des SYNC-Signals wird die Zustandsinformation auf den Anschlüssen $D_\emptyset \ldots D_7$ ausgegeben, gesteuert durch den Φ_2-Takt.

Die ansteigende Flanke des Φ_2-Taktes bewirkt im Zustand T_1 auch, daß die Adreßleitungen $A_\emptyset \ldots A_{15}$ besetzt werden. Mit kurzer Verzögerung (t_{DA}) werden die Signale auf $A_\emptyset \ldots A_{15}$ stabil und anschließend bleiben sie stabil, bis zum ersten Φ_2-Impuls nach Zustand T_3. Dadurch erhält der Prozessor genügend Zeit, die Daten, die vom Speicher zurückkommen, zu lesen.

Hat der Prozessor eine Adresse an den Speicher einmal abgeschickt, kann der Speicher einen Wartezustand (WAIT) anfordern. Das geschieht dadurch, daß er die READY-Leitung des Prozessors auf L-Pegel legt, bevor das Intervall t_{RS} (Neubesetzung der READY-Leitung) beginnt. Dies beginnt während des Φ_2-Impulses im Zustand T_2 bzw. T_W. Solange die READY-Leitung L-Pegel führt, wartet der Prozessor und der Speicher hat Zeit, die Datenanforderung zu erfüllen (siehe Bild 5).

Bild 5
Typischer Befehlsabrufzyklus



* Zustände werden nicht immer durchlaufen

Auf eine Warteanforderung antwortet der Prozessor dadurch, daß er nach dem Zustand T_2 in einen Wartezustand (T_W) statt direkt in T_3 übergeht. Den Beginn von T_W zeigt der Prozessor durch ein WAIT-Signal an und quittiert damit die Anforderung des Speichers. Ein Übergang zu H-Pegel auf der WAIT-Leitung wird durch die ansteigende Flanke von Φ_1 ausgelöst und erfolgt mit kurzer Verzögerung (t_{DC}) auf den tatsächlichen Beginn des T_W-Zustandes.

Eine Warteperiode hat keine feste Dauer. Der Prozessor wartet so lange bis seine READY-Leitung wieder H-Pegel führt. Dieser Pegelwechsel muß der abfallenden Flanke des Φ_2-Taktes um eine bestimmte Zeit (t_{RS}) vorausgehen, damit der T_W-Zustand beendet werden kann. Anschließend kann der Operationszyklus weiter ablaufen, beginnend mit der ansteigenden Flanke des Φ_1-Taktes. Eine Warteperiode dauert daher ein ganzzahliges Vielfaches des T_W-Zustandes und immer ein Vielfaches des Taktintervalls von Φ_1.

Die Ereignisse, die im T_3-Zustand stattfinden, hängen vom gerade ablaufenden Operationszyklus ab. In einem Befehlsabrufzyklus interpretiert der Prozessor die Daten, die auf dem Datenbus anstehen, als Befehlswort. Während eines Speicher-Lesezugriffs oder beim Auslesen aus dem Stack, werden die Signale auf dem gleichen Bus als Daten interpretiert. Beim Einschreiben in den Speicher gibt der Prozessor im Zustand T_3 Daten auf den Bus aus. Bei E/A-Operationen kann der Prozessor entweder Daten ausgeben oder aufnehmen, je nachdem ob es sich um eine Eingabe- oder Ausgabeoperation handelt.

Bild 6
Eingabe-Befehlszyklus

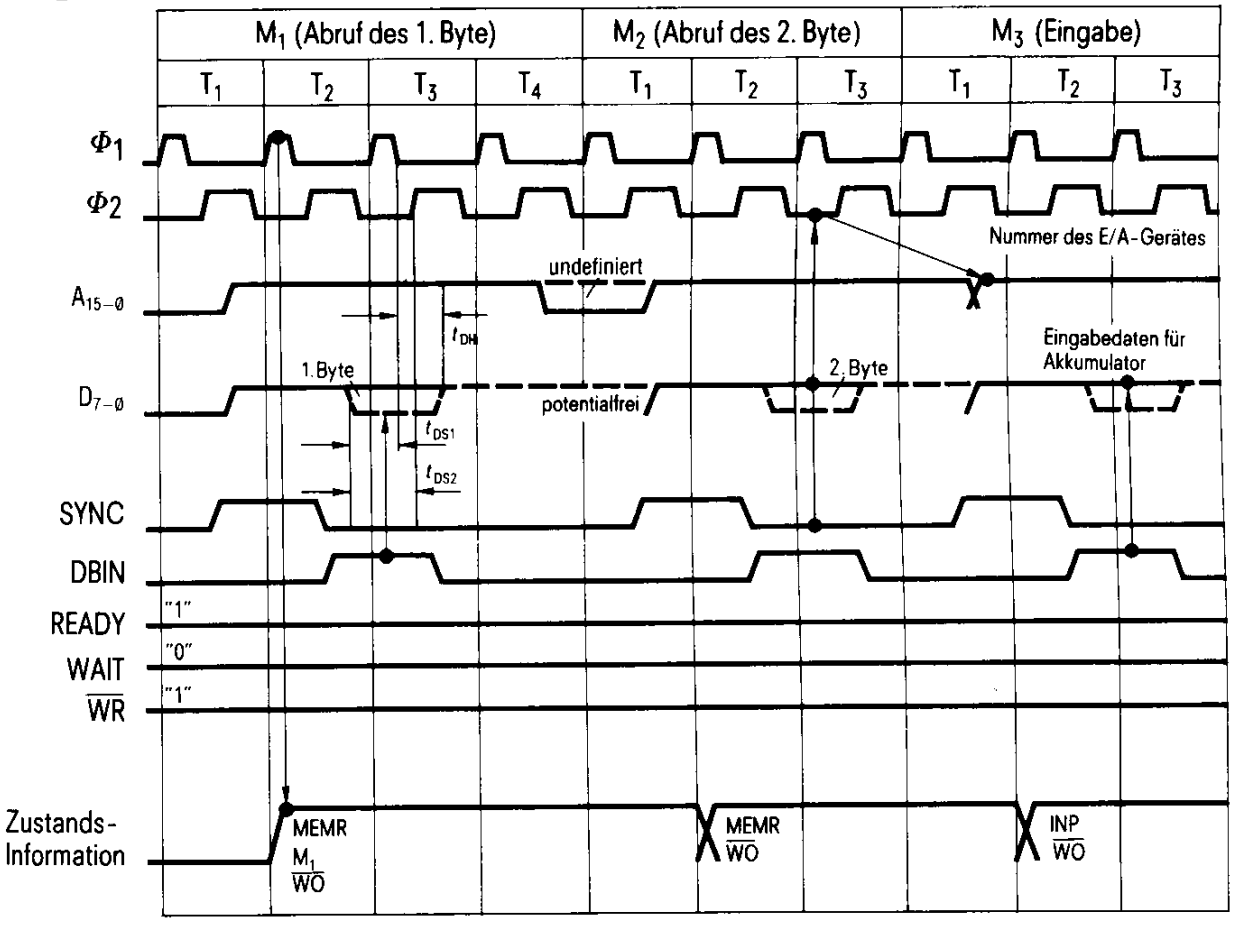


Bild 6 verdeutlicht den Ablauf einer typischen Dateneingabeoperation. Wie daraus ersichtlich ist, löscht der Übergang des Φ_2-Taktimpulses vom L- zum H-Pegel im Zustand T_2 die Zustandsinformation von den Datenleitungen und bereitet sie so für die Aufnahme der ankommenden Daten vor. Diese Daten müssen stabil geworden sein, bevor die folgenden beiden Ereignisse eintreten:

- das „Φ_1-Datenvorbereitungsintervall" t_{DS1}, das der abfallenden Flanke des Φ_1-Taktimpulses vorausgeht, der den Zustand T_3 einleitet.
- das „Φ_2-Datenvorbereitungsintervall" t_{DS2}, das der ansteigenden Flanke des Φ_2-Taktes in T_3 vorausgeht.

Die Daten müssen während des „Datenübernahme-Intervalls" t_{DH} stabil bleiben, der der ansteigenden Flanke des nächsten Φ_2-Taktimpulses vorausgeht. Die Daten, die der Speicher oder eine externe Einheit auf die Datenleitungen ausgegeben hat, werden dann im Zustand T_3 übernommen.

Wenn Daten in den SAB 8080A eingegeben werden, erzeugt der Prozessor ein DBIN-Signal, das von der peripheren Logik dazu benutzt werden kann, die Datenübertragung auszulösen. Zu den Operationszyklen, in denen das DBIN-Signal ausgegeben wird, gehören:

der Befehlsabrufzyklus,

der lesende Speicherzugriff,

das Auslesen vom Stack

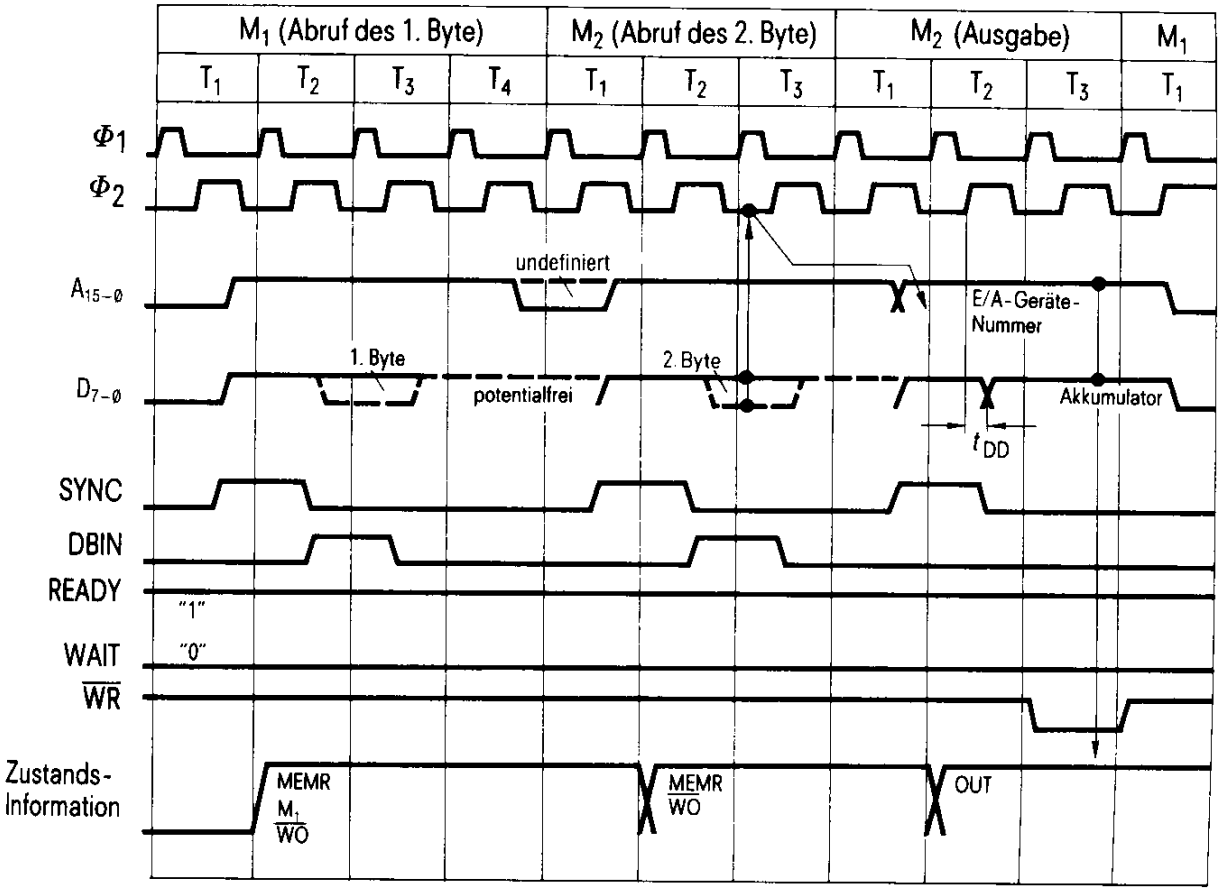
die Programmunterbrechung.

DBIN beginnt mit der ansteigenden Flanke des Φ_2-Taktes im Zustand T_2 und endet mit der ansteigenden Flanke von Φ_2 im Zustand T_3. Wenn zwischen T_2 und T_3 ein Wartezustand T_W vorkommt, wird DBIN um eine oder mehrere Taktperioden verlängert.

Bild 7 zeigt den zeitlichen Verlauf von Operationszyklen, in denen der Prozessor Daten ausgibt. Die Ausgabedaten sind für ein Peripheriegerät bestimmt. Die ansteigende Flanke von Φ_2 im Zustand T_2 löscht die Zustandsinformation von den Datenleitungen und lädt die Ausgabedaten darauf. Diese Ersetzung findet nach der „Datenausgabeverzögerung" t_{DD} statt, die auf die ansteigende Flanke des Φ_2-Taktes folgt. Die Daten auf dem Bus bleiben dann für den Rest des Operationszyklus stabil, bis sie durch die neue Zustandsinformation im nächsten Zustand (T_1) ersetzt werden. Man beachte, daß das READY-Signal für den Abschluß eines Ausgabezyklus notwendig ist. Bis es geliefert wird, bleibt der Prozessor im Anschluß an den Zustand T_2 im Wartezustand T_W. In der Zwischenzeit bleiben die Daten stabil und der Operationszyklus wird erst fortgesetzt, wenn die READY-Leitung wieder H-Pegel führt.

Zur Synchronisation externer Übertragungen erzeugt der SAB 8080 ein $\overline{WR}$-Signal in den Operationszyklen, in denen der Prozessor Daten ausgibt. Dazu gehören der Speicher-Schreibzugriff, das Einschreiben in den Stack und der Ausgabezyklus. Die abfallende Flanke des WR-Impulses ist an die ansteigende Flanke des ersten Φ_1-Taktimpulses nach T_2 gekoppelt und folgt dieser mit einer kurzen Verzögerung t_{DC}. Das $\overline{WR}$-Signal bleibt auf L-Pegel, bis es durch die ansteigende Flanke des Φ_2-Taktes im Zustand, der auf T_3 folgt, wieder hochgesetzt wird. Man

Bild 7
Ausgabe-Befehlszyklus



beachte, daß T_W-Zustände während der Eingabezyklen den gleichen Einfluß auf $\overline{WR}$ haben wie auf DBIN.

Alle Operationszyklen des Prozessors bestehen aus mindestens drei Zuständen T_1, T_2 und T_3, wie es vorher schon erwähnt wurde. Wenn der Prozessor auf ein READY-Signal warten muß, kann ein Operationszyklus auch einen oder mehrere T_W-Zustände enthalten. Während dieser drei grundlegenden Zustände werden Daten vom Prozessor nach außen abgegeben oder von ihm aufgenommen. Nach dem Zustand T_3 kann man nur schwer allgemeine Aussagen machen. Wenn es ein bestimmter Befehl erfordert, können auch die Zustände T_4 und T_5 vorkommen, aber nicht alle Operationszyklen enthalten diese Zustände. Ob sie vorkommen hängt vom Befehlszyklus und vom Operationszyklus darin ab. Der Prozessor beendet einen Operationszyklus, sobald er seine Aufgabe ausgeführt hat und durchläuft nicht jedesmal die Zustände T_4 und T_5. Daher kann ein Operationszyklus mit dem Zustand T_3, T_4 oder T_5 abschließen. Der nächste Zustand ist dann wieder der Zustand T_1 im nachfolgenden Operationszyklus.

Tabelle 2-3 gibt die Ereignisse an, die in den einzelnen Zuständen stattfinden.

Tabelle 2-3
Beschreibung der Zustände

Zustand	Beschreibung
T_1	Eine Speicheradresse oder die Adresse eines E/A-Gerätes wird auf dem Adreßbus ($A_\emptyset \ldots A_{15}$) ausgegeben. Die Zustandsinformation wird auf dem Datenbus ausgegeben.
T_2	Der SAB 8080A prüft die READY- und HOLD-Eingänge und ob ein Haftbefehl ausgeführt wurde.
T_W (nur bei Bedarf)	Der Prozessor wird in den Wartezustand versetzt entweder wenn die READY-Leitung L-Pegel führt oder durch die Ausführung eines Haltbefehls.
T_3	Der SAB 8080A erhält über den Datenbus ein Befehlsbyte (bei Abrufzyklen), ein Datenbyte (bei Lese- und Eingabezyklen) oder einen Unterbrechungsbefehl (bei Unterbrechungszyklen). Es kann aber auch ein Datenbyte auf dem Datenbus ausgegeben werden (bei Schreib- und Ausgabezyklen).
T_4 T_5 (nur bei Bedarf)	Die Zustände T_4 und T_5 stehen für besondere Befehlszyklen Verfügung. Falls sie nicht benötigt werden, können einer oder beide Zustände übersprungen werden. T_4 und T_5 werden nur für interne SAB 8080A-Operationen benutzt.

2.4. Ablauf von Unterbrechungszyklen

Der SAB 8080A kann auf Unterbrechungsanforderungen von außen reagieren. Ein Peripheriegerät kann eine Unterbrechung dadurch auslösen, daß es die Unterbrechungsleitung (INT) des Prozessors auf H-Pegel legt.

Ein Unterbrechungssignal kann ohne Synchronisierung gegeben werden. Es kann daher jederzeit und in jedem Operationszyklus vorkommen. Die interne Logik übernimmt die Synchronisation der Unterbrechungsanforderung mit dem Prozessortakt. Wie Bild 8 zeigt, bewirkt eine ankommende Unterbrechungsanforderung, während die Leitung für Unterbrechungsfreigabe (interrupt enable=INTE) H-Pegel führt, zusammen mit dem Φ_2-Takt eine Besetzung des internen Unterbrechungssignalspeichers. Dieses Ereignis erfolgt im letzten Zustand des Befehlszyklus, in dem das Unterbrechungssignal aufgetreten ist. Es ist also sichergestellt, daß jeder Befehl vollständig abgearbeitet wird, bevor die Unterbrechungsanforderung befolgt wird.

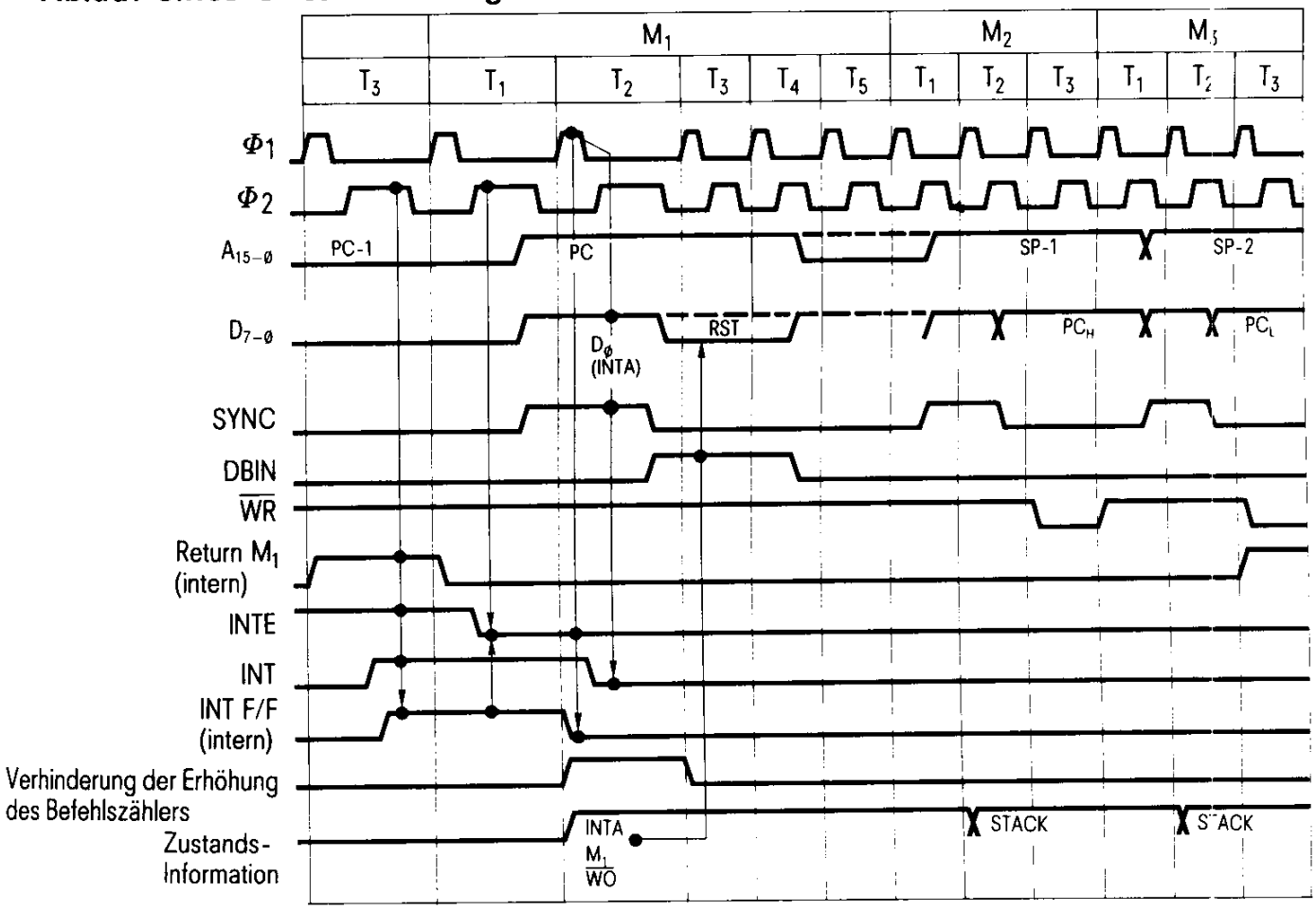
Der Unterbrechungszyklus, der auf eine freigegebene Unterbrechungsanforderung folgt, gleicht einem normalen Befehlsabrufzyklus in fast jeder Hinsicht. Das Zustandssignal M_1 wird wie gewöhnlich im SYNC-Intervall ausgegeben. Es wird jedoch außerdem durch ein INTA-Signal ($D_\emptyset$) ergänzt, das die Unterbrechungsanforderung quittiert. Im Intervall T_1 wird der Inhalt des Befehlszählers auf die Adreßleitungen gelegt, aber der Zähler wird während des Unterbrechungszyklus nicht erhöht, wie dies bei anderen Befehlen der Fall wäre. Auf diese Weise wird der Zustand des Prozessors gesichert, so wie er vor der Unterbrechungsanforderung war. Die Adresse

im Befehlszähler kann dann in den Stack übernommen werden. Das ermöglicht eine Rückkehr in das normale Programm, nachdem die Unterbrechungsanforderung bearbeitet worden ist.

Der Unterbrechungszyklus ist gewöhnlich von einem normalen Operationszyk us, der einen Befehl abruft, nicht zu unterscheiden. Der Prozessor selbst führt keine weiteren Aktionen aus. Es ist die Aufgabe der peripheren Logik dafür zu sorgen, daß ein acht Bit langer Unterbrechungsbefehl im Zustand T_3 auf die Datenleitungen des Prozessors gelegt wird. In einem typischen System muß dazu der Dateneingabebus vom Speicher her vorübergehend vom Datenbus des Prozessors getrennt werden, damit die unterbrechende Einheit den Hauptbus ohne Störung übernehmen kann.

Zum Befehlscode des SAB 8080A gehört ein spezieller, nur ein Byte langer Unterprogrammaufruf-Befehl, der die Behandlung von Programmunterbrechungen erleichtert. (Ein normaler Unterprogrammaufruf umfaßt drei Bytes). Es handelt sich um den Wiederstart-Befehl (restart-Befehl, RST). Ein drei Bit langes Feld im Operationscode von RST, das beliebig vorbesetzt werden kann, erlaubt dem unterbrechenden Gerät, einen Sprungbefehl zu einer von acht festgesetzten Speicheradressen zu geben. Die Dezimaladressen dieser Speicherplätze sind: Ø, 8, 16, 24, 32, 40, 48, und 56. Jede dieser Adressen kann dazu benutzt werden, den oder die ersten Befehle eines Unterprogramms zu speichern, das die Anforderungen des unterbrechenden Gerätes erfüllt.

Bild 8
Ablauf eines Unterbrechungsbefehls



2.5. HOLD-Zustand

Durch ein HOLD-Signal auf dem entsprechenden Eingang des Prozessors kann ein externes Gerät den Prozessor dazu veranlassen, den Adreß- und Datenbus freizugeben. Auf eine derartige Aufforderung reagiert der Prozessor damit, daß er seine Adreß- und Datenausgänge potentialfrei macht, so daß diese für andere Einheiten, die den Bus benutzen, einen hohen Widerstand darstellen. Gleichzeitig quittiert der Prozessor das HOLD-Signal, indem er den HLDA-Anschluß auf H-Pegel legt. Während eines quittierten HOLD-Zustandes werden der Adreß- und der Datenbus von der peripheren Einheit gesteuert, die das HOLD-Signal geliefert hat. Sie erhält dadurch die Möglichkeit, auf den Speicher zuzugreifen, ohne daß der Prozessor zwischengeschaltet wird.

Bild 9
HOLD-Operation (Lesen)

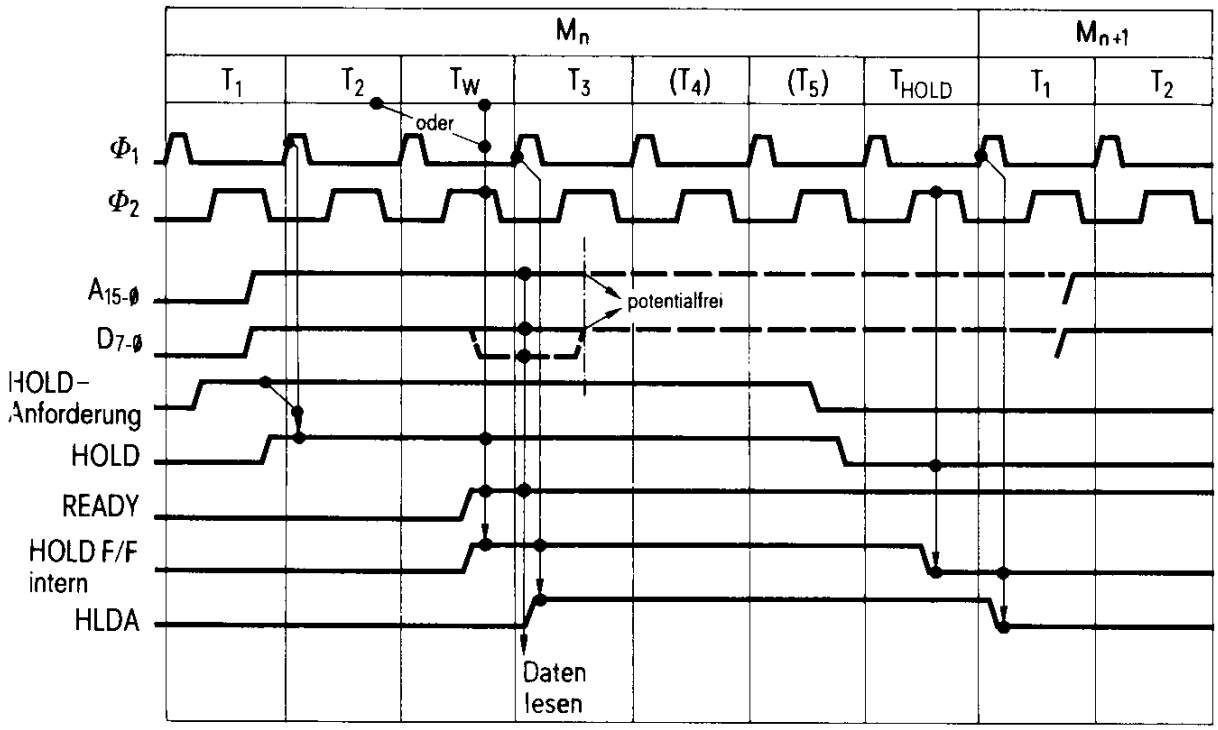
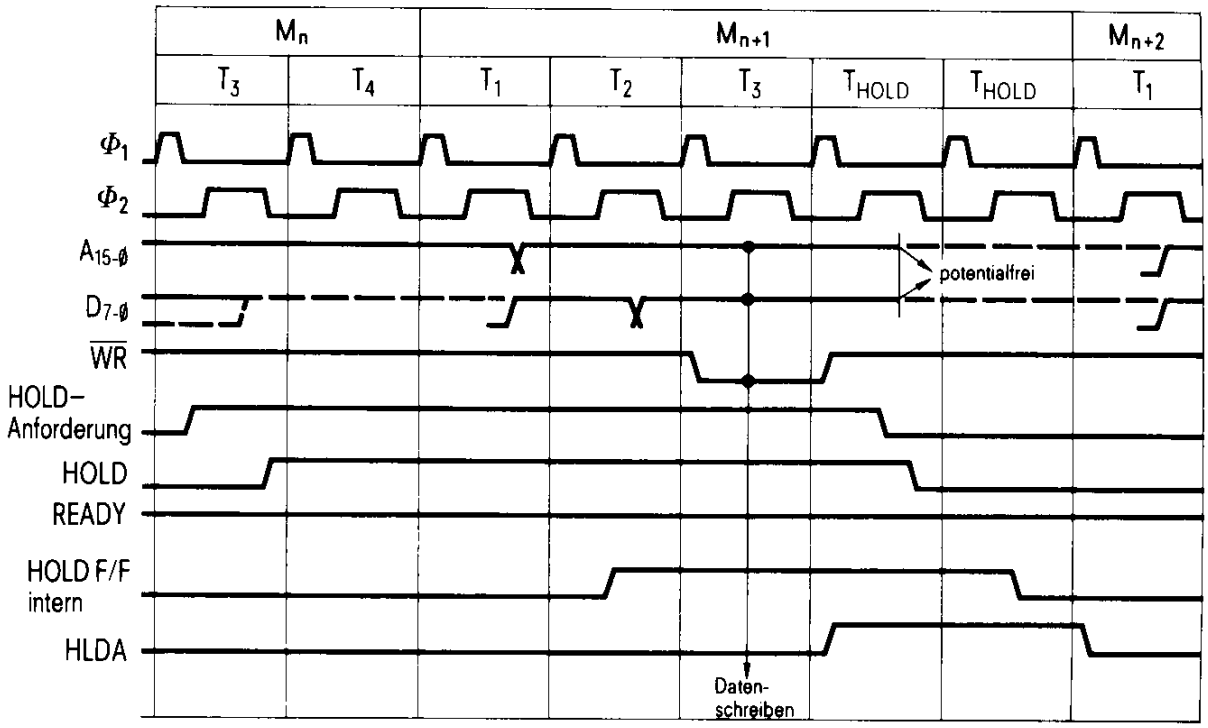


Bild 10
HOLD-Operation (Schreiben)



2.6. HALT-Zustand

Wenn ein Haltbefehl (HLT) ausgeführt werden muß, geht der Prozessor im Anschluß an den Zustand T_2 des nächsten Operationszyklus in den Haltzustand T_{WH} über. Der Prozessor hat nur drei Möglichkeiten, einen Haltzustand zu verlassen:

- Führt die RESET-Leitung H-Pegel, geht der Prozessor in den Zustand T_1 über; der Befehlszähler wird auf Ø gesetzt.
- Ein zusätzliches HOLD-Signal versetzt den Prozessor in den HOLD-Zustand, der oben beschrieben wurde. Wenn das HOLD-Signal wieder gelöscht wird, geht der Prozessor mit der ansteigenden Flanke des nächsten Φ_1-Taktimpulses in den Haltzustand zurück.
- Eine Programmunterbrechung (d.h. der INT-Eingang führt H-Pegel und das INTE-Flipflop ist gesetzt) veranlaßt den Prozessor, den Haltzustand zu verlassen, und mit der ansteigenden Flanke des nächsten Φ_1-Taktimpulses in den Zustand T_1 überzugehen.

Beachte: Das Unterbrechungsfreigabe-Flipflop INTE (=interrupt enable) muß vor dem Eintritt in den Haltzustand gesetzt worden sein. Andernfalls kann der SAB 8080A den Haltzustand nur durch ein RESET-Signal verlassen.

Bild 11
HALT-Zeitdiagramm

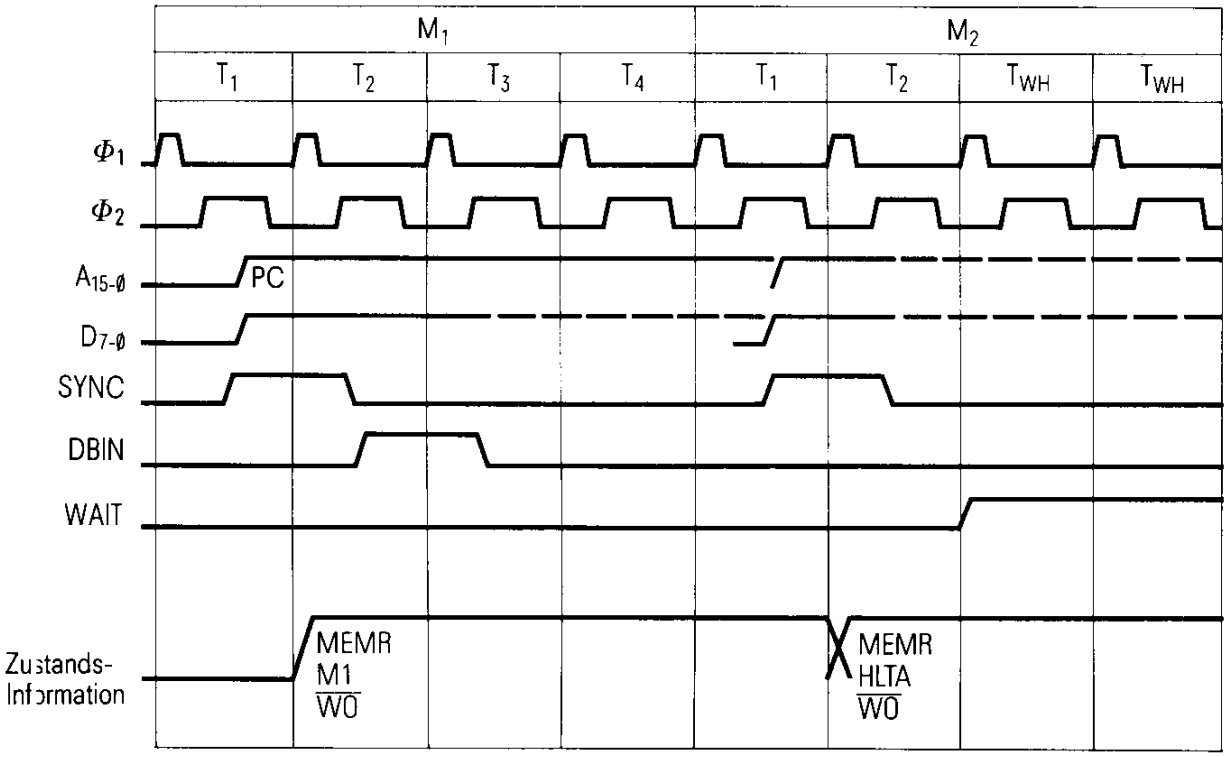
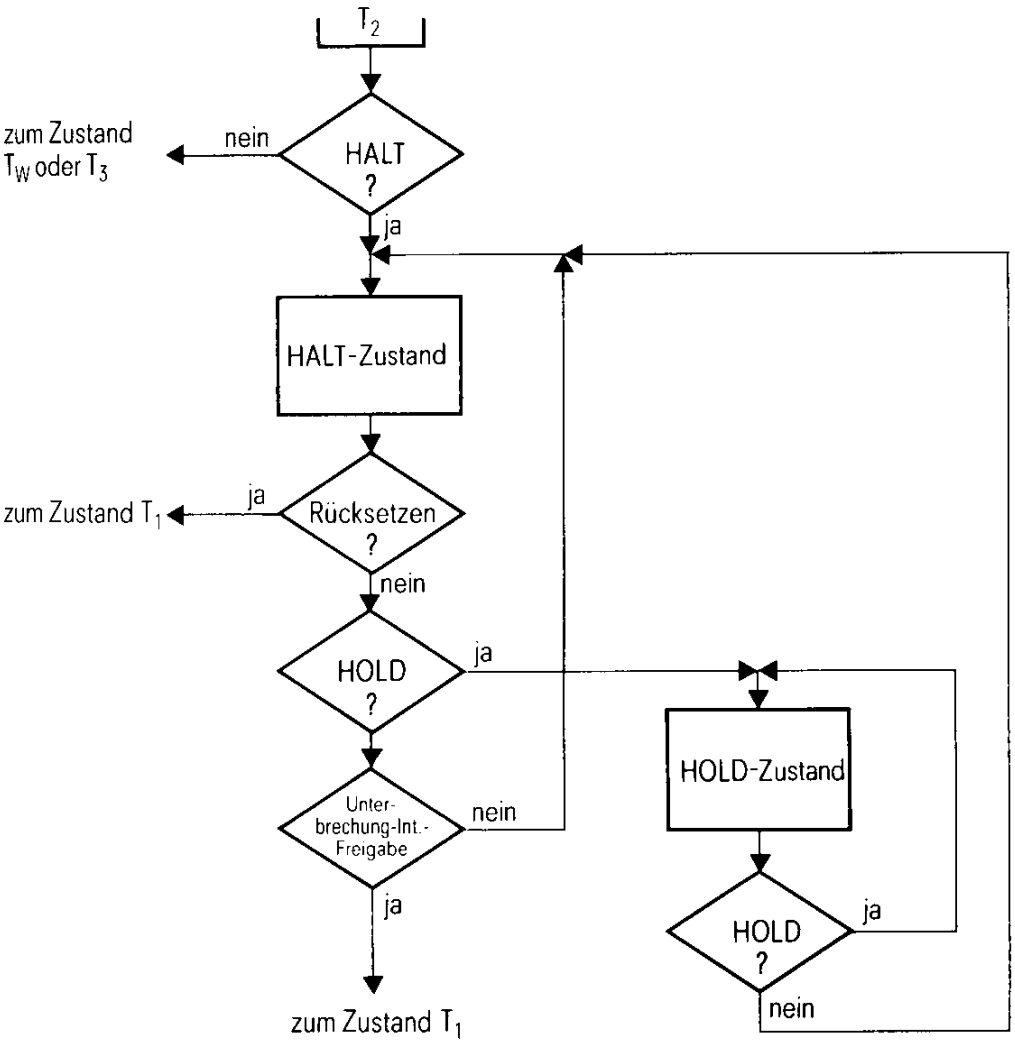


Bild 11 zeigt die zeitlichen Abläufe bei der Ausführung eines HLT-Befehls.
Bild 12 veranschaulicht die logischen Abläufe im Haltzustand, während Bild 14 die Zeitdiagramme (qualitativ) für HOLD und INT im Haltzustand aufzeigt.

Bild 12
HALT-Abläufe



2.7. Inbetriebnahme des SAB 8080A

Sobald die Spannung eingeschaltet wird, beginnt der Prozessor zu arbeiten. Der Inhalt seines Befehlszählers, seines Stackpointers und der übrigen Register ist natürlich zufälligen Einflüssen unterworfen und nicht vorbestimmt. Daher ist es notwer dig, nach dem Einschalten des Stroms ein RESET-Signal zu geben.

Ein externes RESET-Signal, das mindestens drei Taktperioden dauert, setzt den Befehlszähler des Prozessors auf Null. Die Programmausführung beginnt anschließend bei der Speicheradresse Ø. Systeme, die den Prozessor erst auf einen expliziten Befehl starten lassen wollen, müssen in dieser Speicherzelle einen Haltbefehl (HLT) ablegen. Für den Start kann man dann von Hand oder automatisch ein Unterbrechungssignal an den Prozessor geben, um ihn so zu starten. In anderen Systemen kann der Prozessor unmittelbar mit der Programmausführung beginnen. Man muß aber beachten, daß das RESET-Signal keinen Einfluß auf die Merkbits und die übrigen Register hat (also auf den Akkumulator, den Stackpointer usw.). Die Inhalte dieser Register bleiben so lange unbestimmt, bis sie durch das Programm festgesetzt werden.

Bild 13
Rücksetzvorgang

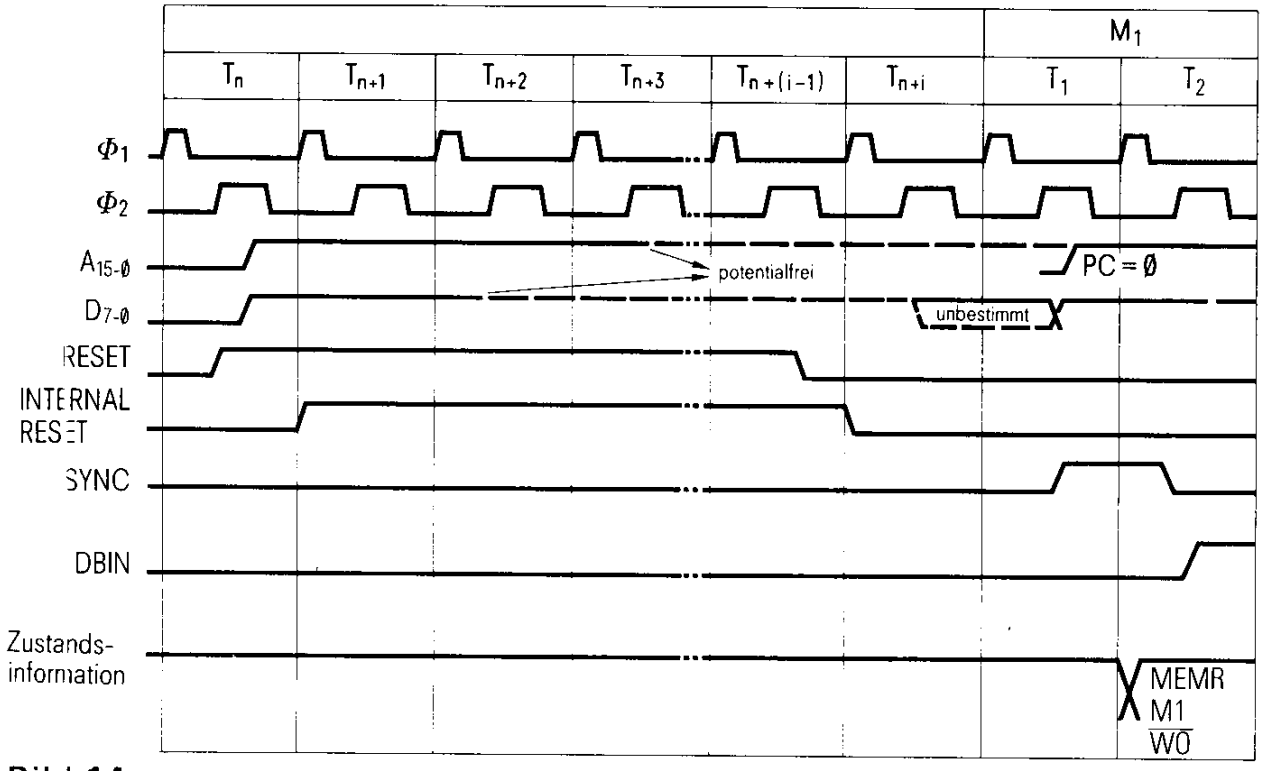
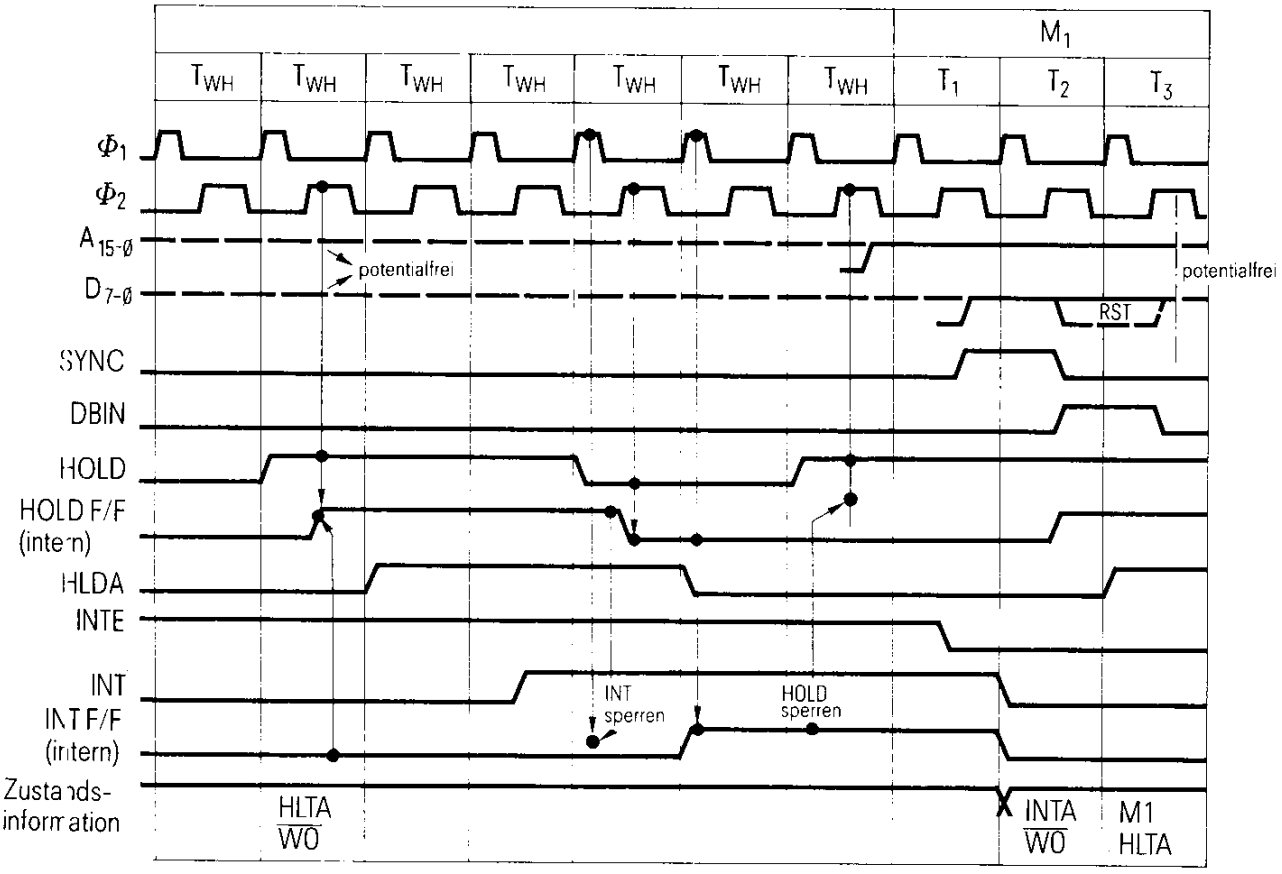


Bild 14
Beziehung zwischen HOLD und INT (Unterbrechung) im HALT-Zustand



3. Aufbau eines Mikrocomputersystems mit dem Mikroprozessor SAB 8080A

3.1. Ein Mikrocomputersystem

Wie jedes Computersystem läßt sich auch ein Mikrocomputersystem aus den drei funktionellen Blöcken Zentraleinheit, Speicher und Ein-/Ausgabe-Einheit aufbauen.

Zentraleinheit: Enthält den Mikroprozessor, den Taktgenerator sowie alle Schaltungen zur Erzeugung der Zeitabläufe und zur Anpassung an den Speicher und die Ein-/Ausgabe-Bausteine.

Speicher: Enthält Festwertspeicher (ROM) und Schreib-/Lese-Speicher (RAM) für die Speicherung von Programmen und Daten.

Ein-/Ausgabe: Enthält alle Schaltungen, die für den Verkehr des Mikrocomputers mit peripheren Geräten, z.B. Tastaturen, Anzeigen, Floppy-Disk usw., notwendig sind.

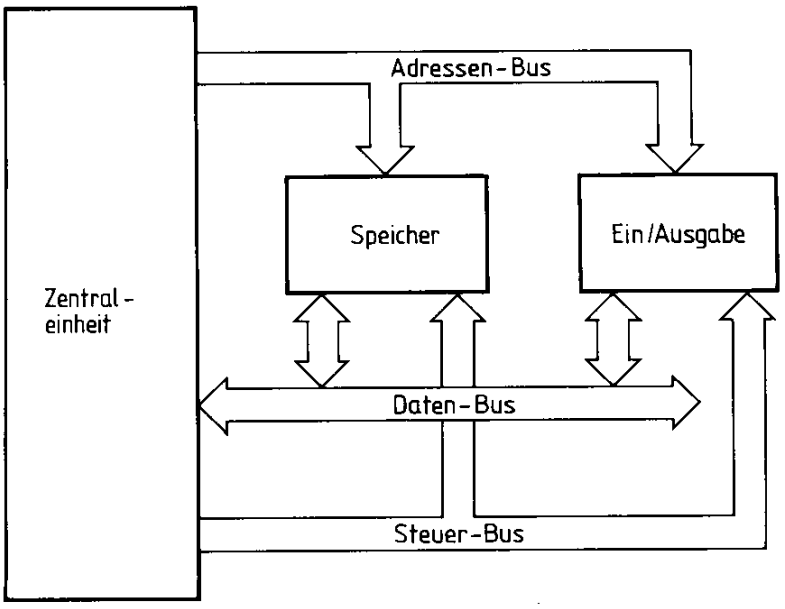
Die Verbindung dieser Einheiten werden durch die sogenannten „Busse" hergestellt. In einem einfachen Mikrocomputersystem gibt es drei Busse.

Daten-Bus: 8 Leitungen, auf denen die Daten zwischen der Zentraleinheit und Speicher bzw. Ein-/Ausgabe-Einheiten in beiden Richtungen übertragen werden.

Adressen-Bus: 16 Leitungen zur Übertragung von Adressen, von der Zentraleinheit an den Speicher und die Ein-/Ausgabe-Einheit.

Steuer-Bus: Eine Anzahl von Leitungen, die zur Übertragung der Steuersignale von der Zentraleinheit an den Speicher und die Ein-/Ausgabe-Einheiten dienen und dabei diesen mitteilen, welche Art von Operation (Kommunikation) gerade durchgeführt wird.

Bild 15
Blockschaltbild eines Computersystems



Im Allgemeinen unterscheidet man fünf verschiedene Operationen zur Kommunikation zwischen den Einheiten:
1. Speicher-Lesen (Memory Read)
2. Speicher-Schreiben (Memory Write)
3. Ein-/Ausgabe-Lesen (I/O Read)
4. Ein-/Ausgabe-Schreiben (I/O Write)
5. Unterbrechungs-Annahme (Interrupt Acknowledge)

Der grundlegende Ablauf der Operationen und die dazu erforderlichen Aktivitäten auf den drei Bussen ist wie folgt:
1. Der Mikroprozessor sendet ein Statuswort, das die Art der Operationen angibt. Die Steuerung in der Zentraleinheit aktiviert auf Grund dieser Information die entsprechende Leitung im Steuer-Bus.
2. Die Zentraleinheit sendet auf dem Adressen-Bus ein Binärwort (Adresse) aus, das die Zelle im Speicher oder die Ein-/Ausgabe-Einheit auswählt, mit der die Operation ausgeführt werden soll.
3. Die Zentraleinheit sendet (Schreiben) oder empfängt (Lesen) Daten über den Daten-Bus. Der Kommunikationspartner ist dabei die in 2. adressierte Speicherzelle oder Ein-/Ausgabe-Einheit.
4. Die Zentraleinheit startet den nächsten Zyklus, indem sie zu Schritt 1 zurückkehrt.

Wie aus diesen Punkten zu ersehen ist, stellt die Zentraleinheit die bestimmende Komponente im Mikrocomputersystem dar, von der alle Aktivitäten ausgehen und die jegliche Operation steuert.

3.2. Praktischer Aufbau der Zentraleinheit

Die Zentraleinheit umfaßt drei größere Teileinheiten (Bild 16):
1. Den Mikroprozessor SAB 8080A
2. Den Taktgeber und die 12 V Treiberstufen
3. Den bidirektionalen Treiber für den Daten-Bus und die System-Steuerlogik

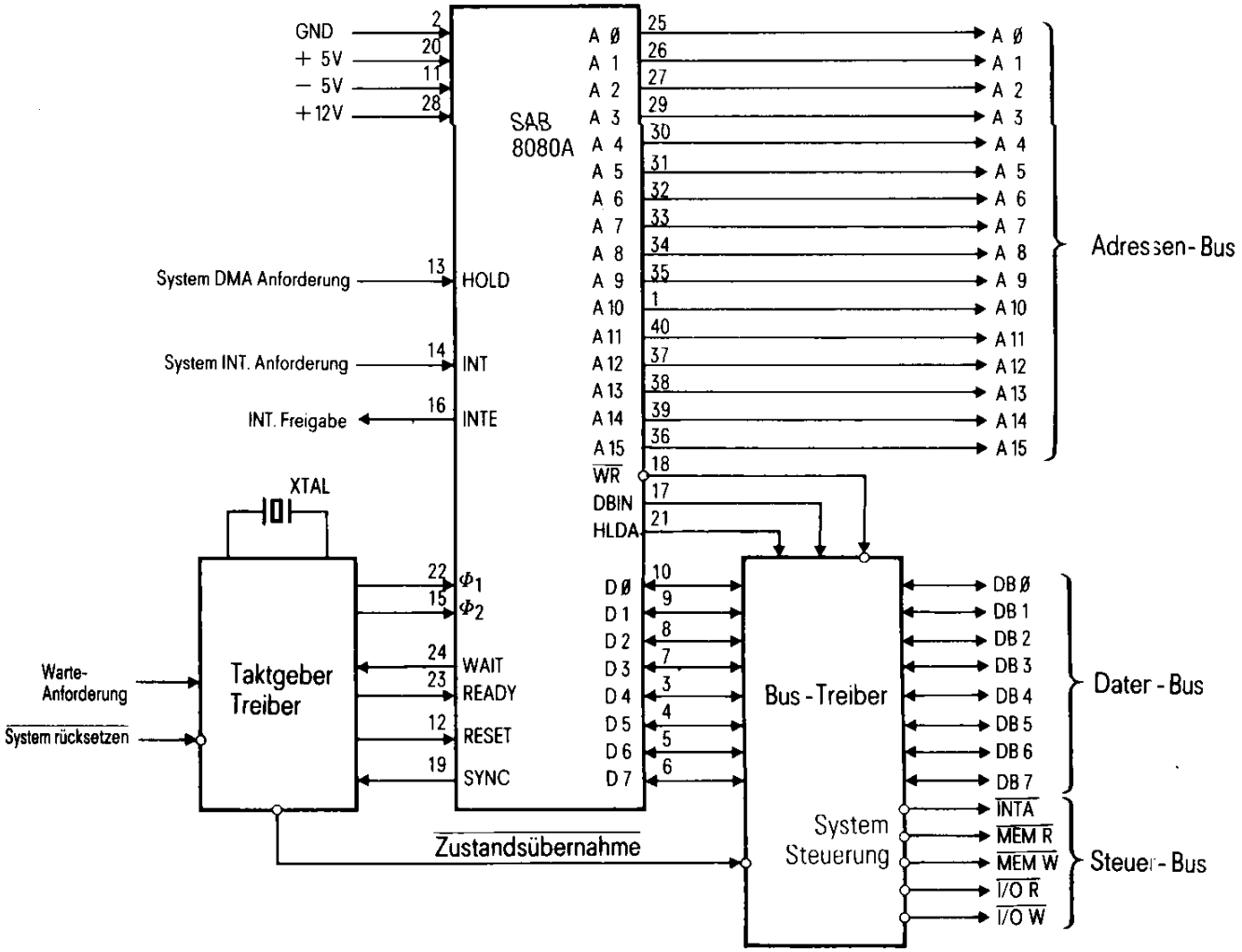
Sowohl der Taktgeber als auch die Systemsteuerung sind als Bausteine verfügbar. Es sind dies der SAB 8224 bzw. SAB 8228.

Im folgenden wird die Funktion der Einheiten in zweierlei Hinsicht beschrieben. Zum einen die Erfordernisse des Prozessors SAB 8080A an diese Einheiten, zum anderen die Erfordernisse von der Systemseite, um eine einfachere Anschaltung der Speicher und Ein-/Ausgabe-Einheiten zu erreichen.

Erzeugung der Takte und Hilfsfunktionen
Taktgeber und 12V Treiberstufen

Der SAB 8080A ist eine dynamische Schaltung, d.h. seine internen Speicherelemente und Logikschaltungen benötigen einen externen Takt zu Wiederauffrischen und zur zeitlichen Steuerung. Der SAB 8080A benötigt zwei derartige Takte, $\Phi 1$ und $\Phi 2$. Diese müssen nicht überlappende Takte sein, die den spezifizierten Zeitbedingungen genügen. Zudem sind die Takteingänge des SAB 8080A nicht TTL-kompatibel wie die anderen Signaleingänge, sondern verlangen einen Spannungshub zwischen 0,6 V und 11 V.

Bild 16
Blockschaltbild der Zentraleinheit mit dem SAB 8080A



Die beiden Takte werden auf einfache Weise aus einem quarzgesteuerten Oszillator abgeleitet. Die Frequenz wird über einen rückgekoppelten 4-Bit-Zähler durch 9 geteilt. Die Ausgänge des Zählers werden über Gatter so dekodiert, daß die Zeitbedingungen an die Takte eingehalten werden. Nachgeschaltete 12V-Treiber mit entsprechender Flankensteilheit erzeugen die SAB 8080A-kompatiblen Takte (Bild 17).

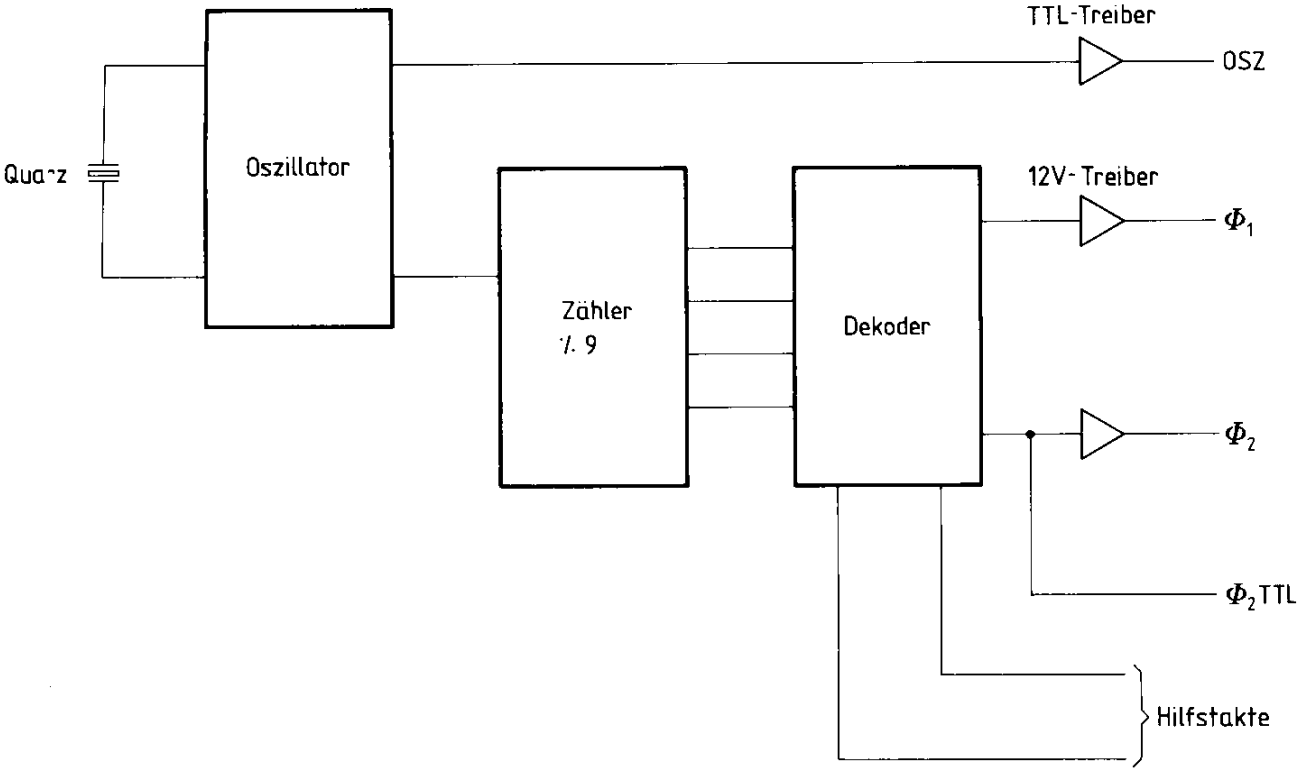
Weitere Takte

In jedem System werden einige Takte benötigt, z.B. für die Ableitung der Übertragungsfrequenz bei Datenübertragungen, Ableitungen von Zeitsignalen usw. Deshalb werden vom Taktgeber weitere Takte zur Verfügung gestellt, OSZ und Φ2TTL, die es ermöglichen oben genannte Funktionen zu realisieren.

Übernahmesignal für Statusinformation

Wie bereits erwähnt, liefert der SAB 8080A am Beginn eines jeden Maschinenzyklus eine Statusinformation, die angibt, welche Operationen in diesem Maschinenzyklus

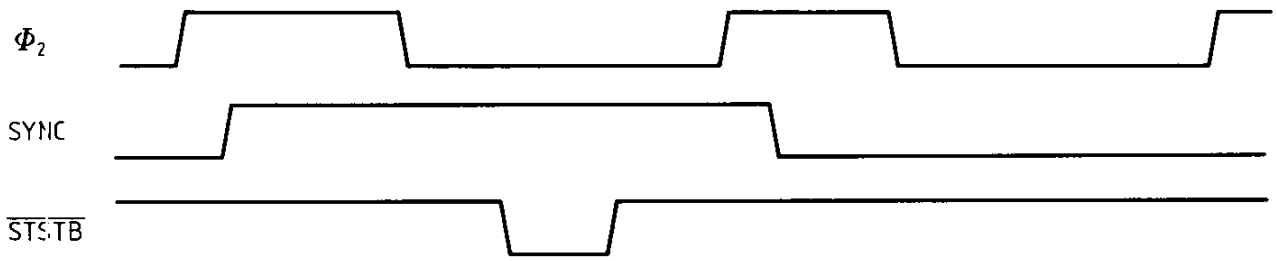
Bild 17
Blockschaltbild: Taktgeber und Treiber



mit den Speicher- bzw. E/A-Einheiten durchgeführt wird. Dieser Status muß zur Generierung der entsprechenden Steuersignale in einem Register gespeichert werden. Der SAB 8080A liefert dazu ein Gültigkeitssignal SYNC.

Um die Statusinformation möglichst frühzeitig zur Verfügung zu haben, wird das SYNC-Signal mit einem aus dem Taktgeber gewonnenen Hilfstakt verknüpft und man erhält somit ein vorgezogenes Übernahmesignal $\overline{\text{STSTB}}$.

Bild 18
Impulsdiagramm (qualitativ)



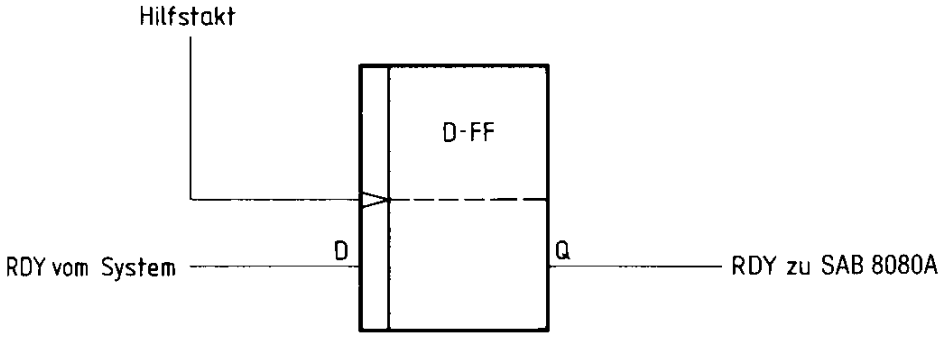
Synchronisierung des RDY-Signals

Mit dem RDY-Signal (Ready = Bereit) kann der Prozessor SAB 8080A dazu veranlaßt werden Wartezyklen in seinem normalen Befehlsablauf einzuschieben und damit die Zeiten zwischen dem Aussenden der Speicher- bzw. E/A-Anforderung und dem Abschluß der Lese- bzw. Schreiboperation zu verlängern. Damit ist es möglich

auch langsamere Einheiten, als der Prozessor sie fordert, an den SAB 8080A anzuschließen. Dieses RDY-Signal muß zu festgelegten Zeiten bezogen auf den Takt $\Phi 2$ stabil sein.

Die Synchronisierung der RDY-Anforderung mit dem vorgegebenen Taktschema kann auf einfache Weise ebenfalls vom Taktgeber übernommen werden, indem das asynchrone RDY-Signal, das von den Speicher- bzw. E/A-Einheiten erzeugt wird, mit einem vom Taktgeber erzeugten Hilfstakt in einem Flipflop gespeichert und damit für den Prozessor synchronisiert wird.

Bild 19
Synchronisierung des RDY-Signals



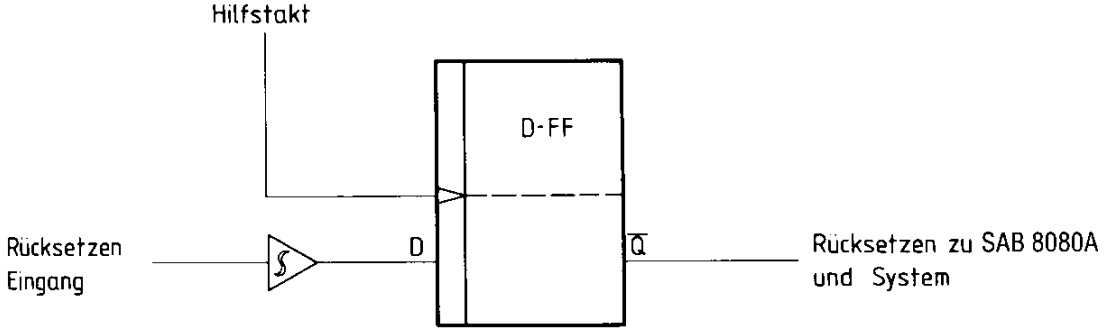
Erzeugung und Synchronisierung des RESET-Signals

Das Rücksetzsignal veranlaßt den Prozessor in einen definierten Zustand zu gehen. Insbesondere wird der Befehlszähler auf Ø gesetzt und die Ablaufsteuerung in einen definierten Zustand gebracht, so daß die nächste Aktion des Prozessors ein Befehlsholzyklus ist.

Die Erzeugung eines Rücksetz-Signals ist in folgenden Fällen wünschenswert:
- Einschalten der Versorgungsspannung
- Eingriff von außen

Die Realisierung des Rücksetz-Signals wird ebenfalls im Taktgeber durchgeführt.

Bild 20
Synchronisierung des RESET-Signals



Der Eingang ist dabei mit einem Schmitt-Trigger ausgestattet, so daß durch Beschaltung mit einem RC-Glied das Rücksetzen bei Einschalten der Versorgungsspannung erreicht wird. Die Synchronisation ist identisch der beim RDY-Signal. Das erhaltene Rücksetzsignal kann auch zur Durchführung der Rücksetzfunktion, d.h. in einen definierten Anfangszustand bringen, von anderen Systemteilen verwendet werden, z.B. Schaltung zur Steuerung einer seriellen Datenübertragung usw.

Daten-Bus-Treiber und Systemsteuerung

Der Datenverkehr zwischen dem Prozessor und dem Speicher bzw. den E/A-Einheiten wird über einen bidirektionalen (Zweiweg-)Bus abgewickelt. Dies bedeutet, daß Daten auf derselben Leitung sowohl vom Prozessor an den Speicher bzw. die E/A-Einheiten übertragen werden als auch in umgekehrter Richtung. Die Datenrichtung auf dem Bus und auch die Zeitverhältnisse werden vom Prozessor diktiert. Dabei ist zu beachten, daß von allen am Bus angeschlossenen Einheiten zu jedem Zeitpunkt nur eine Einheit Daten auf den Bus treiben darf. Alle anderen Einheiten müssen auf Eingabe bzw. hochohmig geschaltet sein. Diese Technik wird als „Tri state" (3 Zustände – „low", „high", hochohmig)-Technik bezeichnet. Sie erlaubt dem Schaltungsentwickler ein System ausgehend von einem 8-Bit-parallelen Zweiweg-Datenbus aufzubauen und durch selektives Ein- oder Abschalten der Einheiten, Daten zum oder vom Bus zu übertragen. Die Steuerung wird durch Signale vom Steuer-Bus unter Verknüpfung mit Information vom Adreß-Bus durchgeführt.

Bidirektionale Bus-Treiber

In der Spezifikation des Daten-Bus des SAB 8080A sind zwei Punkte besonders zu beachten.

- Der Eingangspegel (V_{IH}) beträgt 3,3 V (min.)
- Die Belastbarkeit der Ausgangstreiber ist im H-Zustand 150 µA und im L-Zustand 1,9 mA max.

Gemäß Eingangspegel-Spezifikation muß ein Halbleiterspeicher oder ein E/A-Baustein in der Lage sein, im H-Pegelzustand minimal 3,3 V anzubieten. Die meisten Halbleiterspeicher und Standard-TTL-E/A-Einheiten haben eine Ausgangsspannung zwischen 2,0 und 2,8 V, so daß eine direkte Verbindung zum Daten-Bus 8080 „pull-up"-Widerstände zur Spannungserhöhung erfordert. Die Werte dieser Widerstände dürfen weder die Arbeitsgeschwindigkeit des Bus vermindern noch die Treiberleistung der Speicher- bzw. E/A-Einheiten beeinträchtigen.

Der Ausgangstreiberstrom (I_{OL}) des SAB 8080A von max. 1,9 mA reicht für kleine Systeme aus. Bei Anschaltung größerer Speicherkapazität und mehreren E/A-Bausteinen ist jedoch eine Pufferung der SAB 8080A-Ausgänge nötig.

Der integrierte Systemsteuer- und Bustreiber-Baustein SAB 8228 beinhaltet sowohl eine Pegelanpassung als auch entsprechende Bus-Treiber. Reicht die hier angebotene Treiberleistung von 10 mA (I_{OL}) nicht aus, so sind spezielle Bus-Treiber vorzusehen. Dafür eignen sich besonders die Bausteine SAB 8216/8226, die je 4 bidirektionale Treiber enthalten und eine Treiberleistung von 50 mA (I_{OL}) ermöglichen (Bild 21).

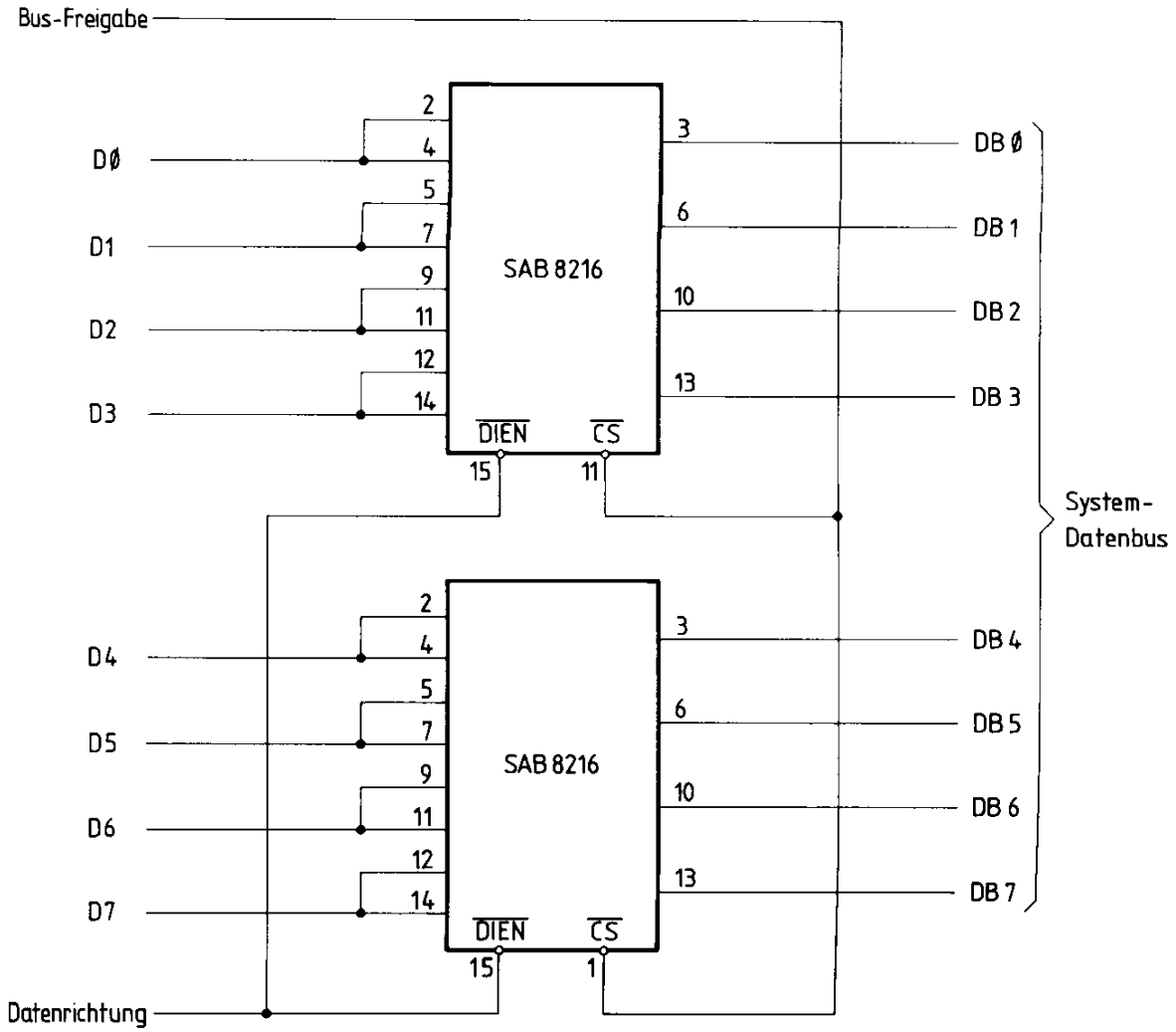
Die Richtung ist durch einen Steuereingang $\overline{\text{DIEN}}$ umschaltbar.

Durch Abschalten der Treiber mit dem Signal Bus-Freigabe gehen diese in den hochohmigen Zustand. Damit ist es möglich, einen Datenverkehr über den System-Daten-Bus zwischen angeschalteten Einheiten unter Umgehung des Prozessors abzuwickeln. Diesen Vorgang nennt man „Direkter Speicherzugriff" (DMA, direct memory access).

Erzeugung der Steuersignale

Wie schon erläutert, liefert der Prozessor SAB 8080A zu Beginn eines jeden Maschinenzyklus eine Statusinformation auf dem Daten-Bus, die Auskunft über die Art

Bild 21
System-Daten-Bus-Treiber

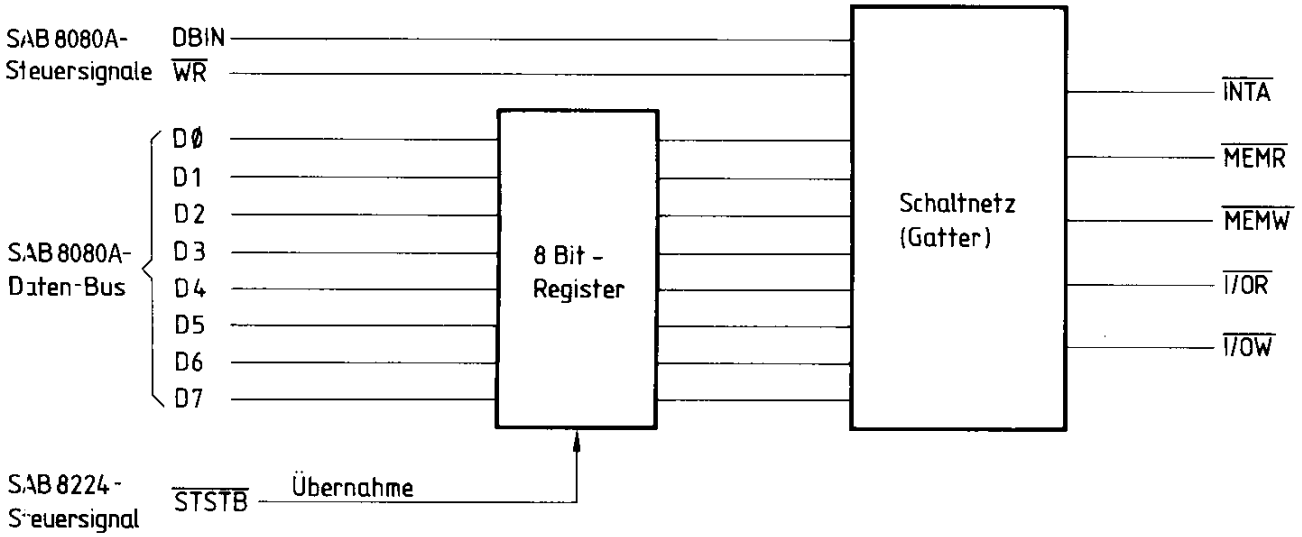


der Operation in diesem Maschinenzyklus gibt. Die Aufgabe besteht nun darin, aus dieser Information entsprechende Steuersignale, einschließlich deren Impulse und Impulslänge, bezogen auf das durch den Taktgeber vorgegebene Zeitraster zu erzeugen.

Die Statusinformation wird in ein 8-Bit-Register übernommen. Das dafür notwendige Übernahmesignal wird im Taktgeber aus dem SAB 8080A-Signal SYNC und einem speziellen Takt erzeugt. Die Ausgänge des Registers werden zusammen mit den Steuersignalen DBIN (data bus in, d.h. Daten lesen) und $\overline{WR}$ (write, Daten schreiben) durch Gatter zu den im System notwendigen Steuersignalen dekodiert.

Es sind dies:
- $\overline{MEMR}$ (memory read), Speicher lesen
- $\overline{MEMW}$ (memory write), Speicher schreiben
- $\overline{I/OR}$ (I/O read), E/A lesen
- $\overline{I/OW}$ (I/O write), E/A schreiben
- $\overline{INTA}$ (interrupt acknowledge), Unterbrechungsannahme

Bild 22
Erzeugung der Steuersignale



Diese Signale können direkt mit den meisten Speicher- bzw. E/A-Bausteinen verbunden werden, wobei eine richtige zeitliche Steuerung dieser Bausteine und des Datenverkehrs auf dem Bus ebenfalls gewährleistet ist.

Bei Durchführung eines DMA ist zu beachten, daß auch die Steuersignale von der Zentraleinheit hochohmig schaltbar sein müssen, damit die DMA-Steuereinheit die Kontrolle über den Steuer-Bus erhalten und selbst die entsprechenden Steuersignale erzeugen kann.

Wie eingangs dieses Kapitels erwähnt, werden alle bisher besprochenen Funktionen von den beiden Bausteinen SAB 8224, Taktgenerator, und SAB 8228/8238, Daten-Bus-Treiber und Systemsteuerung, durchgeführt.

Adreß-Bus-Treiber und Dekodierung

Die Treiberleistung auf dem Adreß-Bus (A15-AØ) des SAB 8080A ist wie beim Daten-Bus ausreichend für ein kleines System mit mäßiger Speichergröße und E/A-Struktur auf einer einzigen oder sehr wenigen Baugruppen. Bei diesen Systemen ist es auch sinnvoll, eine zentrale Dekodierung der Adressen für die Auswahl von Speicher- bzw. E/A-Einheiten vorzunehmen. Besonders geeignet ist der Baustein SAB 8205, der bei geringer Belastung des Adreß-Bus hohe Ausgangsleistung und kurze Schaltzeiten bietet. Ebenso sind die Freigabe-Eingänge E1 bis E3 oftmals vorteilhaft anzuwenden.

Bei mittleren und großen Systemen ist eine Adreß-Pufferung notwendig, oftmals sogar eine mehrstufige Pufferung, um die entsprechende Treiberleistung zu erhalten. Bei mehrstufiger Pufferung hat man zudem den Vorteil, daß die Treiberleistung dort erzeugt wird, wo es notwendig ist. Damit wird der Bus von der Übertragung großer Leistungen freigehalten.

Bei der Realisierung eines DMA ist zu beachten, daß auch der Adreß-Bus von mehreren Einheiten die Adreßinformation erhalten muß, und somit eine gegenseitige Verriegelung durch Hochohmig-Schalten durchzuführen ist.

Bei der Erzeugung von Steuersignalen aus den Adressen muß berücksichtigt werden, daß die Adreßausgänge des SAB 8080A zu bestimmten Zeiten zufällige Information enthalten oder auch hochohmig sein können. Dies bedingt, daß die Adressen oder die daraus erzeugten Steuersignale mit einem Gültig-Signal verknüpft werden müssen (z.B. $\overline{MEMR}$, $\overline{MEMW}$).

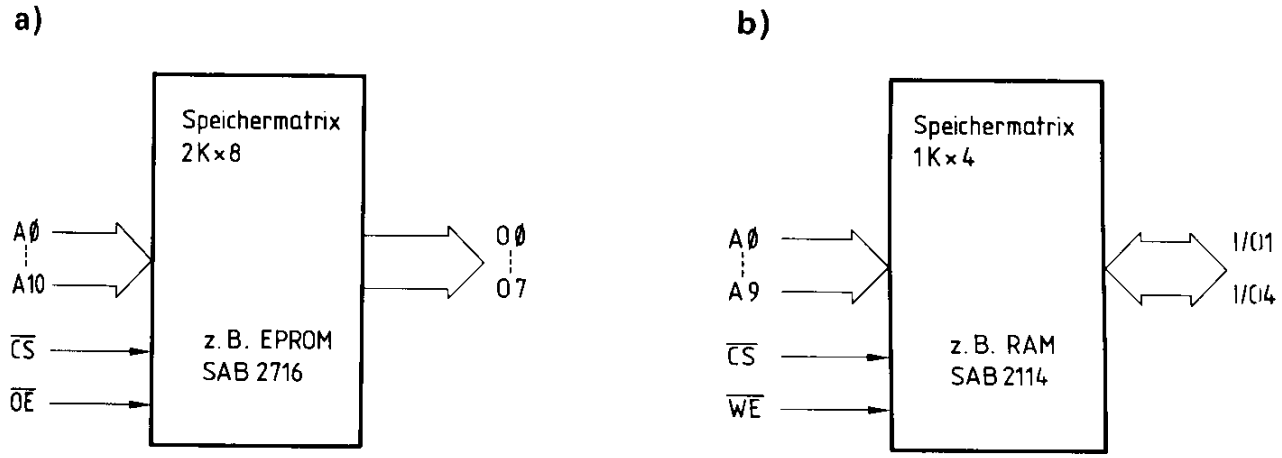
3.3. Anschluß von Speichern an den SAB 8080A

Wie im vorhergehenden Kapitel erläutert, stellt die Zentraleinheit eine Schnittstelle, bestehend aus Daten-Bus, Adreß-Bus und Steuer-Bus, zur Verfügung.

Speicher-Bausteine

Ein Standard-Speicherbaustein hat neben den Anschlüssen für die Versorgungsspannung(en) Eingänge für Adressen, Ein-/Ausgänge für Daten und je nach Art des Speichers Steuereingänge für Bausteinauswahl, Lesen und Schreiben. Die Anzahl der Adreß- und Datenleitungen hängen ab von der Kapazität (=Anzahl der Speicherelemente), der Speicherorganisation und davon, ob die Dateneingänge und -ausgänge getrennt oder gemeinsam sind.

Bild 23
Funktionales Anschlußschema verschiedener Speicherbausteine

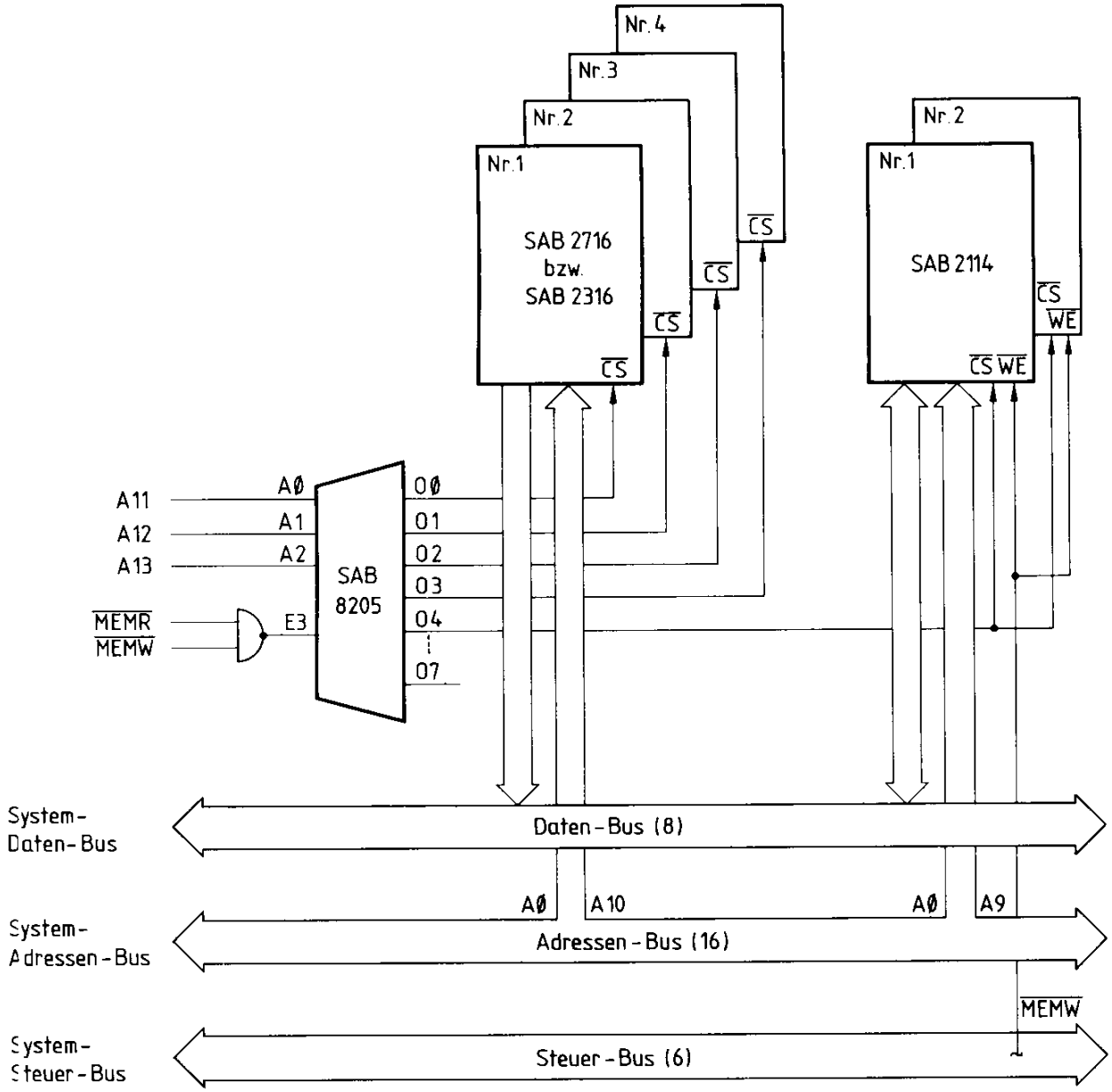


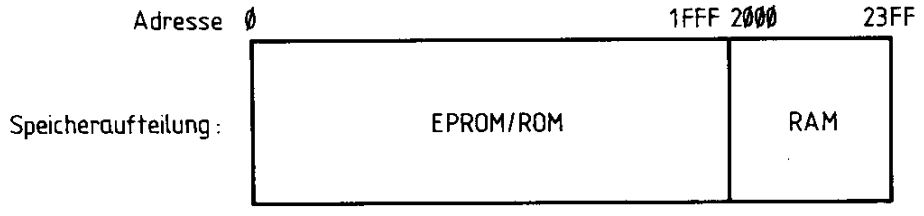
In **Bild 23a)** handelt es sich um einen EPROM-Speicherbaustein mit der Kapazität 2K×8=16384 Speicherelemente (Bit). Die Organisation ist 2K=2048 Worte zu je 8 Bit parallel.
Für die Auswahl eines Wortes sind dabei 11 Adreßbits ($2^{11}=2048$) notwendig. Da ein EPROM- bzw. ROM-Speicherbaustein normalerweise nur gelesen werden kann, sind hier nur Datenausgänge vorhanden (siehe jedoch „Programmieren von MOS-EPROM's"). Zwei UND-verknüpfte Steuersignale dienen zur Bausteinauswahl.

In **Bild 23b)** ist ein RAM-Speicherbaustein mit einer Kapazität von 1K×4=4096 Bit dargestellt. Die Organisation ist 1K=1024 Worte zu je 4 Bit parallel. Um auf eine Wortbreite von 8 Bit zu kommen, wie sie in einem SAB 8080-System gefordert ist, müssen zwei solche Speicherbausteine parallel geschaltet werden.

Für die Auswahl eines Wortes sind 10 Adreßbits notwendig. Die Dateneingänge und -ausgänge sind gemeinsam, was unmittelbar dem bidirektionalen Daten-Bus in einem SAB 8080-System entspricht. Zur Steuerung des Speicherbausteins sind ein Bausteinauswahlsignal ($\overline{CS}$) und für die Datenrichtung, d.h. Lesen oder Schreiben, ein weiteres Steuersignal ($\overline{WE}$) verfügbar.

Bild 24
Typischer Speicher für ein SAB 8080-System





Aufbau und Anschluß eines Speichers an das Bus-System

Die Auswahl der für ein System geeigneten Speicherbausteine richtet sich vor allem nach der Größe des zu realisierenden Speichers. Für kleine Speicher eignen sich besonders Bausteine mit möglichst großer Wortbreite, da damit die Bausteinanzahl klein gehalten werden kann. Für große Speicherkapazitäten hingegen sind Bausteine mit der Wortbreite 1 Bit vorteilhaft, da dabei der Dekodieraufwand minimal gehalten werden kann.

RAM-Speicherbausteine mit der höchsten Integrationsdichte, d.h. Anzahl der Speicherelemente pro Baustein, sind die dynamischen RAM's. Heute sind Bausteine mit 16K×1 Bit verfügbar (siehe SAB 2116/2117). In Kürze sind dynamische Speicherbausteine mit 64K×1 Bit zu erwarten. Bei dynamischen Speichern kommt allerdings die Problematik des Wiederauffrischens des Speicherinhalts hinzu, was von der Systemstruktur konkurrierenden Speicherzugriff vom Prozessor und der Refresheinheit bedeutet. Diese Problematik wird verringert durch die Verfügbarkeit von entsprechenden Steuerbausteinen z.B. SAB 8202.

Bild 24 zeigt den Anschluß eines typischen Speichers an ein SAB 8080-System (8K-Byte EPROM/ROM, 1K-Byte RAM).

Bei dieser Dekodierung ist zu beachten, daß der Adreßraum des SAB 8080A von 64K mehrfach belegt ist, d.h. die Adresse Ø und auch die Adresse 4ØØØ (A_{14}==1) sprechen den Baustein SAB 2716 Nr. 1 an. Analoges gilt für den RAM-Speicher im Adreßraum 2ØØØ bis 27FF.

3.4. Anschluß von Ein-/Ausgabe an den SAB 8080A

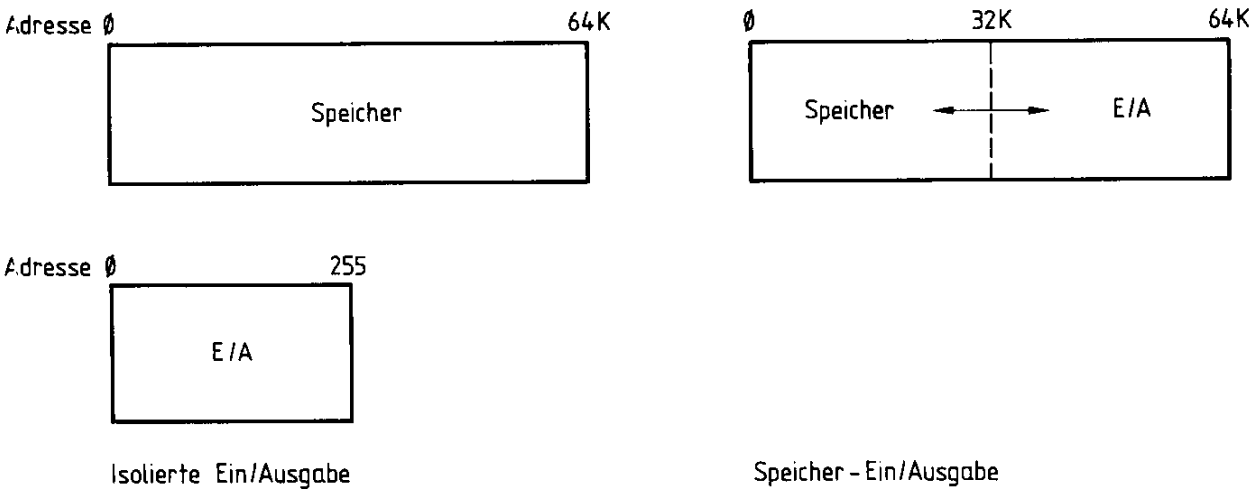
Wie der Zentralprozessor eines jeden Rechensystems, muß auch der Mikroprozessor SAB 8080A mit Elementen oder Systemen außerhalb seiner eigentlichen Computerschaltung verkehren können. Geräte wie Tastaturen, Lochstreifen, Floppy-Disks, aber auch Analoge und Digitale Signale werden verwendet, um Informationen in ein Mikrocomputersystem einzugeben. Entsprechend gilt für die Ausgabe, daß Drukker, Bildschirme, Anzeigen und auch Analogsysteme angeschlossen werden müssen.

Ein wesentlicher Punkt für die Gesamtleistung eines Systems ist die Flexibilität und Leistungsfähigkeit der zur Verfügung stehenden Ein-/Ausgabe-Bausteine und ihre Adaptionsmöglichkeit an das jeweilige Problem. Speziell diesem Gesichtspunkt ist mit einer Fülle von anwendungsspezifischen Steuerbausteinen Rechnung getragen, bei denen die gesamte Schnittstelle zwischen dem E/A-Gerät, z.B. Datensichtstation, und dem Prozessor mit seinem Bus-System vollständig von einem Baustein dargestellt wird.

Prinzipielle Möglichkeiten

Aus der Sicht des Prozessors stellt sich das gesamte E/A-System als eine Matrix von adressierbaren 1Byte Speicherzellen dar. Für den Datenverkehr zwischen diesen Speicherzellen und dem Prozessor stehen 2 Befehle (INadr und OUTadr) zur Verfügung, wobei die Übertragung der Daten zwischen dem Akkumulator und der adressierten E/A-Einheit auf dem Daten-Bus abgewickelt wird. Zwei eigene Steuersignale $\overline{I/OR}$ (E/A-Lesen) und $\overline{I/OW}$ (E/A-Schreiben) entkoppeln den E/A-Datenver-

Bild 25
Ein/Ausgabe-Verfahren



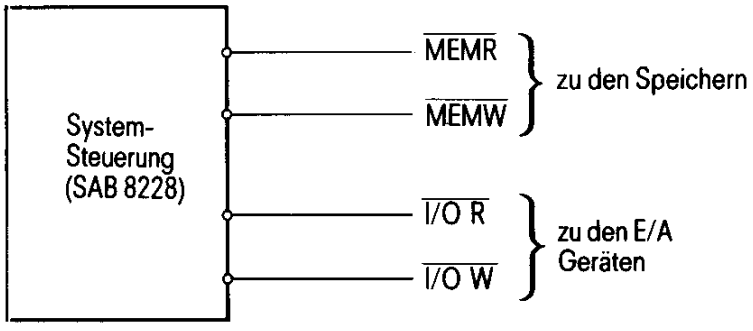
kehr von den Speicherzugriffen, der prinzipielle Ablauf ist jedoch vollkommen identisch. Dieses Verfahren der Adressierung von E/A-Bausteinen heißt „Isolierte Ein/Ausgabe" (isolated I/O). Da bei den E/A-Befehlen eine Adresse von 8 Bit zur Verfügung steht, sind je 256 E/A-Adressen für Eingabe und Ausgabe möglich.

Eine andere Methode des Aufbaus eines E/A-Systems besteht darin, die E/A-Bausteine wie Speicher zu behandeln und für sie demzufolge einen gewissen Anteil des 64K-Adreßraums des Speichers zur Verfügung zu stellen. Dieses Verfahren heißt „Speicher-Ein/Ausgabe" (memory mapped I/O). Es hat den Vorteil, daß auch für die Ein/Ausgabe-Bausteine alle Befehle wie für den Speicherzugriff anwendbar sind und somit die Durchführung von E/A-Operationen softwaremäßig eleganter und effizienter ist.

Isolierte Ein/Ausgabe

Wie im vorhergehenden Punkt bereits ausgeführt, werden bei diesem E/A-Verfahren zwei gesonderte Steuersignale $\overline{I/OR}$ und $\overline{I/OW}$ für den Datenverkehr zwischen Prozessor und E/A-Geräten verwendet. Diese Steuersignale werden vom System-Steuer-Baustein SAB 8228/8238 durch die Befehle IN bzw. OUT generiert. Die dazugehörigen Adressen sind 8-Bit breit, wodurch ein Adreßraum von je 256 Adressen für Ein- und Ausgabe verfügbar ist.

Bild 26
Isolierte Ein/Ausgabe



Speicher-Ein/Ausgabe

Durch die Reservierung eines Bereichs von Speicher-Adreßplätzen *für E/A lassen* sich E/A-Bausteine mit den gleichen Befehlen bedienen, wie sie auch für Speicherzugriffe verwendet werden. Auf diese Weise ergibt sich ein „neuer" Befehlssatz für die Bedienung von E/A-Bausteinen.

Wie **Bild 27** zeigt, werden neue Steuersignale gebildet durch Verknüpfung der $\overline{MEMR}$- und $\overline{MEMW}$-Signale mit A_{15}, dem Adreß-Bit mit der höchsten Wertigkeit. Die neuen E/A-Steuersignale sind in derselben Weise wie bei isolierten E/A angeschlossen. Infolgedessen bleiben die Eigenschaften des System-Busses gleich.

Wenn man A_{15} als E/A-Kennzeichen verwendet, hat man eine einfache E/A-Zuordnung:

Wenn $A_{15} = Ø$ ist, ist der Speicher aktiv.
Wenn $A_{15} = 1$ ist, ist E/A aktiv.

Auch andere Adreß-Bits können für diese Funktion verwendet werden. A_{15} wurde gewählt, weil es das Adreß-Bit mit der höchsten Wertigkeit ist und daher leichter mit Software gesteuert werden kann.

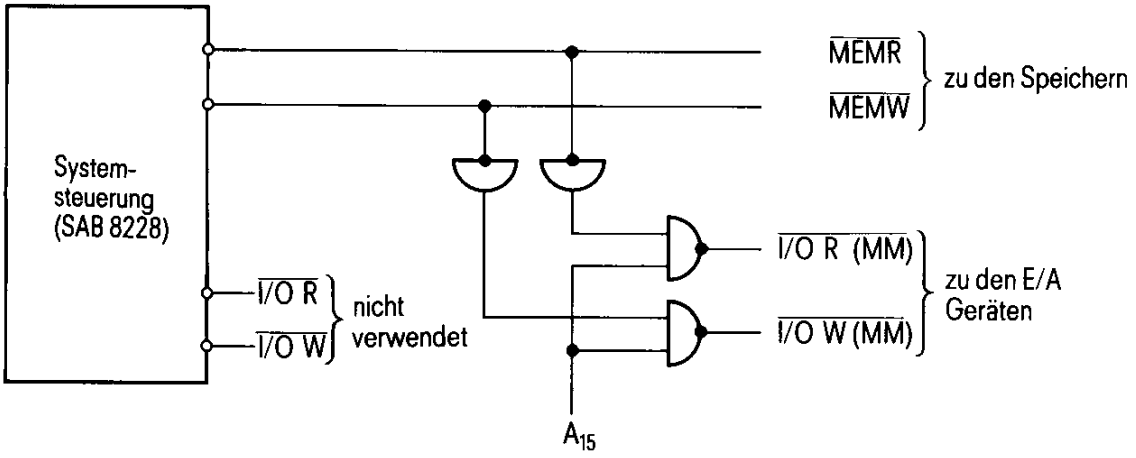
Statt des Akkumulators als einzigem Übertragungsregister, kann jetzt jedes beliebige interne Register für die Übertragung verwendet werden; praktisch lassen sich alle Befehle zur Adressierung von Speicherplätzen auch für E/A einsetzen.

Beispiele:

MOV r, M	Eingabe von E/A-Bausteininhalt zu einem beliebigen Register
MOV M, r	Ausgabe von einem beliebigen Register an E/A Baustein
MVI M, data	Daten unmittelbar zum E/A-Baustein ausgeben
LDA adr	Eingabe zum ACC (Akkumulator)
STA adr	Ausgabe von ACC zum E/A-Baustein
LHLD adr	16-Bit-Eingabe
SHLD adr	16-Bit-Ausgabe
ADD M	E/A Bausteininhalt zum ACC addieren
ANA M	E/A Bausteininhalt mit ACC UND-verknüpfen.

Aus der Aufstellung der möglichen neuen Befehle ist leicht zu ersehen, daß diese Art des E/A-Aufbaus den Durchsatz des Systems deutlich erhöhen kann.

Bild 27
Speicher-E/A



E/A-Adressierung

Für beide Möglichkeiten des Aufbaus eines E/A-Systems kann die Adressierung, d.h. Erzeugen eines Auswahlsignals, der verschiedenen Bausteine auf unterschiedliche Weise durchgeführt werden.

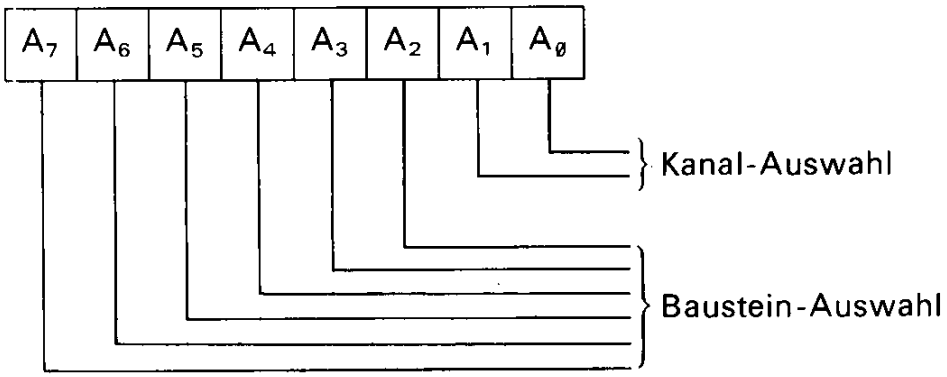
Beim Aufbau großer E/A-Systeme ist es notwendig die E/A-Adressen zu dekodieren und damit die entsprechenden Auswahlsignale zu erzeugen. Dadurch kann die vom Adreßraum her maximale Anzahl von E/A-Bausteinen angeschlossen werden.

Eine andere Methode der E/A-Adressierung ist die sogenannte „lineare Auswahl". Dabei werden nicht die Adressen dekodiert, sondern einzelne Bits des Adreß-Bus haben die Aufgabe, direkt einen E/A-Baustein auszuwählen. Diese Art der Adressierung schränkt zwar die Anzahl der anschließbaren E/A-Bausteine erheblich ein, doch ist sie für viele Systeme vollkommen ausreichend.

Zwei einfache Beispiele veranschaulichen die Leistungsfähigkeit der E/A-Adressierung. Das erste Beispiel zeigt isolierte E/A und lineare Auswahl beim Anschluß von parallelen E/A-Bausteinen SAB 8255A. Dieser Baustein beinhaltet drei 8-Bit-E/A-Ports (Kanäle), sowie ein weiteres Register für Steuer- bzw. Statussignale, d.h. er benötigt 2 Adreß-Bits für die interne Adressierung. Die restlichen 6 Adreß-Bits der IN- bzw. OUT-Befehle ermöglichen somit den Anschluß von insgesamt 6 solchen Bausteinen.

Bild 28
Isolierte E/A (lineare Auswahl, SAB 8255A)

Beispiel 1

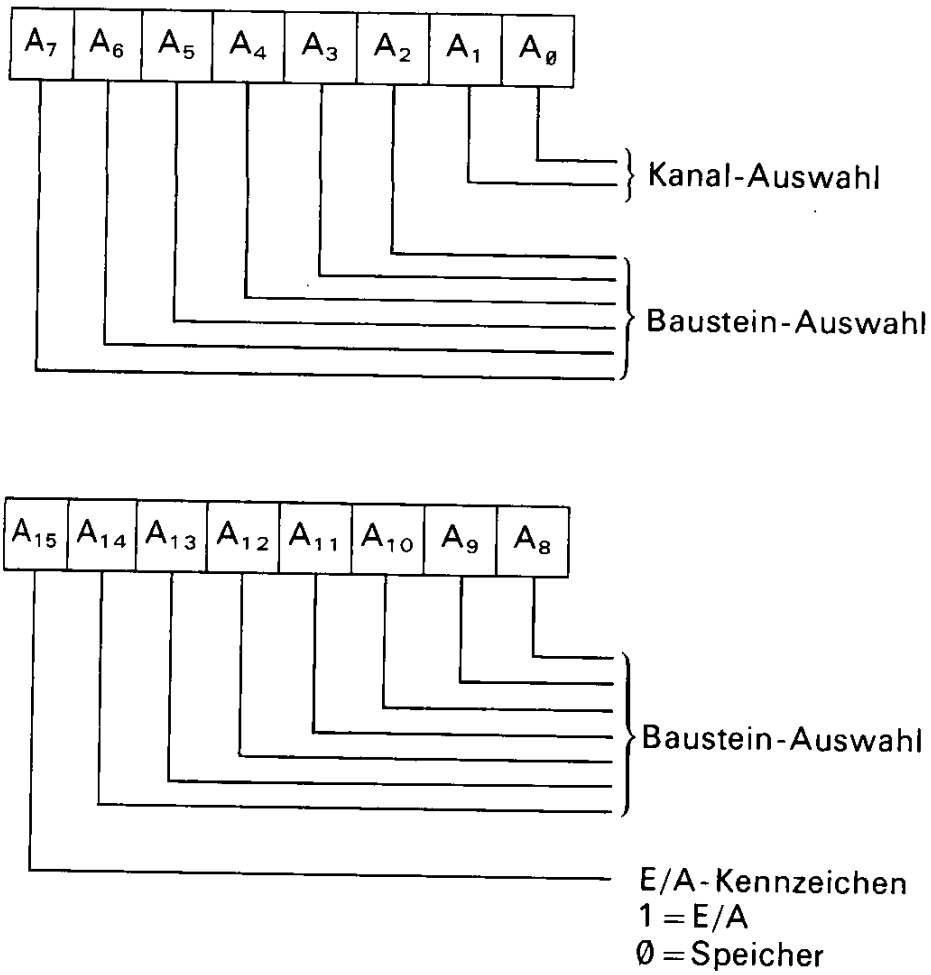


Adressiert 6 Bausteine SAB 8255A
(18 Kanäle – 144 Bits)

Im zweiten Beispiel wird statt der isolierten E/A das Speicher-E/A-Verfahren verwendet und ebenfalls lineare Auswahl. Von den vorhandenen 16 Adreß-Bits werden wiederum die beiden niederwertigsten für die Kanal-Auswahl auf dem Baustein verwendet. Das höchstwertigste Adreß-Bit A_{15} dient zur Trennung zwischen Speicher- und E/A-Operationen. Somit verbleiben 13 Bit für die Bausteinauswahl.

Bild 29
Speicher-E/A (lineare Auswahl, SAB 8255A)

Beispiel 2



Adressiert 13 Bausteine SAB 8255A
(39 Kanäle – 312 Bits)

E/A-Schaltungsbeispiel

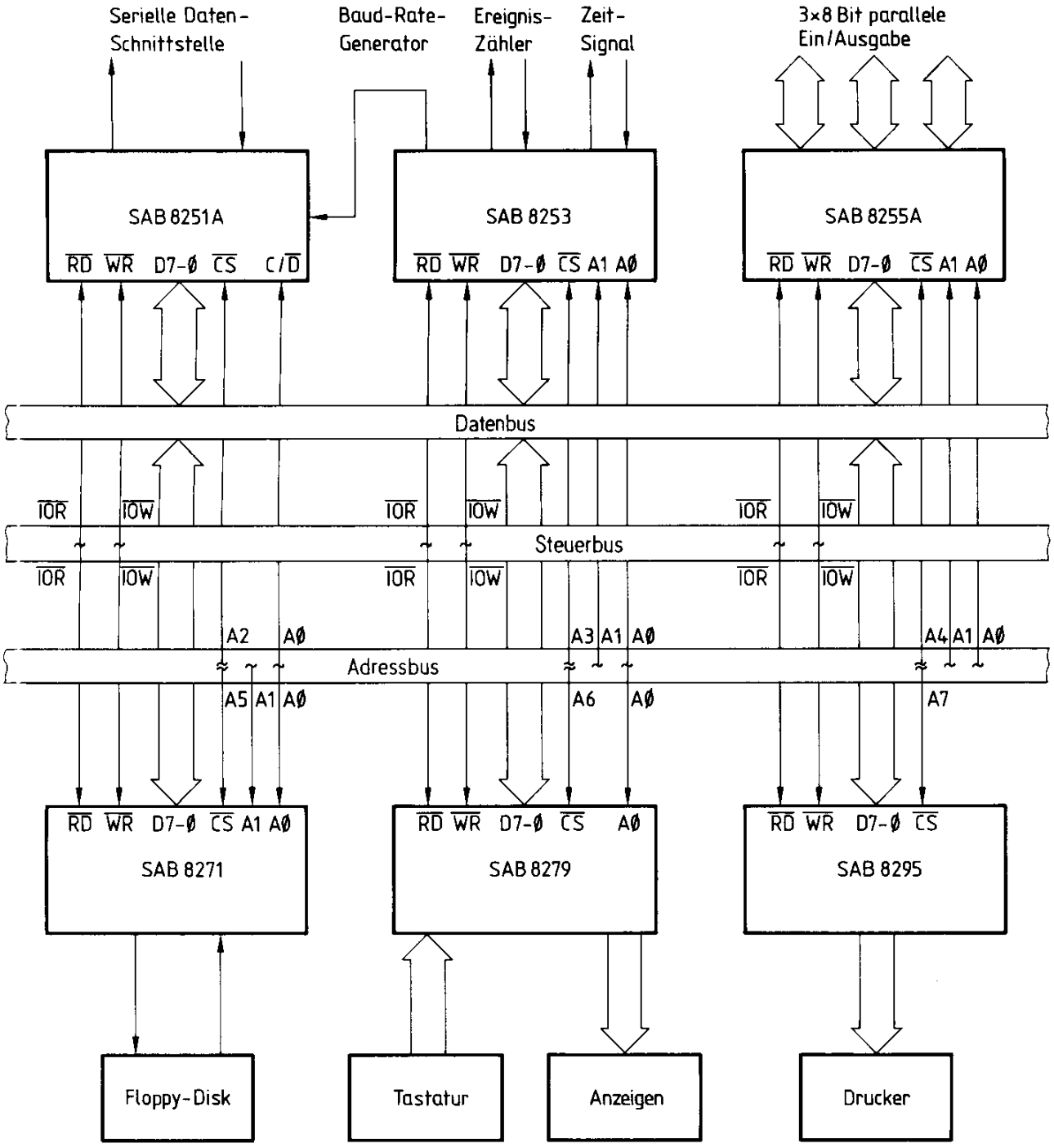
Bild 29 zeigt ein E/A-System, das mit zahlreichen Bausteinen aus der SAB 8080-Familie aufgebaut ist. Im einzelnen sind hier folgende Funktionen integriert:

SAB 8251A	Serielle E/A-Schnittstelle
SAB 8253	3 × 16-Bit-Zähler/Zeitgeber
SAB 8255A	3 × 8-Bit-parallele E/A-Schnittstelle
SAB 8271	Floppy-Disk-Steuerung
SAB 8279	Tastatur- und Anzeige-Schnittstelle
SAB 8295	5 × 7-Punkt-Matrix-Drucker-Steuerung

Dieses Beispiel ist repräsentativ für eine relativ komplexe und E/A-intensive Anwendung, wie sie z.B. in der Prozeßsteuerung auftritt.

Die einzelnen E/A-Bausteine sind als isolierte E/A mit linearer Auswahl angeschlossen. Aus den Adreß-Bits A_2 bis A_7 werden die Bausteinauswahl-Signale erzeugt. Dekodierer sind dabei nicht notwendig. Das Beispiel zeigt, wie eine sehr leistungsfähige und dabei flexible E/A-Struktur mit einem Minimum an Bausteinen aufgebaut werden kann.

Bild 30
Blockschaltbild eines E/A-Systems



4. Befehlssatz

4.1. Allgemeines

Da auch der raffinierteste Computer der Welt nur ausführt, was ihm zuvor aufgetragen worden ist, muß man ihm mitteilen, was er tun soll. Dies geschieht, indem man ihm eine Folge von Befehlen eingibt, die als „Programm" bezeichnet werden.

Das Ergebnis des Programmierens wird „Software" genannt, im Gegensatz zur „Hardware", womit Bausteine und Gerätebestandteile des Computers gemeint sind. Die Software eines Computers umfaßt alle Programme, die für den betreffenden Computer geschrieben worden sind.

Bei der Entwicklung eines Computers wird seine Zentraleinheit mit der Fähigkeit ausgestattet, bestimmte Operationen durchzuführen. Sie wird hierzu so gestaltet daß sich als Folge der Dekodierung eines bestimmten Befehls durch ihre Steuerlogik ein ganz spezieller Funktionsablauf ergibt. Die Gesamtheit der Befehle, die von einer Zentraleinheit ausgeführt werden können, stellt den Befehlssatz („Instruction Set") des Prozessors dar.

Mit jedem Befehl kann der Programmierer Operationen veranlassen, wie arithmetische und „logische" Operationen, Registerbefehle (z.B. die Erhöhung eines Registerinhalts um 1), Befehle zur Übertragung von Daten zwischen Registern, zwischen einem Register und einem Speicher oder zwischen einem Register und einem E/A-Baustein. In allen Befehlssätzen sind auch bedingte Anweisungen (conditional instructions) vorgesehen.

Ein solcher „bedingter Befehl" besagt, daß eine bestimmte Operation nur dann ausgeführt werden soll, wenn spezifizierte Bedingungen erfüllt sind, z.B. „springe, wenn das Ergebnis der letzten Operation Null war". Durch bedingte Befehle erhält ein Programm die Fähigkeit, Entscheidungen zu treffen.

Wird eine Befehlsfolge durch logische Anordnung zu einem zusammenhängenden Programm, kann der Programmierer dem Computer Anweisungen für ganze Funktionsabläufe erteilen.

Der Computer kann jedoch nur solche Programme ausführen, deren Befehle in einer binär-kodierten Form vorliegen, d.h. einer Folge von Nullen und Einsen, dem sogenannten Maschinen-Code. Da es äußerst schwierig wäre, ein Programm in diesem Code zu erstellen, wurden Programmiersprachen entwickelt. Es gibt spezielle Computerprogramme, mit deren Hilfe man jede Anweisung der Programmiersprache in den betreffenden Maschinen-Code übertragen kann, der vom Prozessor verstanden wird. Eine Art der Programmiersprache ist die „Assembler"-Sprache (Assembly-Language). Einem jeden Computerbefehl ist in dieser Sprache in eindeutiger Weise ein bestimmtes mnemonisches Symbol zugeordnet. Der Programmierer kann mit diesen mnemonischen Symbolen und Operanden ein Programm erstellen, das sogenannte Primärprogramm (Source Program). Das Primärprogramm wird anschließend in die Maschinensprache („Object Code") übersetzt. Jeder Befehl der Assembler-Sprache wird dabei von einem ASSEMBLER-Programm in einen Befehl in Maschinensprache übersetzt (aus einem oder mehreren Bytes). Assembler-Sprachen sind maschinenspezifisch, d.h. sie sind nur für einen Prozessor-Typ verwendbar.

Der Befehlssatz des SAB 8080A

Der Befehlssatz des SAB 8080A umfaßt folgende Befehlstypen:

Datentransferbefehle zur Datenübertragung zwischen dem Akkumulator, denn allgemeinen Registern, Speicher-Stellen und Ein-/Ausgabe-Bausteinen in direkter, indirekter und unmittelbarer Adressierung.

Arithmetische und logische Befehle ebenfalls mit direkter, indirekter und unmittelbarer Adressierung. Neben den Operationen mit 8-Bit-Argumenten stehen arithmetische Operationen doppelter Genauigkeit (16-Bit-Argumente) und Befehle zur Durchführung von Dezimalarithmetik zur Verfügung.

Befehle zur Programmverzweigung, wie bedingte und unbedingte Sprünge und Unterprogrammaufrufe.

Sonderbefehle, wie HALT (Anhalten des Prozessors), NOP (Leerbefehl), STC (Setzen des Übertragsbits), CMC (Komplementieren des Übertragsbits), CMA (Komplementieren des Akkumulatorinhalts), XCHG (Austausch zweier Registerpaarinhalte) und Anweisungen zum bitweisen Schieben des Akkumulatorinhalts.

Nachfolgend werden alle Befehle näher beschrieben.

Daten- und Befehls-Formate

Daten im SAB 8080A werden in der Form von 8-Bit binären ganzen Zahlen gespeichert. Alle Datenübertragungen zum Daten-Bus des Systems erfolgen mit dem gleichen Format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Daten-Wort

Die **Befehle** können 1, 2 oder 3 Bytes lang sein. Mehr-Byt-Befehle müssen in aufeinanderfolgenden Speicherstellen im Programmspeicher abgelegt werden. Die Befehlsformate hängen von der jeweils auszuführenden Operation ab.

1-Byte-Befehle

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

OP-Code

Typische Befehle

Register/Register-, Speicher-, arithmetische-, logische-, Rotations-, Rückkehr-, Stack-, Unt.-Freigabe-, Sperren-Unt.-Befehle

2-Byte-Befehle

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

OP-Code

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Operand oder E/A-Adresse

Befehle mit unmittelbarer Adressierung oder E/A-Operationen

3-Byte-Befehle

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

OP-Code

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Niederwertige Adresse oder Operand

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Höherwertige Adresse oder Operand

Befehle zum Springen, Aufrufen eines Unterprogramms oder Laden und Speichern mit direkter Adressierung

Adressierungsarten

Oft befinden sich die Daten, mit denen eine Operation ausgeführt werden soll, im Speicher. Wenn numerische Mehrbyte-Daten verwendet werden, werden auch diese, wie Befehle, in aufeinanderfolgenden Speicherplätzen gespeichert. Hierbei kommt das Byte mit der geringsten Wertigkeit zuerst, gefolgt von den Bytes mit ansteigend höherem Stellenwert. Beim SAB 8080A gibt es vier verschiedene Arten der Adressierung von Daten, die sich im Speicher oder in Registern befinden:

- Direkt – Bytes 2 und 3 des Befehls enthalten die genaue Speicheradresse des Datenwortes (die niederwertigen Bits der Adresse befinden sich in Byte 2, die höherwertigen Bits in Byte 3).
- Register – Der Befehl gibt das Register oder Registerpaar an, in dem sich die Daten befinden.
- Register, indirekt – Der Befehl gibt ein Registerpaar an, welches die Speicheradresse beinhaltet, unter der sich die Daten befinden (die höherwertigen Bits der Adresse befinden sich im ersten Register des Paares, die niederwertigen Bits im zweiten).
- Unmittelbar – Der Befehl selbst enthält die Daten. Diese können entweder aus einer 8-Bit- oder 16-Bit-Einheit bestehen (das niederwertige Byte zuerst, das höherwertige als zweites).

Mit Ausnahme von Unterbrechungs- oder Verzweigungsoperationen erfolgt die Ausführung der Befehle über aufeinanderfolgende Speicherplätze. Ein Verzweigungsbefehl kann die Adresse des nächsten durchzuführenden Befehls auf zwei Arten angeben.

- Direkt – Der Verzweigungsbefehl enthält die Adresse des nächsten Befehls, der auszuführen ist (mit Ausnahme des RST-Befehls enthält Byte 2 die niederwertige Adresse, Byte 3 die höherwertige).
- Register, indirekt – Der Verzweigungsbefehl nennt ein Registerpaar, welches die Adresse des nächsten durchzuführenden Befehls enthält (die höherwertigen Bits der Adresse befinden sich im ersten Register des Paares, die Bits mit der geringeren Wertigkeit im zweiten).

Der RST-Befehl ist ein besonderer 1-Byte-Aufruf-Befehl, der in der Regel bei Unterbrechungsfolgen verwendet wird. RST beinhaltet ein 3-Bit-Feld. Die Programmsteuerung wird dem Befehl übertragen, dessen Adresse dem 8fachen Wert dieses 3-Bit-Feldes entspricht.

Bedingungsbits (Zustandsbits)

Es gibt 5 Bedingungsbits, die bei der Ausführung eines Befehls im SAB 8080A beeinflußt werden können, bzw. die die Ausführung eines Befehls beeinflussen:

Z	(zero)	= Null
S	(sign)	= Vorzeichen
P	(parity)	= Parität
CY	(carry)	= Übertrag
AC	(aux. carry)	= Hilfsübertrag

Jedes dieser Bedingungsbits wird in einem eigenen 1-Bit-Register im SAB 8080A abgelegt.

Wenn ein Befehl den Zustand eines Bedingungsbits beeinflußt, hat dies im allgemeinen folgende Bedeutung:

Null (Z): Wenn das Resultat einer arithmetischen oder logischen Operation den Wert Null hat, wird dieses Bit gesetzt. Andernfalls wird es rückgesetzt.

Vorzeichen (S): Wenn das Bit mit der höchsten Wertigkeit als Ergebnis einer arithmetischen oder logischen Operation den Wert Eins hat, wird das Vorzeichenbit gesetzt. Andernfalls wird es rückgesetzt.

Parität (P): Wenn die Modulo2-Summe der Bits eines Ergebnisses einer Operation gleich Null ist, d.h. die Anzahl der Einsen im Binärwort ist eine gerade Zahl, wird das Paritätsbit gesetzt. Andernfalls, d.h. das Ergebnis hat ungerade Parität, wird es rückgesetzt.

Übertrag (CY): Wenn bei der Befehlsausführung ein Übertrag aus dem Bit der höchsten Wertigkeit entstanden ist (positiver Übertrag bei einer Addition bzw. negativer Übertrag (Borrow) bei einer Subtraktion oder einem Vergleich) wird das Übertragsbit gesetzt. Andernfalls wird es rückgesetzt.

Hilfsübertrag (AC): Wenn bei der Befehlsausführung ein Übertrag von Bit 3 nach Bit 4 entstanden ist, wird das Hilfsübertragsbit gesetzt. Andernfalls wird es rückgesetzt. Das Hilfsübertragsbit wird ausschließlich bei der Verarbeitung von BCD-Zahlen benötigt und von dem Befehl zur Dezimalkorrektor (DAA) ausgewertet.

Symbole und Abkürzungen

Im folgenden werden die Symbole und Abkürzungen, wie sie in der Beschreibung des Befehlssatzes des SAB 8080A verwendet werden, aufgeführt und erläutert.

Symbol	Bedeutung
A	Akkumulator, Register A
addr	16-Bit-Adresse
data	8-Bit-Daten
data 16	16-Bit-Daten
port	8-Bit-Ein-/Ausgabeadresse
byte 2	2. Byte eines Befehls
byte 3	3. Byte eines Befehls
reg, reg 1, reg 2	eines der Register A, B, C, D, E, H, L
ddd, sss	das Bitmuster, welches eines der Register reg bezeichnet (ddd = destination = Zielregister, sss = source = Quellregister)

ddd oder sss	Register
111	A
ØØØ	B
ØØ1	C
Ø1Ø	D
Ø11	E
1ØØ	H
1Ø1	L

rp	Registerpaar B steht für das Paar B, C; höherwertiges Register ist B, das niederwertige ist C. D steht für das Paar D, E; höherwertiges Register ist D, das niederwertige ist E. H steht für das Paar H, L; höherwertiges Register ist H, das niederwertige ist L. SP bezeichnet den 16-Bit-Stackpointer

rp	Registerpaar
ØØ	B, C
Ø1	D, E
1Ø	H, L
11	SP

rh	Das erste (höherwertige) Register eines angegebenen Registerpaares.
rl	Das zweite (niederwertige) Register eines angegebenen Registerpaares.
PC	16-Bit-Befehlszähler (PCH und PCL bezeichnen das höherwertige bzw. niederwertige 8-Bit-Wort).
SP	16-Bit-Stackpointer (SPH und SPL bezeichnen das höherwertige bzw. niederwertige 8-Bit-Wort).
reg_m	Bit m des Registers reg. Dabei sind die Bits von der höherwertigen zur niederwertigen Stelle (von links nach rechts) mit den Nummern 7 bis Ø gekennzeichnet).
n	RESTART-Adreßkennung (Ø bis 7)

NNN — Die binäre Darstellung ØØØ bis 111 für RESTART-Adreßkennung Ø bis 7.

Z, S, P, CY, AC — Bedingungsbits mit folgender Bedeutung

Bedingungsbit	Bedeutung
Z (zero)	Null
S (sign)	Vorzeichen
P (parity)	Parität
CY (carry)	Übertrag
AC (aux. carry)	Hilfsübertrag

condition — Bedingung bei bedingten Befehlen

CCC — Das Bitmuster, das eine bestimmte Bedingung kennzeichnet.

CCC	condition	Bedingung	
ØØØ	NZ (not zero)	Nicht Null;	Z = Ø
ØØ1	Z (zero)	Null;	Z = 1
Ø1Ø	NC (no carry)	Kein Übertrag;	CY = Ø
Ø11	C (carry)	Übertrag;	CY = 1
1ØØ	PO (parity odd)	Ungerade Parität;	P = Ø
1Ø1	PE (parity even)	Gerade Parität;	P = 1
11Ø	P (plus)	Positiv;	S = Ø
111	M (minus)	Negativ;	S = 1

( ) — Der Inhalt der Speicherstelle oder des Registers, dessen Adresse in den Klammern angegeben ist.

← — Transportpfeil; Bedeutung: „Wird übertragen nach"

∧ — Logisch UND

⊻ — Ex-ODER (exklusiv-ODER)

∨ — In-ODER (inklusiv-ODER)

\+ — Addition

− — Subtraktion mit Zweierkomplement

* — Multiplikation

↔ — „Wird ausgetauscht mit"

¯ — Das Einer-Komplement (z.B. $(\bar{A})$)

Beschreibungsformat

Auf den folgenden Seiten wird der Befehlssatz des SAB 8080A ausführlich beschrieben. Zum schnellen Auffinden sind die Befehle gemäß ihrem mnemotechnischen Code in alphabetischer Reihenfolge aufgeführt. Die Beschreibung des einzelnen Befehls erfolgt nach folgendem Format:

1. Für jeden Befehl wird der mnemonische Code und die Bedeutung angegeben.
2. Anschließend folgt eine verbale Beschreibung des Befehlsablaufs und der dabei durchgeführten Operation.
3. Eine formale Beschreibung der Operation mit Angabe des Maschinencodes, der Anzahl der Operationszyklen und Operationsschritte, sowie der Adressierungsart und der veränderten Bedingungsbits schließt die Beschreibung ab.
4. Beispiele, soweit notwendig, erläutern die Wirkung der Befehle sowie ihre Anwendung.

4.2. Befehlssatz (in alphabetischer Reihenfolge)

ACI – Addieren einer Konstanten zum Akkumulator mit Carry

Der Befehl ACI addiert den Inhalt des zweiten Befehlsbytes und das Carrybit zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation: (A) + (byte 2) + (CY) → A

Maschinencode:

1 1 0 0 1 1 1 0
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit

Beispiel

Angenommen, der Akkumulator enthält den Wert 14H und das Carrybit ist auf 1 gesetzt. Der Befehl ACI 42H bewirkt nun folgendes:

Akkumulator	= 14H	=	0001 0100
unmittelbare Daten	= 42H	=	0100 0010
	Carrybit =		1
			0101 0111 = 57H

ADC – Addieren eines Registers oder Speicherbytes zum Akkumulator mit Carry

Der Befehl ADC addiert den Inhalt eines Datenbytes und das Carrybit zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

ADC reg – Addieren eines Registers zum Akkumulator mit Carry

Dieser Befehl addiert den Inhalt des festgelegten Registers und das Carrybit zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) + (reg) + (CY) → A
Maschinencode:	1 Ø Ø Ø 1 | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

ADC M – Addieren eines Speicherbytes zum Akkumulator mit Carry

Dieser Befehl addiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, und das Carrybit zum Akkumulator. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) + ((H, L)) + (CY) → A
Maschinencode:	1 Ø Ø Ø 1 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, das Register C enthält 3DH, der Akkumulator enthält 42H und das Carrybit ist auf Null gesetzt. Der Befehl ADC C führt die Addition wie folgt aus:

```
3DH     = 0011 1101
42H     = 0100 0010
Carrybit =      0
          ---------
          0111 1111  = 7FH
```

Die Bedingungsbits werden wie folgt gesetzt:

```
Carrybit      = 0
Signbit       = 0
Zerobit       = 0
Paritybit     = 0
Hilfs-Carrybit = 0
```

Falls das Carrybit auf Eins gesetzt ist, hat der Befehl folgendes Resultat:

```
3DH     = 0011 1101
42H     = 0100 0010
Carrybit =      1
          ---------
          1000 0000  = 80H
```

```
Carrybit      = 0
Signbit       = 1
Zerobit       = 0
Paritybit     = 0
Hilfs-Carrybit = 1
```

ADD – Addieren eines Registers oder Speicherbytes zum Akkumulator

Der Befehl ADD addiert ein Datenbyte zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

ADD reg – Addieren eines Registers zum Akkumulator

Dieser Befehl addiert den Inhalt des festgelegten Registers zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) + (reg) → A
Maschinencode:	1 Ø Ø Ø Ø | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

ADD M – Addieren eines Speicherbytes zum Akkumulator

Dieser Befehl addiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, zum Akkumulator. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) + ((H, L)) → A
Maschinencode:	1 Ø Ø Ø Ø 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, der Akkumulator enthält 6CH und das Register D enthält 2EH. Der Befehl ADD D führt die Addition wie folgt aus:

```
2EH  =  ØØ1Ø 111Ø
6CH  =  Ø11Ø 11ØØ
-----------------
9AH  =  1ØØ1 1Ø1Ø
```

Nach Ausführung des Befehls ADD D enthält der Akkumulator 9AH. Der Inhalt des Registers D bleibt unverändert.

Die Bedingungsbits werden wie folgt gesetzt:

Carrybit	= Ø
Signbit	= 1
Zerobit	= Ø
Paritybit	= 1
Hilfs-Carrybit	= 1

Der folgende Befehl verdoppelt den Akkumulatorinhalt
ADD A

ADI – Addieren einer Konstanten zum Akkumulator

Der Befehl ADI addiert den Inhalt des zweiten Befehlsbytes zum Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation: (A) + (byte 2) → A

Maschinencode:

1 1 Ø Ø Ø 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, der Akkumulator enthält den Wert 14H. Der Befehl ADI 42H hat folgende Wirkung:

Akkumulator = 14H = ØØØ1 Ø1ØØ
ADI 42H = 42H = Ø1ØØ ØØ1Ø
Ø1Ø1 Ø11Ø = 56H

ANA – UNDieren eines Registers oder Speicherbytes mit dem Akkumulator

Der Befehl ANA führt eine logische UND-Operation aus. Er verwendet dazu die Inhalte des festgelegten Bytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert.

Zusammenfassung der logischen Operationen

Die logische Operation UND erzeugt im Resultat nur dann ein Bit=1, wenn die entsprechenden Bits in den Test- und den Maskendaten=1 sind.
Die logische Operation ODER erzeugt im Resultat ein Bit=1, wenn die entsprechenden Bits in den Test- oder den Maskendaten=1 sind (oder in beiden).
Die logische Operation exklusives-ODER erzeugt im Resultat nur dann ein Bit=1, wenn die entsprechenden Bits in den Test- und den Maskendaten **verschieden** sind, d.h., ein Bit=1 **entweder** in den Testdaten **oder** in den Maskendaten – aber nicht in beiden – erzeugt ein Bit=1 im Ergebnis.

UND	**ODER**	**exklusives-ODER**
1010 1010	1010 1010	1010 1010
0000 1111	0000 1111	0000 1111
0000 1010	1010 1111	1010 0101

ANA reg – UNDieren Register mit Akkumulator

Dieser Befehl UNDiert den Inhalt des festgelegten Registers mit dem Akkumulator. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit wird auf 0 zurückgesetzt.**

Operation: (A) ∧ (reg) → A

Maschinencode:

1	0	1	0	0	s	s	s

Operationszyklen: 1
Operationsschritte: 4
Adressierung: Register (sss = Quellregister)
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

ANA M – UNDieren Speicherbyte mit Akkumulator

Dieser Befehl UNDiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, mit dem Akkumulator. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit wird auf 0 zurückgesetzt.**

Operation: (A) ∧ ((H, L)) → A

Maschinencode:

1	0	1	0	0	1	1	0

Operationszyklen: 2
Operationsschritte: 7
Adressierung: Register, indirekt
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Da jedes mit Ø UNDierte Bit eine Ø erzeugt und jedes mit 1 UNDierte Bit unverändert bleibt, wird das UNDieren oft dazu benutzt, bestimmte Bitgruppen auf Ø zu setzen. Das folgende Beispiel stellt sicher, daß die höherwertigen vier Bits des Akkumulators = Ø sind und die niederwertigen vier Bits unverändert bleiben. Es wird angenommen, daß das Register C ØFH enthält

Akkumulator	=	1111 1100	=	FCH
Register C	=	ØØØØ 1111	=	ØFH
		ØØØØ 1100	=	ØCH

ANI – UNDieren einer Konstanten mit dem Akkumulator

Der Befehl ANI führt eine logische UND-Operation aus und verwendet dazu die Inhalte des zweiten Befehlsbytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit wird auf Ø zurückgesetzt.**

Operation: (A) ∧ (byte 2) → A

Maschinencode:

1 1 1 Ø Ø 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Zusammenfassung der logischen Operationen siehe Befehl ANA.

Beispiel einer logischen Operation

Der folgende Befehl wird verwendet, um das Bit 2^6 des Bytes im Akkumulator auf Ø zurückzusetzen:

ANI 1Ø111111B

Da jedes mit 1 UNDierte Bit unverändert bleibt und jedes mit Ø UNDierte auf Ø gesetzt wird, setzt der ANI-Befehl das Bit 2^6 auf Ø und läßt die übrigen unverändert. Diese Technik ist dann von Nutzen, wenn ein Programm bestimmte Bits als Zustandskennzeichen verwendet.

CALL – Unbedingter Unterprogrammaufruf

CALL stellt den Inhalt des Befehlszählers (die Adresse des nächstfolgenden Befehls) im Stack sicher und springt dann zu der im CALL-Befehl festgelegten Adresse.

Operation:
(PCH) → (SP) – 1
(PCL) → (SP) – 2
(SP) – 2 → SP
(byte 3), (byte 2) → PC

Maschinencode:

1 1 0 0 1 1 0 1
niederwert. Adreßteil
höherwert. Adreßteil

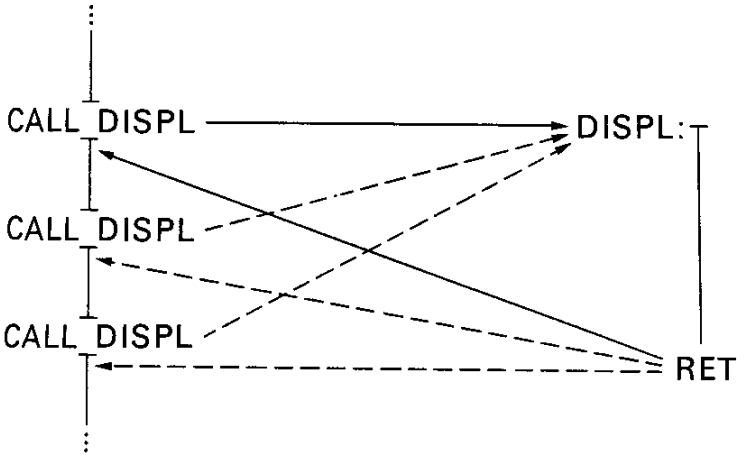
Operationszyklen: 5
Operationsschritte: 17
Adressierung: unmittelbar/Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Wird eine Codefolge mehrmals in einem Programm benötigt, kann man normalerweise Speicherplatz einsparen, indem man die Codefolge als Unterprogramm codiert. Dieses Unterprogramm wird durch den CALL-Befehl oder eine seiner Varianten aufgerufen. Z.B. betreibt ein Anwenderprogramm eine sechsziffrige Leuchtdiodenanzeige. Die Anzeige wird entweder durch eine Operatoreneingabe oder auf Grund von zwei verschiedenen Berechnungen des Programms auf den neuesten Stand gebracht. Die Programmteile zum Veranlassen der Anzeige können an jedem der drei erforderlichen Punkte eingefügt oder als Unterprogramm codiert werden. Wird dem ersten Befehl des Anzeigetreibers DISPL zugewiesen, kann der folgende CALL-Befehl verwendet werden, um das Anzeige-Unterprogramm anzustoßen:

CALL DISPL

Dieser CALL-Befehl stellt die Adresse des nächsten Befehls im Stack sicher und überträgt die Steuerung dem Unterprogramm DISPL. Das Unterprogramm DISPL muß einen Rücksprungbefehl ausführen, um den normalen Programmablauf wieder aufzunehmen. Es folgt eine graphische Darstellung der Wirkungsweise von CALL- und Rücksprungbefehlen:



Überlegungen zur Verwendung von Unterprogrammen

Je größer der zu wiederholende Programmteil ist und je öfter er wiederholt werden muß, desto größer ist die Speicherplatzersparnis, wenn dieser Programmteil als Unterprogramm geschrieben ist. Falls daher der Anzeigetreiber im vorangegangenen Beispiel einhundert Bytes braucht, würde eine gestreckte Programmierung dreihundert Bytes benötigen. Als Unterprogramm codiert, sind nur einhundert Bytes plus neun Bytes für die drei CALL-Befehle erforderlich.

Zu beachten ist, daß Unterprogramme die Verwendung eines Stacks erfordern. Dies verlangt, daß das Anwendersystem einen Schreib-/Lesespeicher (RAM) für den Stack besitzt. Wenn das Anwendersystem sonst keinen RAM braucht, kann es sein, daß der Systemplaner die Verwendung von Unterprogrammen zu vermeiden sucht.

Ccondition – Bedingter Unterprogrammaufruf

Wenn die spezifizierte Bedingung erfüllt ist, wird ein Unterprogrammaufruf durchgeführt, d.h. der Inhalt des Befehlszählers (die Adresse des nächstfolgenden Befehls) wird auf den Stack gebracht und das Programm an der im Befehl angegebenen Adresse fortgesetzt.
Ist die Bedingung nicht erfüllt, wird das Programm mit dem nächstfolgenden Befehl fortgesetzt.

Operation: Wenn Bedingung (CCC) erfüllt:

(PCH) → (SP) – 1
(PCL) → (SP) – 2
(SP) – 2 → SP
(byte 3), (byte 2) → PC

Maschinencode:

1 1	C C C	1 Ø Ø
niederwert. Adreßteil		
höherwert. Adreßteil		

Operationszyklen: 3 oder 5[1])
Operationsschritte: 11 oder 17[1])
Adressierung: unmittelbar/Register, indirekt
Veränderte Bedingungsbits: keine

[1]) Erste Zahl gilt, wenn Bedingung nicht erfüllt ist, die Zweite, wenn Bedingung erfüllt ist.

CMA – Negieren des Akkumulators

Der Befehl CMA negiert jedes Akkumulatorbit, um das Einerkomplement zu erzeugen. Alle Bedingungsbits bleiben unverändert.

Operation:	$(\overline{A}) \rightarrow A$
Maschinencode:	0 0 1 0 1 1 1 1
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	keine

Beispiel

Angenommen, der Akkumulator enthält den Wert 51 H. Wird er durch CMA negiert, entsteht AEH.

51 H = 0101 0001
AEH = 1010 1110

CMC – Negieren des Carrybit

Falls das Carrybit=Ø ist, wird es durch CMC auf 1 gesetzt. Ist das Carrybit=1, setzt CMC es auf Ø zurück.

Operation:	$(\overline{CY}) \rightarrow CY$
Maschinencode:	Ø Ø 1 1 1 1 1 1
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	Carrybit

CMP – Vergleich zwischen einem Register- oder Speicherbyte und dem Akkumulator

Der Befehl CMP vergleicht das festgelegte Byte mit dem Akkumulatorinhalt und zeigt das Resultat durch Setzen des Carry- und des Zerobits an. Die verglichenen Werte bleiben unverändert.

Ist das Zerobit=1, zeigt dies Gleichheit an. Ist das Carrybit=Ø, zeigt dies an, daß der Akkumulator größer oder gleich als das festgelegte Byte ist. Ist das Carrybit=1, ist der Akkumulator kleiner als das Byte.

Vergleiche werden so ausgeführt, daß das festgelegte Byte vom Akkumulatorinhalt subtrahiert wird. Dies ist der Grund, warum das Zero- und das Carrybit das Resultat anzeigen. Diese Subtraktion verwendet die internen Register des Prozessors, so daß die Ursprungsdaten erhalten bleiben.

CMP reg – Vergleichen eines Registers mit dem Akkumulator

Dieser Befehl vergleicht den Inhalt des festgelegten Registers mit dem Inhalt des Akkumulators durch Durchführung einer Subtraktion. Das Vergleichsergebnis ist an den Bedingungsbits abzulesen.

Z = 1 : (A) = (reg)
CY = Ø : (A) ≧ (reg)
CY = 1 : (A) < (reg)

Der Inhalt von A und reg bleibt unverändert.

Operation:	(A) – (reg)
Maschinencode:	1 Ø 1 1 1 | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss=Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

CMP M – Vergleichen eines Speicherbytes mit dem Akkumulator

Dieser Befehl vergleicht den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, mit dem Akkumulatorinhalt.

Beschreibung siehe CMP reg.

Operation:	(A) – ((H, L))
Maschinencode:	1 Ø 1 1 1 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel 1

Angenommen, der Akkumulator enthält den Wert ØAH und Register E den Wert Ø5H. Der Befehl CMP E führt folgende interne Subtraktion aus (man muß sich daran erinnern, daß eine Subtraktion in Wirklichkeit eine Zweierkomplementaddition ist):

```
Akkumulator      = ØØØØ 1Ø1Ø
+(–Register E)   = 1111 1Ø11
                   ØØØØ Ø1Ø1  +  (–Carry)
```

Nach dem Negieren des Carrybits als Folge der Subtraktion sind das Zero- und das Carrybit=Ø, was anzeigt, daß der Inhalt des Akkumulators größer als der des Registers E ist.

Beispiel 2

Angenommen, der Akkumulator enthält den Wert – 1BH und Register E den Wert Ø5H:

```
Akkumulator      = 1110 Ø1Ø1
+(–Register E)   = 1111 1Ø11
                   1110 ØØØØ  +  (–Carry)
```

Nach dem Negieren des Carrybits sind das Carry- und das Zerobit beide=Ø. Normalerweise zeigt dies an, daß der Inhalt des Akkumulators größer als der des Registers E ist. Die Bedeutung des Carrybits wird jedoch umgesetzt, weil die Werte verschiedene Vorzeichen haben. Für die richtige Interpretation des Carrybits ist das Anwenderprogramm verantwortlich.

CPI – Vergleichen einer Konstanten mit dem Akkumulator

Der Befehl CPI vergleicht die in byte 2 angegebene Konstante mit dem Akkumulatorinhalt.

Beschreibung siehe CMP reg.

Operation: (A) – (byte 2)

Maschinencode:

1 1 1 1 1 1 1 0
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Siehe Befehl CMP.

DAA – Dezimalkorrektur des Akkumulators

Der Befehl DAA setzt den 8-Bit-Wert im Akkumulator so um, daß er zwei binärcodierte (BCD-) Ziffern mit je 4 Bits bildet.

Der Befehl DAA wird bei der Addition von Dezimalzahlen verwendet. Er ist der einzige Befehl, für dessen Funktion das Hilfs-Carrybit benötigt wird. In arithmetischen Mehrbyteoperationen wird der DAA-Befehl typischerweise unmittelbar hinter dem arithmetischen Befehl codiert, damit das Hilfs-Carrybit nicht unbeabsichtigt geändert wird, bevor es vom DAA ausgewertet wurde.

Operation:

1. Falls die niederwertigen vier Bits des Akkumulators einen Wert größer 9 haben oder das Hilfs-Carrybit auf 1 gesetzt ist, wird 6 zum Akkumulator addiert.
2. Falls die höherwertigen vier Bits des Akkumulators nun einen Wert größer 9 haben oder das Carrybit auf 1 gesetzt ist, wird 6 auf die höherwertigen vier Bits des Akkumulators addiert.

Maschinencode: | Ø Ø 1 Ø Ø 1 1 1 |

Operationszyklen: 1
Operationsschritte: 4
Adressierung: Register
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel 1

Angenommen, der Akkumulator enthält den Wert 93H als Resultat einer Addition der BCD-Zahlen 93 und 8.

```
1001 0011          93 im BCD-Format
0000 1000          08 im BCD-Format
-----------
1001 1011  = 9BH
```

Carrybit = Ø Hilfs-Carrybit = Ø

Ein jetzt gegebener DAA-Befehl addiert 6 zum Akkumulatorinhalt, weil die niederwertigen vier Bits größer 9 sind.

```
1001 1011          9BH im BCD-Format
     0110          06H im BCD-Format
-----------
1010 0001  = A1H
```

Carrybit = Ø Hilfs-Carrybit = 1

Jetzt haben die vier höherwertigen Bits einen Wert größer 9, deshalb addiert der DAA-Befehl 6 dazu.

```
1010 0001          A1H im BCD-Format
0110               06H im BCD-Format
-----------
0000 0001
```

Carrybit = 1 Hilfs-Carrybit = 1

Wenn der DAA-Befehl beendet ist, enthält der Akkumulator den Wert Ø1 im BCD-Format. Das Carry- und das Hilfs-Carrybit sind auf 1 gesetzt. Da das eigentliche Resultat der Addition 1Ø1 ist, ist das Carrybit bedeutsam für das Programm. Dafür, daß diese Information verwendet wird, ist das Programm selbst verantwortlich. Zu beachten ist, daß der Stand des Carrybits verloren ist, sobald das Programm irgendeinen Befehl ausführt, der das Carrybit ändert.

Beispiel 2

Angenommen, es sind die zwei im BCD-Format gespeicherten Dezimalzahlen 88 und 99 zu addieren

```
   1ØØØ 1ØØØ        88 im BCD-Format
+  1ØØ1 1ØØ1        99 im BCD-Format
   ØØ1Ø ØØØ1  = 21H
```

Carrybit = 1 Hilfs-Carrybit = 1

Ein jetzt gegebener DAA-Befehl addiert 6 auf die niederwertigen vier Bits, weil das Hilfs-Carrybit=1 ist, und 6 auf die höherwertigen vier Bits, weil das Carrybit=1 ist.

```
   ØØ1Ø ØØØ1
+        Ø11Ø
   ØØ1Ø Ø111
+  Ø11Ø
   1ØØØ Ø111        87 im BCD-Format
```

Carrybit = 1 Hilfs-Carrybit = 1

Der Stand des Carrybits=1 muß für das eigentliche Resultat von 187 vom Programm berücksichtigt werden.

DAD – 16-Bit-Addition

Der Befehl DAD addiert den 16-Bit-Wert des festgelegten Registerpaars zum Inhalt des Registerpaars H und L. Das Resultat wird in H und L gespeichert.

Der Befehl DAD kann nur die Inhalte der Registerpaare B & C, D & E, H & L oder des Stackpointers SP zum Inhalt des Registerpaares H & L addieren.

Der Befehl DAD setzt das Carrybit auf 1, wenn es einen Überlauf aus dem H- und L-Register gibt. DAD beeinflußt keines der Bedingungsbits, mit Ausnahme des Carrybits.

Operation: (H, L) + (rp) → H, L

Maschinencode:

Ø Ø	r p	1 Ø Ø 1

Operationszyklen: 3
Operationsschritte: 10
Adressierung: Register (rp = Registerpaar)
Veränderte Bedingungsbits: Carrybit

Beispiel

Der Befehl DAD bietet die Möglichkeit, den augenblicklichen Inhalt des Stackpointers sicherzustellen.

```
LXI H, ØØH      ; Löschen von H & L zu null
DAD SP          ; Stackpointer nach H & L bringen
SHLD SAVSP      ; Sicherstellen des Stackpointers im Speicher
```

Der Befehl DAD H verdoppelt die Zahl in den Registern H und L. Dies gilt nicht, wenn die Operation einen Überlauf des H-Registers verursacht.

DCR – Dekrementieren eines Registers oder Speicherbytes

Der Befehl DCR subtrahiert 1 vom Inhalt des festgelegten Registers oder Speicherbytes. DCR beeinflußt alle Bedingungsbits, **mit Ausnahme** des Carrybits. Da der DCR-Befehl das Carrybit beibehält, kann er in Programmen zur Mehrbyte-Arithmetik verwendet werden, um Schleifenzähler zu dekrementieren oder für ähnliche Zwecke.

DCR reg – Dekrementieren Register

Der Befehl subtrahiert 1 vom Inhalt des festgelegten Registers. Das Carrybit wird von diesem Befehl nicht verändert.

Operation: (reg) − 1 → reg

Maschinencode:

Ø Ø	d d d	1 Ø 1

Operationszyklen: 1
Operationsschritte: 5
Adressierung: Register (ddd=Zielregister)
Veränderte Bedingungsbits: Zero-, Sign-, Parity- und Hilfs-Carrybit.

DCR M – Dekrementieren Speicherbyte

Dieser Befehl subtrahiert 1 vom Inhalt der Speicherstelle, die durch die Register H und L adressiert wird. Das Carrybit wird von diesem Befehl nicht verändert.

Operation: ((H, L)) − 1 → (H, L)

Maschinencode:

Ø Ø 1 1 Ø 1 Ø 1

Operationszyklen: 3
Operationsschritte: 10
Adressierung: Register, indirekt
Veränderte Bedingungsbits: Zero-, Sign-, Parity- und Hilfs-Carrybit.

Beispiel

Der DCR-Befehl wird oft verwendet, um Mehrbyte-Operationen zu steuern, z.B. das Übertragen einer Zeichenmenge von einem Speicherbereich zu einem anderen:

```
        MVI B, 5H     ; Setzen des Schleifenzählers
        LXI H, 260H   ; Laden H&L mit Ursprungsadresse
        LXI D, 900H   ; Laden D&E mit Zieladresse
LOOP:   MOV A, M      ; Laden des zu übertragenden Bytes
        STAX D        ; Speichern des Bytes
        DCX D         ; Dekrementieren der Zieladresse
        DCX H         ; Dekrementieren der Ursprungsadresse
        DCR B         ; Dekrementieren des Schleifenzählers
        JNZ LOOP      ; Wiederholen bis Schleifenzähler=Ø
```

Dieses Beispiel zeigt außerdem eine wirtschaftliche Programmiertechnik. Zu beachten ist, daß der Schleifenzähler auf Ø dekrementiert wird, und nicht bis zum Erreichen der gewünschten Anzahl inkrementiert wird. Diese Technik vermeidet die Notwendigkeit eines Vergleichsbefehls und spart dadurch Speicherplatz und Ausführungszeit.

DCX – Dekrementieren eines Registerpaares

Der Befehl DCX subtrahiert 1 vom Inhalt des festgelegten Registerpaars. DCX beeinflußt keines der Bedingungsbits.

Der Befehl DCX kann nur die Registerpaare B & C, D & E, H & L oder den Stackpointer SP dekrementieren.

Operation: (rp) − 1 → rp

Maschinencode:

Ø Ø	r p	1 Ø 1 1

Operationszyklen: 1
Operationsschritte: 5
Adressierung: Register (rp = Registerpaar)
Veränderte Bedingungsbits: keine

Beispiel

Angenommen, die Register H und L enthalten die Adresse 98ØØ H, wenn der Befehl DCX H ausgeführt wird. DCX betrachtet die Inhalte der zwei Register als einen einzelnen 16-Bit-Wert und ‚borgt' sich eine 1 vom H-Register, um den Wert 97 FFH zu erzeugen.

DI – Unterbrechung sperren

Das Unterbrechungssystem wird gesperrt, wenn der Prozessor eine Unterbrechung anerkennt oder **unmittelbar** nach der Ausführung eines DI-Befehls.

Maschinencode:	1 1 1 1 0 0 1 1
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	keine

Beispiel

In Anwendungsprogrammen mit Unterbrechungs-Logik wird der DI-Befehl häufig verwendet, wenn eine Befehlsfolge nicht unterbrochen werden darf. Zum Beispiel werden zeitabhängige Befehlsfolgen ungenau, wenn sie unterbrochen werden. Man kann das Unterbrechungssystem sperren, indem man einen DI-Befehl am Anfang der Befehlsfolge einfügt. Da das Auftreten von Unterbrechungen nicht voraussagbar ist, sollte man einen EI-Befehl am Ende der zeitabhängigen Befehlsfolge einfügen.

EI – Unterbrechung freigeben

Der Befehl EI gibt das Unterbrechungssystem **nach der Ausführung des nächsten Programmbefehls** frei. Die Freigabe des Unterbrechungssystems wird um einen Befehl verzögert, um den Unterprogrammen zur Unterbrechungsbehandlung die Rückkehr zum Hauptprogramm zu ermöglichen, bevor eine nachfolgende Unterbrechung anerkannt wird.

Maschinencode:	1 1 1 1 1 Ø 1 1
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	keine

Beispiel

Nach Rücksetzen des Prozessors durch Anlegen eines RESET-Signals ist das Unterbrechungssystem gesperrt. Falls in einem Anwendersystem Unterbrechungen vorgesehen sind, muß der EI-Befehl Bestandteil der Initialisierungsroutine sein.

HLT – Halt

Der Befehl HLT hält den Prozessor an. Der Befehlszähler enthält weiterhin die Adresse des nächstfolgenden Befehls. Ferner bleiben die Bedingungsbits und die Register unverändert.

Maschinencode:	Ø 1 1 1 Ø 1 1 Ø
Operationszyklen:	1
Operationsschritte:	7
Veränderte Bedingungsbits:	keine

Beispiel

Sobald der Prozessor im Haltzustand ist, kann er nur durch ein externes Ereignis neu gestartet werden, typischerweise eine Unterbrechung. Deshalb sollte man sicherstellen, daß Unterbrechungen freigegeben sind, bevor der HLT-Befehl ausgeführt wird. Dazu wird auf die Beschreibung des EI-Befehls verwiesen.

Wenn beim Mikroprozessor SAB 8080A ein HLT-Befehl ausgeführt wird, während die Unterbrechungen gesperrt sind, ist der einzige Weg zum Neustarten des Prozessors, daß man ein RESET-Signal anwendet.

Der Prozessor kann den Haltzustand zeitweise verlassen, um eine DMA-Anforderung (direkter Speicherzugriff) zu bedienen. Der Prozessor geht jedoch wieder in den Haltzustand zurück, sobald die Anforderung bedient ist.

IN – Eingeben von einem Kanal

Der Befehl IN liest acht Bits aus dem festgelegten Kanal (port) und lädt sie in den Akkumulator.

Operation: (port) → A

Maschinencode:

1 1 Ø 1 1 Ø 1 1
port

Operationszyklen: 3
Operationsschritte: 10
Adressierung: direkt
Veränderte Bedingungsbits: keine

INR – Inkrementieren eines Registers oder Speicherbytes

Der Befehl INR addiert 1 zum Inhalt des festgelegten Registers oder Speicherbytes. INR beeinflußt alle Bedingungsbits, **mit Ausnahme** des Carrybits. Da der INR-Befehl das Carrybit beibehält, kann er in Programmen zur Mehrbyte-Arithmetik verwendet werden, um Schleifenzähler zu inkrementieren oder für ähnliche Zwecke.

INR reg – Inkrementieren Register

Der Befehl addiert 1 zum Inhalt des festgelegten Registers.

Operation: (reg) + 1 → reg

Maschinencode:

Ø Ø	d d d	1 Ø Ø

Operationszyklen: 1
Operationsschritte: 5
Adressierung: Register (ddd = Zielregister)
Veränderte Bedingungsbits: Zero-, Sign-, Parity- und Hilfs-Carrybit.

INR M – Inkrementieren Speicherbyte

Dieser Befehl addiert 1 zum Inhalt der Speicherstelle, die durch die Register H und L adressiert wird.

Operation: ((H, L)) + 1 → (H, L)

Maschinencode:

Ø Ø 1 1 Ø 1 Ø Ø

Operationszyklen: 3
Operationsschritte: 10
Adressierung: Register, indirekt
Veränderte Bedingungsbits: Zero-, Sign-, Parity- und Hilfs-Carrybit.

Beispiel

Falls das Register C 99H enthält, inkrementiert der Befehl INR C den Registerinhalt auf 9AH.

INX – Inkrementieren eines Registerpaares

Der Befehl INX addiert 1 zum Inhalt des festgelegten Registerpaares. INX beeinflußt keines der Bedingungsbits. Da der INX-Befehl alle Bedingungsbits beibehält, kann er zur Adreßmodifikation in Programmen zur Mehrbyte-Arithmetik verwendet werden.

Operation: (rp) + 1 → rp

Maschinencode:

Ø	Ø	r	p	Ø	Ø	1	1

Operationszyklen: 1
Operationsschritte: 5
Adressierung: Register (rp = Registerpaar)
Veränderte Bedingungsbits: keine

Beispiel

Angenommen, die Register D und E enthalten den Wert Ø1FFH. Der Befehl INX D inkrementiert den Wert auf Ø2ØØH. Im Gegensatz dazu beachtet der Befehl INR E nicht den Überlauf aus dem niederwertigen Byte und erzeugt Ø1ØØH als Resultat. (Dieser Fall kann durch Abfragen des Zerobits festgestellt werden.)

Falls der Stackpointer den Wert FFFFH enthält, inkrementiert der Befehl INX SP den Inhalt des Stackpointers auf ØØØØH. Der INX-Befehl setzt keine Bedingungsbits, um diesen Fall anzuzeigen.

Jcondition – Bedingter Sprung

Wenn die spezifizierte Bedingung erfüllt ist, wird ein Sprung ausgeführt, d.h. das Programm wird an der im Befehl angegebenen Adresse fortgesetzt.

Ist die Bedingung nicht erfüllt, wird das Programm mit dem nächstfolgenden Befehl fortgesetzt.

Operation: Wenn Bedingung (CCC) erfüllt:
(byte 3), (byte 2) → PC

Maschinencode:

1 1	C C C	Ø 1 Ø
niederwert. Adreßteil		
höherwert. Adreßteil		

Operationszyklen: 3
Operationsschritte: 10
Adressierung: unmittelbar
Veränderte Bedingungsbits: keine

Beispiel

Siehe Befehl JMP

JMP – Unbedingter Sprung

Der Befehl JMP ändert die Ausführungsfolge, indem er die in den Bytes 2 und 3 des JMP-Befehls festgelegte Adresse in den Befehlszähler lädt.

Operation: (byte 3), (byte 2) → PC

Maschinencode:

1 1 Ø Ø Ø Ø 1 1
niederwert. Adreßteil
höherwert. Adreßteil

Operationszyklen: 3
Operationsschritte: 10
Adressierung: unmittelbar
Veränderte Bedingungsbits: keine

Beispiel

Dieses Beispiel zeigt drei verschiedene, aber gleichwertige Methoden, um auf einen von zwei Punkten eines Programms zu springen, abhängig davon, ob das Signbit einer Zahl gesetzt ist oder nicht. Angenommen, das zu testende Byte steht im Register C.

```
Methode 1:   EINS:   MOV ← A, C
                     ANI ← 8ØH
                     JZ  ← PLUS
                     JNZ ← MINUS

Methode 2:   ZWEI:   MOV ← A, C
                     RLC
                     JNC ← PLUS
                     JMP ← MINUS

Methode 3:   DREI:   MOV ← A, C
                     ADI ← Ø
                     JM  ← MINUS

             PLUS:                    ; bei Signbit = Ø

             MINUS:                   ; bei Signbit = 1
```

Der Befehl ANI (UNDieren einer Konstanten mit dem Akkumulator) im Block EINS setzt alle Bits auf Ø, mit Ausnahme des Signbits, das unverändert bleibt. Falls das Signbit=Ø ist, wird das Zerobit auf 1 gesetzt. Der JZ-Befehl bewirkt dann, daß die Programmsteuerung dem Befehl bei PLUS übertragen wird. Andernfalls erhöht der JZ-Befehl einfach den Befehlszähler um 3 und der JNZ-Befehl wird ausgeführt. Dieser bewirkt, daß die Programmsteuerung dem Befehl bei MINUS übertragen wird. (Das Zerobit wird von keinem der Sprungbefehle beeinflußt.)

Der Befehl RLC (Rotieren des Akkumulators nach links) im Block ZWEI bewirkt, daß das Carrybit dem Signbit des Datenbytes gleichgesetzt wird. Falls das Signbit=Ø ist, bewirkt der JNC-Befehl einen Sprung nach PLUS. Andernfalls wird der JMP-Befehl ausgeführt, der die Steuerung unbedingt nach MINUS überträgt. (Zu beachten ist, daß man z.B. einen JC-Befehl anstelle des unbedingten Sprungbefehls hätte verwenden können, mit dem selben Ergebnis).

Der Befehl ADI (Addieren einer Konstanten zum Akkumulator) im Block DREI bewirkt, daß die Bedingungsbits gesetzt werden. Falls das Signbit=1 ist, bewirkt der JM-Befehl, daß die Programmsteuerung nach MINUS übertragen wird. Andernfalls geht die Programmsteuerung automatisch auf die PLUS-Routine über.

LDA – Direktes Laden des Akkumulators

Der Befehl LDA lädt den Akkumulator mit einer Kopie des Bytes, das an der Speicherstelle steht, die durch die Bytes 2 und 3 des LDA-Befehls angesprochen ist.

Operation: ((byte 3), (byte 2)) → A

Maschinencode:

Ø Ø 1 1 1 Ø 1 Ø
niederwert. Adreßteil
höherwert. Adreßteil

Operationszyklen: 4
Operationsschritte: 13
Adressierung: direkt
Veränderte Bedingungsbits: keine

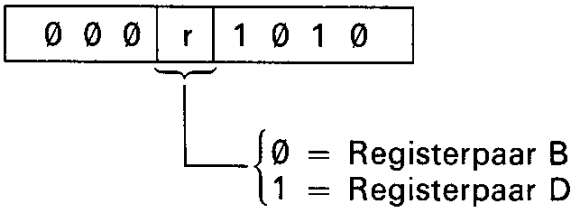
LDAX – Indirektes Laden des Akkumulators

Der Befehl LDAX lädt den Akkumulator mit einer Kopie des Bytes, das an der Speicherstelle steht, die durch den Inhalt der Registerpaare B, C oder D, E adressiert ist.

Dieser Befehl kann nur das Registerpaar B oder D festlegen.

Operation: ((r)) → A

Maschinencode:



Operationszyklen: 2
Operationsschritte: 7
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Angenommen, das Register D enthält 93H und das Register E 8BH. Der folgende Befehl lädt den Akkumulator mit dem Inhalt der Speicherstelle 938BH:

```
LDAX    D
```

LHLD – Direktes Laden der Register H und L

Der Befehl LHLD lädt das Register L mit einer Kopie des Bytes, das an der Speicherstelle steht, die durch die Bytes 2 und 3 des LHLD-Befehls festgelegt ist. LHLD lädt dann das Register H mit einer Kopie des Bytes, das an der nächsthöheren Speicherstelle steht.

Operation: ((byte 3), (byte 2)) → L
((byte 3), (byte 2)+1) → H

Maschinencode:

Ø Ø 1 Ø 1 Ø 1 Ø
niederwert. Adreßteil
höherwert. Adreßteil

Operationszyklen: 5
Operationsschritte: 16
Adressierung: direkt
Veränderte Bedingungsbits: keine

Beispiel

Angenommen, die Stellen 3ØØØ H und 3ØØ1 H enthalten die Adresse Ø64 EH, gespeichert in der Form 4 EØ6. In der nachstehenden Befehlsfolge überträgt der MOV-Befehl eine Kopie des Bytes, das im Speicherplatz mit der Adresse Ø64 EH steht, zum Akkumulator.

```
LHLD    3ØØØ H     ; Eintragen der Adresse
MOV     A, M       ; Laden des Akkumulators aus Adresse
```

LXI – Laden eines Registerpaares mit einer Konstanten

Der Befehl LXI lädt den Inhalt des zweiten Befehlsbytes in das niederwertige Register (rl) des angegebenen Registerpaares und den Inhalt des dritten Befehlsbytes in das höherwertige Register (rh). Der Befehl LXI kann die Registerpaare B, C, D, E, H, L oder den Stackpointer laden.

Operation: (byte 2) → rl
(byte 3) → rh

Maschinencode:

Ø Ø	r p	Ø Ø Ø 1
niederwert. Datenteil		
höherwert. Datenteil		

Operationszyklen: 3
Operationsschritte: 10
Adressierung: unmittelbar (rp = Registerpaar)
Veränderte Bedingungsbits: keine

Beispiel

Üblicherweise wird LXI dazu verwendet, eine Speicheradresse für den Gebrauch nachfolgender Befehle einzusetzen. In der nachstehenden Befehlsfolge lädt der LXI-Befehl die Adresse von STRNG in die Register H und L. Der MOV-Befehl lädt dann die an dieser Adresse gespeicherten Daten in den Akkumulator.

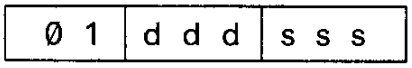
```
LXI H,STRNG     ; Eintragen der Adresse
MOV A,M         ; Laden String-Daten nach Akkumulator
```

MOV – Datenübertragung

Der Befehl MOV überträgt ein Datenbyte, indem es den Inhalt der Quelle in das Ziel kopiert. Die Ursprungsdaten bleiben unverändert. Die Befehlsoperanden legen fest, ob die Übertragung von Register zu Register, von Register zum Speicher oder vom Speicher zum Register vor sich geht.

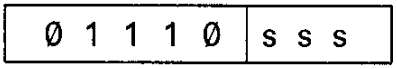
MOV reg1, reg2 – Übertragen Register zu Register

Der Befehl kopiert den Inhalt des Registers 2 (Quelle) in das Register 1 (Ziel). Wenn für beide Operanden dasselbe Register festgelegt wird, (wie in MOVA,A) ist die Wirkung des MOV-Befehls ein NOP-Befehl (Leeroperation). Dieses Format des MOV-Befehls erfordert einen Operationsschritt mehr als NOP und hat deshalb eine etwas längere Ausführungszeit.

Operation:	(reg 2) → reg 1
Maschinencode:	
Operationszyklen:	1
Operationsschritte:	5
Adressierung:	Register (ddd = Zielregister; sss = Quellregister)
Veränderte Bedingungsbits:	keine

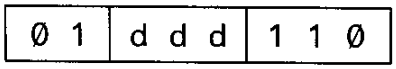
MOV M, reg – Übertragen Register zum Speicher

Dieser Befehl kopiert den Inhalt des festgelegten Registers in die Speicherstelle, die durch die Register H und L adressiert ist.

Operation:	(reg) → (H, L)
Maschinencode:	
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt (sss = Quellregister)
Veränderte Bedingungsbits:	keine

MOV reg, M – Übertragen Speicher zum Register

Dieser Befehl kopiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert ist, in das festgelegte Register.

Operation:	((H, L)) → reg
Maschinencode:	
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt (ddd = Zielregister)
Veränderte Bedingungsbits:	keine

Beispiel

```
MOV ← A, M      ; Laden des Akkumulators aus dem Speicher
MOV ← E, A      ; Kopieren des Akkumulators in das Register E
MOV ← C, C      ; Leeroperation
```

MVI – Übertragen einer Konstanten zum Register oder Speicher

Der Befehl MVI überträgt die in seinem zweiten Byte angegebenen Daten in das spezifizierte Register bzw. die adressierte Speicherstelle.

MVI reg, daten – Übertragen einer Konstanten zum Register

Die in Byte 2 des Befehls angegebene Konstante wird in das spezifizierte Register übertragen.

Operation: (byte 2) → reg

Maschinencode:

Ø Ø	d d d	1 1 Ø
daten		

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar (ddd = Zielregister)
Veränderte Bedingungsbits: keine

MVI M, daten – Übertragen einer Konstanten zum Speicher

Dieser Befehl kopiert die in seinem zweiten Byte gespeicherten Daten in die Speicherstelle, die durch die Register H und L adressiert ist.

Operation: (byte 2) → (H, L)

Maschinencode:

Ø Ø 1 1 Ø 1 1 Ø
daten

Operationszyklen: 3
Operationsschritte: 10
Adressierung: unmittelbar/Register, indirekt
Veränderte Bedingungsbits: keine

NOP – Leerbefehl

Der Befehl NOP führt keine Operation aus und beeinflußt keines der Bedingungsbits.

Operation:	keine
Maschinencode:	Ø Ø Ø Ø Ø Ø Ø Ø
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	keine

Beispiel

Der NOP-Befehl wird häufig verwendet, um Zeitschleifen zu verlängern oder um die Durchlaufzeit von verschiedenen Zweigen eines Verteilers gleich zu machen.

ORA – ODERieren eines Registers oder Speicherbytes mit dem Akkumulator

Der Befehl ORA führt eine logische ODER-Funktion aus und verwendet dazu die Inhalte des festgelegten Bytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert.

Für eine Zusammenfassung der logischen Befehle siehe ANA-Befehl.

ORA reg – ODERieren Register mit Akkumulator

Dieser Befehl ODERiert den Inhalt des festgelegten Registers mit dem Akkumulator. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Operation:	(A) ∨ (reg) → A
Maschinencode:	1 Ø 1 1 Ø | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit

ORA M – ODERieren Speicherbyte mit Akkumulator

Dieser Befehl ODERiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, mit dem Akkumulator. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Operation:	(A) ∨ ((H, L)) → A
Maschinencode:	1 Ø 1 1 Ø 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Da jedes mit 1 inklusiv ODERierte Bit eine 1 erzeugt und jedes mit ØODERierte Bit unverändert bleibt, wird der Befehl ORA oft dazu benutzt, bestimmte Bitgruppen auf 1 zu setzen. Das folgende Beispiel stellt sicher, daß das Bit 2^3 des Akkumulators auf 1 gesetzt wird, die restlichen Bits aber unverändert bleiben. Dies wird häufig getan, wenn in einem Programm einzelne Bits als Zustandskennzeichen verwendet werden. Angenommen, das Register D enthält den Wert Ø8H, der Befehl ORA D hat dann folgende Wirkung:

```
Akkumulator = Ø 1 Ø Ø Ø Ø 1 1
Register D  = Ø Ø Ø Ø 1 Ø Ø Ø
              -----------------
Akkumulator = Ø 1 Ø Ø 1 Ø 1 1
```

ORI – ODERieren einer Konstanten mit dem Akkumulator

Der Befehl ORI führt eine logische ODER-Operation aus und verwendet dazu die Inhalte des zweiten Befehlsbytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Operation: (A) $\vee$ (byte 2) $\rightarrow$ A

Maschinencode:

1 1 1 1 Ø 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Für ein Anwendungsbeispiel des inklusiven ODERs wird auf die Beschreibung des ORA-Befehls verwiesen.

OUT – Ausgeben auf Kanal

Der Befehl OUT legt den Inhalt des Akkumulators auf den 8-Bit-Datenbus und die Nummer des ausgewählten Kanals auf den 16-Bit-Adreßbus. Da die Kanalnummer von Ø bis 255 gehen, wird die Kanalnummer auf dem Adreßbus dupliziert. Die externe Logik ist dafür verantwortlich, die Kanalnummer zu dekodieren und die Ausgabedaten anzunehmen.

Operation: (A) → port

Maschinencode:

1 1 Ø 1 Ø Ø 1 1
port

Operationszyklen: 3
Operationsschritte: 10
Adressierung: direkt
Veränderte Bedingungsbits: keine

PCHL – Laden des Befehlszählers aus Registerpaar H & L

Der Befehl PCHL lädt den Inhalt der Register H und L in den Befehlszähler. Da der Prozessor den nächsten Befehl aus der neuesten Adresse des Befehlszählers entnimmt, wirkt PCHL wie ein Sprungbefehl.

Der Befehl PCHL überträgt den Inhalt des H-Registers zu den höherwertigen acht Bits des Befehlszählers und den Inhalt des L-Registers zu dessen niederwertigen acht Bits.

Operation:	(H) → PCH (L) → PCL
Maschinencode:	1 1 1 Ø 1 Ø Ø 1
Operationszyklen:	1
Operationsschritte:	5
Adressierung:	Register
Veränderte Bedingungsbits:	keine

Beispiel

Eine Technik zur Übergabe von Daten an ein Unterprogramm besteht darin, daß man die Daten unmittelbar hinter dem Unterprogrammaufruf ablegt. Die Rücksprungadresse, die vom CALL-Befehl in den Stack gebracht wird, adressiert augenblicklich die Daten und nicht den nächsten Befehl hinter dem CALL.

Zum Beispiel sei angenommen, dem Unterprogrammaufruf folgen zwei Datenbytes. Dann führt die nachstehende Befehlsfolge einen Rücksprung auf den nächsten Befehl hinter dem Aufruf durch:

```
GOBACK: POP H     ; Holen der Datenadresse
        INXH      ; Addieren 2, um Rücksprung-
        INXH      ; adresse zu bilden
        PCHL      ; Zurückspringen
```

POP – Datenübertragung aus dem Stack

Der Befehl POP überträgt zwei Datenbytes aus dem Stack zurück und kopiert sie in ein Registerpaar oder im Falle des Programmzustandsworts in den Akkumulator und in die Bedingungsbits.

POP rp – Übertragen der Stackdaten zum Registerpaar

Der Befehl POP kopiert den Inhalt der Speicherstelle, die durch den Stackpointer adressiert wird, in das niederwertige Register des festgelegten Registerpaares. Dann erhöht POP den Stackpointer um eins und kopiert den Inhalt der resultierenden Adresse in das höherwertige Register des Paares. Anschließend inkrementiert POP den Stackpointer nochmals, so daß dieser den nächstälteren Eintrag im Stack adressiert.

Als Registerpaar ist B, D, H möglich, jedoch **nicht SP.**

Operation:
((SP)) → rl
((SP)+1) → rh
(SP)+2 → SP

Maschinencode:

1 1	r p	Ø Ø Ø 1

Operationszyklen: 3
Operationsschritte: 10
Adressierung: Register, indirekt (rp=Registerpaar)
Veränderte Bedingungsbits: keine

POP PSW – Übertragen des Programmzustandswortes aus dem Stack

Der Befehl POP PSW verwendet den Inhalt der Speicherstelle, die durch den Stackpointer adressiert wird, um die Bedingungsbits wiederherzustellen. POP PSW erhöht den Stackpointer um eins und stellt aus dem Inhalt dieser Adresse den Akkumulator wieder her. Anschließend inkrementiert POP PSW den Stackpointer nochmals, so daß dieser den nächstälteren Eintrag im Stack adressiert.

Operation:
$((SP))_{Ø} \rightarrow CY$ $((SP))_2 \rightarrow P$
$((SP))_4 \rightarrow AC$ $((SP))_6 \rightarrow Z$
$((SP))_7 \rightarrow S$
((SP)+1) → A
(SP)+2 → SP

Maschinencode:

1 1 1 1 Ø Ø Ø 1

Operationszyklen: 3
Operationsschritte: 10
Adressierung: Register, indirekt
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, ein Unterprogramm wird aufgrund einer externen Unterbrechung aufgerufen. Im allgemeinen sollte ein solches Unterprogramm alle benutzten Register sicherstellen und auch wiederherstellen, damit das Hauptprogramm normal fortfahren kann, sobald es die Steuerung zurückerhalten hat. Die nachstehende Folge von PUSH- und POP-Befehlen stellt das Programmzustandswort und alle Register sicher und stellt sie nach Ablauf des Unterprogramms wieder zur Verfügung.

```
PUSH   PSW
PUSH   B
PUSH   D
PUSH   H
.
.
.
Unterprogrammbefehle
.
.
.
POP    H
POP    D
POP    B
POP    PSW
RET
```

Zu beachten ist, daß die Reihenfolge der POP-Befehle umgekehrt zur PUSH-Reihenfolge ist.

PUSH – Datenübertragung in den Stack

Der Befehl PUSH kopiert zwei Datenbytes in den Stack. Diese können der Inhalt eines Registerpaares oder das Programmzustandswort sein.

PUSH rp – Übertragen des Registerpaares zum Stack

Der Befehl vermindert den Stackpointer um eins und kopiert den Inhalt des höherwertigen Registers des Registerpaares in die resultierende Adresse. Dann dekrementiert PUSH den Stackpointer nochmals und kopiert den Inhalt des niederwertigen Registers in die resultierende Adresse. Die Ursprungsregister bleiben unverändert.

Als Registerpaar ist B, D, H möglich, jedoch **nicht SP.**

Operation: (rh) → (SP) – 1
(rl) → (SP) – 2
(SP) – 2 → SP

Maschinencode:

1	1	r	p	Ø	1	Ø	1

Operationszyklen: 3
Operationsschritte: 11
Adressierung: Register, indirekt (rp = Registerpaar)
Veränderte Bedingungsbits: keine

PUSH PSW – Übertragen des Programmzustandswortes zum Stack

Der Befehl PUSH PSW kopiert das Programmzustandswort in den Stack. Das Programmzustandswort umfaßt den Akkumulatorinhalt und die Bedingungsbits. Da es nur fünf Bedingungsbits gibt, teilt PUSH PSW die Bedingungsbits in einem 8-Bit-Byte wie folgt auf:

Bit	7	6	5	4	3	2	1	Ø
	S	Z	Ø	AC	Ø	P	1	CY

Beim SAB 8080A sind die Bits 3 und 5 immer null; Bit 1 ist immer auf 1 gesetzt.

Der Befehl PUSH PSW vermindert den Stackpointer um eins und kopiert den Akkumulatorinhalt in die resultierende Adresse. Dann dekrementiert PUSH PSW den Stackpointer nochmals und kopiert das Byte mit den formatierten Bedingungsbits in die resultierende Adresse. Der Akkumulatorinhalt und die Bedingungsbits bleiben unverändert.

Operation:

$(A) \rightarrow (SP) - 1$
$(CY) \rightarrow [(SP) - 2]_{\emptyset}$ $\quad 1 \rightarrow [(SP) - 2]_1$
$(P) \rightarrow [(SP) - 2]_2$ $\quad \emptyset \rightarrow [(SP) - 2]_3$
$(AC) \rightarrow [(SP) - 2]_4$ $\quad \emptyset \rightarrow [(SP) - 2]_5$
$(Z) \rightarrow [(SP) - 2]_6$ $\quad (S) \rightarrow [(SP) - 2]_7$
$(SP) - 2 \rightarrow SP$

Maschinencode:

1	1	1	1	Ø	1	Ø	1

Operationszyklen: 3
Operationsschritte: 11
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Siehe Beispiel beim Befehl POP.

RAL – Rotieren des Akkumulators nach links durch das Carrybit

Der Befehl RAL rotiert den Akkumulatorinhalt und das Carrybit um eine Bitposition nach links. Das Carrybit, das wie ein Teil des Akkumulators behandelt wird, schiebt sich in das niederwertige Akkumulatorbit. Das höherwertige Akkumulatorbit schiebt sich in das Carrybit.

Operation: $(A_n) \rightarrow A_{n+1}$
$(A_7) \rightarrow CY$
$(CY) \rightarrow A_0$

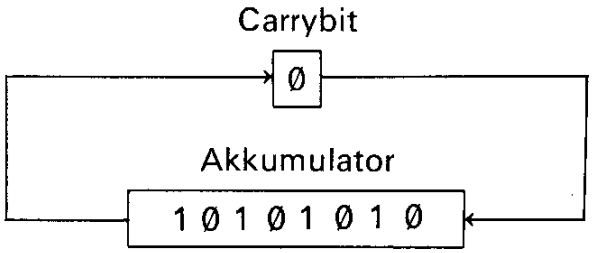
Maschinencode: | 0 0 0 1 0 1 1 1 |

Operationszyklen: 1
Operationsschritte: 4
Veränderte Bedingungsbits: Carrybit

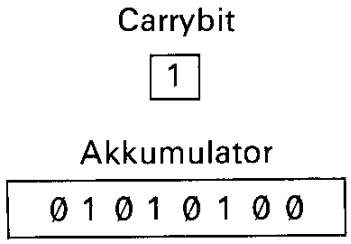
Beispiel

Angenommen, der Akkumulator enthält den Wert AAH und das Carrybit ist null. Das folgende Bild zeigt die Wirkung des RAL-Befehls:

Vorher:



Nachher:



RAR – Rotieren des Akkumulators nach rechts durch das Carrybit

Der Befehl RAR rotiert den Akkumulatorinhalt und das Carrybit um eine Bitposition nach rechts. Das Carrybit, das wie ein Teil des Akkumulators behandelt wird, schiebt sich in das höchstwertige Akkumulatorbit. Das niederwertige Akkumulatorbit schiebt sich in das Carrybit.

Operation: $(A_{n+1}) \rightarrow A_n$
$(A_0) \rightarrow CY$
$(CY) \rightarrow A_7$

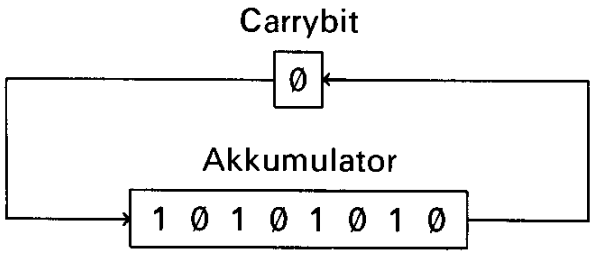
Maschinencode: | 0 0 0 1 1 1 1 1 |

Operationszyklen: 1
Operationsschritte: 4
Veränderte Bedingungsbits: Carrybit

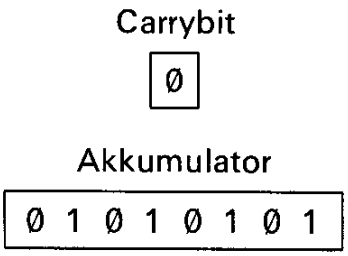
Beispiel

Angenommen, der Akkumulator enthält den Wert AAH und das Carrybit ist null. Das folgende Bild zeigt die Wirkung des RAR-Befehls:

Vorher:



Nachher:



Rcondition – Bedingter Rücksprung aus Unterprogramm

Wenn die spezifizierte Bedingung erfüllt ist, werden die beiden obersten Einträge im Stack in den Befehlszähler übertragen. Der Stackpointer SP wird um 2 erhöht und das Programm an der neuen Adresse fortgesetzt.

Ist die Bedingung nicht erfüllt, wird das Programm mit dem nächstfolgenden Befehl fortgesetzt.

Operation: Wenn Bedingung (CCC) erfüllt:

((SP)) → PCL
((SP)+1) → PCH
(SP)+2 → SP

Maschinencode:

1 1	C C C	Ø Ø Ø

Operationszyklen: 1 oder 3
Operationsschritte: 5 oder 11 *)
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

*) Erste Zahl gilt, wenn Bedingung nicht erfüllt, die zweite, wenn Bedingung erfüllt.

Beispiel

Siehe Beschreibung des RET-Befehls.

RET – Rücksprung aus Unterprogramm

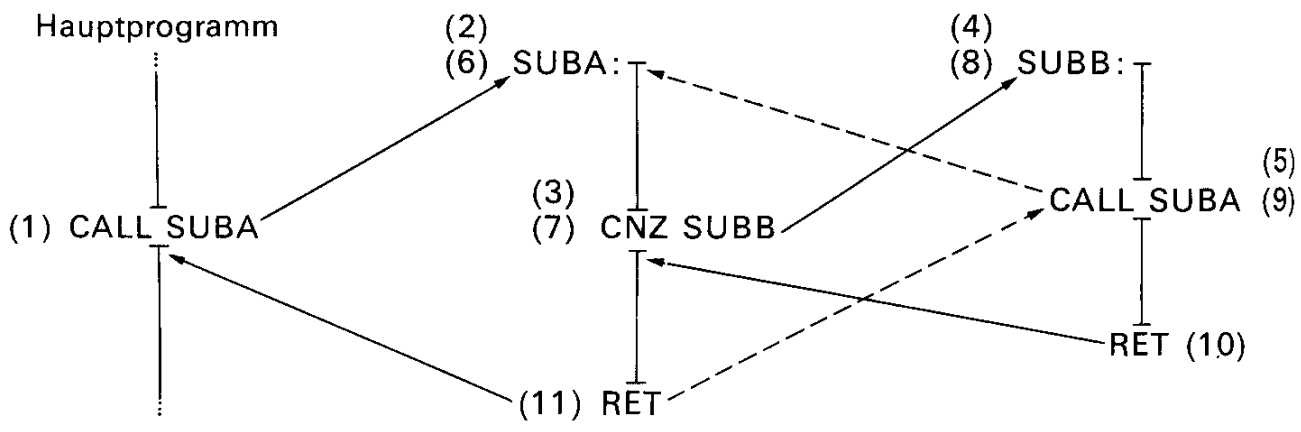
Bei Ausführung des Befehls RET werden die beiden obersten Einträge im Stack in den Befehlszähler übertragen. Der Stackpointer SP wird um 2 erhöht und das Programm an der neuen Adresse fortgesetzt.

Operation:	((SP)) → PCL ((SP)+1) → PCH (SP)+2 → SP
Maschinencode:	1 1 Ø Ø 1 Ø Ø 1
Operationszyklen:	3
Operationsschritte:	10
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	keine

Beispiel

Üblicherweise werden RET-Befehle in Verbindung mit CALL-Befehlen verwendet (dasselbe gilt für die Varianten dieser Befehle). In diesem Falle wird angenommen, daß die vom RET-Befehl aus dem Stack geholten Daten eine Rücksprungadresse sind, die durch einen früheren CALL-Befehl dorthin gebracht worden ist. Dies bewirkt die Rückgabe der Steuerung an den nächsten Befehl hinter dem CALL. Der Anwender muß sicherstellen, daß der RET-Befehl die Adresse eines ausführbaren Befehls im Stack findet. Findet der RET-Befehl die Adresse von Daten, liegt ein schwerwiegender Fehler vor; denn der Prozessor behandelt Daten wie Befehle.

Unterprogramme können geschachtelt werden. Das heißt, daß ein Unterprogramm ein Unterprogramm aufrufen kann, das seinerseits ein anderes Unterprogramm aufruft. Die praktisch einzige Grenze für die Anzahl geschachtelter Aufrufe ist der verfügbare Speicherplatz, um die Rücksprungadressen im Stack unterzubringen. Ein Unterprogramm kann sogar das Unterprogramm aufrufen, von dem es selbst aufgerufen worden ist, wie folgendes Beispiel zeigt. (Zu beachten ist, daß das Programm eine Logik enthalten muß, die gegebenenfalls die Steuerung an das Hauptprogramm zurückgibt. Andernfalls würden sich die beiden Unterprogramme unbegrenzt gegenseitig aufrufen.) Die mit angegebenen Zahlen geben die Reihenfolge an, in der die Befehle abgearbeitet werden. Dabei wird angenommen, daß beide Unterprogramme jeweils zweimal gerufen werden.



RLC – Rotieren des Akkumulators nach links

Der Befehl RLC setzt das Carrybit gleich dem höchstwertigen Akkumulatorbit und überschreibt so den alten Stand. Dann rotiert RLC den Akkumulatorinhalt um eine Bitposition nach links und überträgt dabei das höchstwertige Bit zur niederwertigen Position des Akkumulators.

Operation: $(A_n) \rightarrow A_{n+1}$
$(A_7) \rightarrow A_0$
$(A_7) \rightarrow CY$

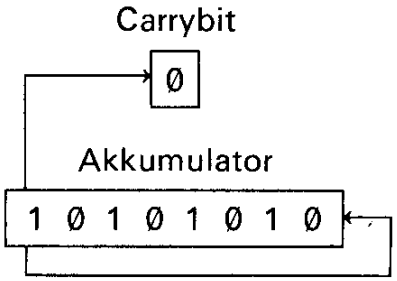
Maschinencode: | 0 0 0 0 0 1 1 1 |

Operationszyklen: 1
Operationsschritte: 4
Veränderte Bedingungsbits: Carrybit

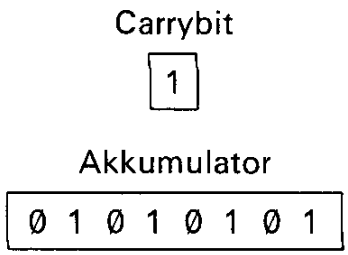
Beispiel

Angenommen, der Akkumulator enthält den Wert AAH und das Carrybit ist null. Das folgende Bild zeigt die Wirkung des RLC-Befehls:

Vorher:



Nachher:



RRC – Rotieren des Akkumulators nach rechts

Der Befehl RRC setzt das Carrybit gleich dem niederwertigen Akkumulatorbit und überschreibt so den alten Inhalt. Dann rotiert RRC den Akkumulatorinhalt um eine Bitposition nach rechts und überträgt dabei das niederwertige Bit zur höchstwertigen Position des Akkumulators.

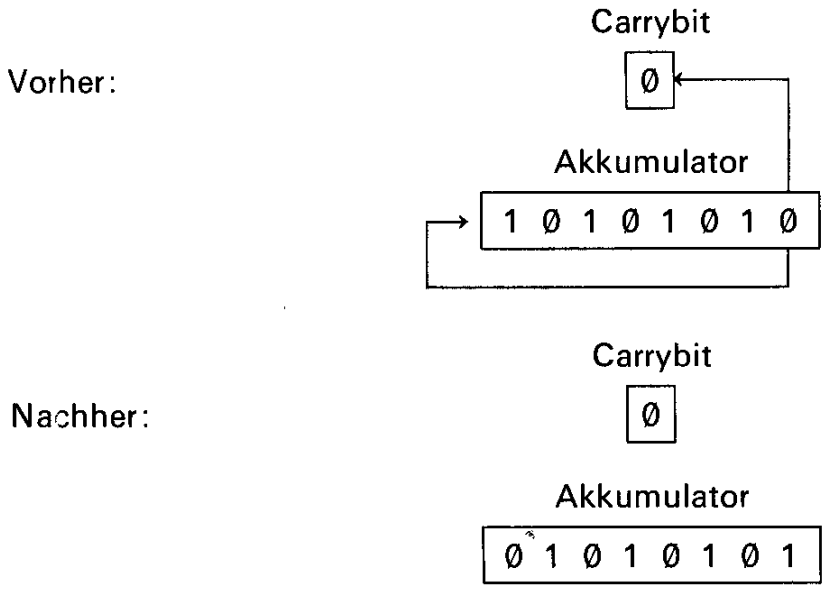
Operation: $(A_{n+1}) \rightarrow A_n$
$(A_\emptyset) \rightarrow A_7$
$(A_\emptyset) \rightarrow CY$

Maschinencode: | 0 0 0 0 1 1 1 1 |

Operationszyklen: 1
Operationsschritte: 4
Veränderte Bedingungsbits: Carrybit

Beispiel

Angenommen, der Akkumulator enthält den Wert AAH und das Carrybit ist null. Das folgende Bild zeigt die Wirkung des RRC-Befehls:



RST – Wiederanlauf

Der Befehl RST ist ein spezieller Unterprogrammaufruf und in erster Linie zur Verwendung bei Unterbrechungen geschaffen. RST stellt den Inhalt des Befehlszählers im Stack sicher, um eine Rücksprung-Adresse zu liefern und springt dann auf eine von acht vorbestimmten Adressen. Ein 3-Bit-Code, der in die Operationscodestellen des RST-Befehls gebracht wird, legt die Sprungadresse fest.

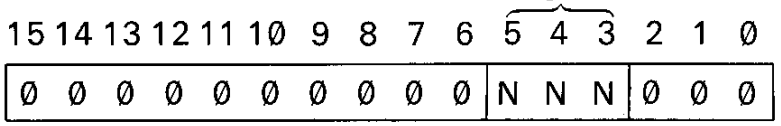
Die höherwertigen 8 Bit des Befehlszählers PCH werden in den Stack auf die Adresse (SP) – 1 gebracht, die niederwertigen 8 Bit PCL auf die Adresse (SP) – 2. Der Inhalt des Stackpointers wird um 2 erniedrigt. Anschließend wird das Programm an der Adresse fortgesetzt, die sich aus dem 8fachen von der Kennzahl NNN im RST-Befehl ergibt.

Operation:

(PCH) → (SP) – 1
(PCL) → (SP) – 2
(SP) – 2 → SP
8*(NNN) → PC

Maschinencode:

Befehlszähler nach RST



Operationszyklen: 3
Operationsschritte: 11
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Der Einsatz des RST-Befehls im Primärcode eines Anwenderprogramms ist selten; viel häufiger senden die Peripheriegeräte, die eine Unterbrechnungsbehandlung verlangen, diesen 1-Byte-Befehl an den Prozessor.

Wenn ein Gerät eine Unterbrechungsbehandlung verlangt und Unterbrechungen freigegeben sind, bestätigt der Prozessor die Aufforderung und bereitet seine Datenkanäle darauf vor, einen beliebigen 1-Byte-Befehl von dem Gerät anzunehmen. Im allgemeinen wird der RST-Befehl gewählt, weil ein spezieller Unterprogrammaufruf einen Rücksprung zum Hauptprogramm durch den RET-Befehl ermöglicht.

Der Prozessor schiebt den 3-Bit-Adressencode aus dem RST-Befehl in die Bits 3, 4 und 5 des Befehlszählers. Dies wirkt sich wie eine Multiplikation mit 8 aus. Die Programmausführung wird nun bei der Adresse fortgesetzt, die sich aus der Multiplikation der RST-Adresse mit 8 ergeben hat.

Bei Verwendung aufeinanderfolgender RST-Befehle steht ein zusammenhängender Speicherbereich von 8 Byte für eine Unterbrechungsbearbeitungsroutine zur Verfügung. Reicht dieser nicht aus, kann auf einfache Weise zu einem Bearbeitungsprogramm gesprungen oder dieses als Unterprogramm aufgerufen werden.

SBB – Subtrahieren eines Registers oder Speicherbytes vom Akkumulator mit Borrow

Der Befehl SBB subtrahiert ein Datenbyte und den Inhalt des Carrybits vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

SBB reg – Subtrahieren eines Registers vom Akkumulator mit Borrow

Dieser Befehl subtrahiert den Inhalt des festgelegten Registers und das Carrybit vom Akkumulator. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) – (reg) – (CY) → A
Maschinencode:	1 Ø Ø 1 1 | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

SBB M – Subtrahieren eines Speicherbytes vom Akkumulator mit Borrow

Dieser Befehl subtrahiert das Carrybit und den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) – ((H, L)) – (CY) → A
Maschinencode:	1 Ø Ø 1 1 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Da der Befehl SBB das Carrybit benutzt, ermöglicht er dem Programm, Zahlenfolgen zu subtrahieren, die sich über mehrere Bytes erstrecken. SBB setzt das Carrybit so ein, daß er es zu dem Byte addiert, das vom Akkumulator subtrahiert werden soll. Das Resultat subtrahiert er dann vom Akkumulator und verwendet dazu die Zweierkomplementarithmetik.

Angenommen, das Register B enthält 2, der Akkumulator enthält 4 und das Carrybit ist auf 1 gesetzt. Der Befehl SBB B arbeitet wie folgt:

```
2H + Carrybit              = 3H
Zweierkomplement von 3H    = 1111 1101

Akkumulator                = 0000 0100
                             1111 1101
                             ---------
                             0000 0001  = 1H
```

Zu beachten ist, daß die Zweierkomplementaddition einen Überlauf erzeugt. Da SBB das durch die Addition entstandene Carrybit negiert, wird es auf Ø zurückgesetzt. Aus dem SBB-Befehl resultieren folgende Bedingungsbits:

```
Carrybit       = Ø
Signbit        = Ø
Zerobit        = Ø
Paritybit      = Ø
Hilfs-Carrybit = 1
```

SBI – Subtrahieren einer Konstanten vom Akkumulator mit Borrow

Der Befehl SBI subtrahiert den Inhalt des zweiten Befehlsbytes und des Carrybits vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation: (A) – (byte 2) – (CY) → A

Maschinencode:

1 1 Ø 1 1 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Siehe SBB-Befehl.

SHLD – Direktes Abspeichern der Register H und L

Der Befehl SHLD überträgt den Inhalt des Registers L auf die Speicherstelle, die durch die Bytes 2 und 3 des SHLD-Befehls festgelegt ist. Der Inhalt des Registers H wird auf die nächsthöhere Speicherstelle übertragen.

Operation: (L) → (byte 3), (byte 2)
(H) → (byte 3), (byte 2) + 1

Maschinencode:

Ø Ø 1 Ø Ø Ø 1 Ø
niederwert. Adreßteil
höherwert. Adreßteil

Operationszyklen: 5
Operationsschritte: 16
Adressierung: direkt
Veränderte Bedingungsbits: keine

Beispiel

Der Befehl SHLD 3ØØØH speichert den Inhalt des L-Registers auf die Speicherstelle mit der Adresse 3ØØØH und den Inhalt des H-Registers auf die Speicherstelle mit der Adresse 3ØØ1H.

SPHL – Laden des Stackpointers mit Register H und L

Der Befehl SPHL lädt den Inhalt der Register H und L in das Stackpointerregister SP. Dabei wird der Inhalt des Registers H in das höherwertige Byte von SP geladen und der Inhalt des Registers L in das niederwertige Byte von SP.

Operation:	(H, L) → SP
Maschinencode:	1 1 1 1 1 0 0 1
Operationszyklen:	1
Operationsschritte:	5
Adressierung:	Register
Veränderte Bedingungsbits:	keine

Beispiel

SP ist ein spezielles 16-Bit-Register, das zur Adressierung des Stack verwendet wird. Der Stack muß im Schreib-/Lesespeicher (RAM) liegen. Da verschiedene Anwendungen verschiedene Speicherzusammenstellungen verwenden, muß das Anwenderprogramm das SP-Register mit der Anfangsadresse des Stack laden. Dem Stack wird üblicherweise die höchste, verfügbare Stelle im RAM zugewiesen. Die Hardware dekrementiert den Stackpointer, wenn Einträge in den Stack gebracht werden und inkrementiert ihn, wenn sie wieder entnommen werden.

Der Stackpointer muß initiiert werden, bevor irgendein Befehl versucht, auf den Stack zuzugreifen. Üblicherweise geschieht die Initiierung des Stackpointers ganz am Anfang des Programms, sobald er einmal eingerichtet ist, sollte er nur mit großer Vorsicht geändert werden. Willkürliche Verwendung des SPHL-Befehls kann zum Verlust von Daten führen.

Zum Laden des Stackpointers mit einer festen Adresse ist der Befehl LXI SP,adr vorzuziehen. Wird die Adresse jedoch erst im Programmlauf bestimmt, so muß der Befehl SPHL verwendet werden.

Angenommen, die Register H und L enthalten 50H bzw. FFH. SPHL lädt den Stackpointer mit dem Wert 50FFH.

STA – Direktes Abspeichern des Akkumulatorinhalts

Der Befehl STA speichert eine Kopie des augenblicklichen Akkumulatorinhalts in die Speicherstelle, die durch die Bytes 2 und 3 des STA-Befehls festgelegt ist.

Operation: (A) → (byte 3), (byte 2)

Maschinencode:

Ø Ø 1 1 Ø Ø 1 Ø
niederwert. Adreßteil
höherwert. Adreßteil

Operationszyklen: 4
Operationsschritte: 13
Adressierung: direkt
Veränderte Bedingungsbits: keine

Beispiel

Der folgende Befehl speichert eine Kopie des Akkumulatorinhalts in die Speicherstelle 5B3H:

STA 5B3H

Im Programmspeicher ist dieser Befehl in aufsteigend adressierten Speicherstellen folgendermaßen abgelegt:

32 B3 Ø5H.

STAX – Indirektes Abspeichern des Akkumulators

Der Befehl STAX speichert eine Kopie des Akkumulatorinhalts in die Speicherstelle, die durch das Registerpaar B oder das Registerpaar D adressiert wird.

Dieser Befehl kann ausschließlich das Registerpaar B oder D festlegen.

Operation: (A) → (r)

Maschinencode:

Ø Ø Ø	r	Ø Ø 1 Ø

r: Ø = Registerpaar B; 1 = Registerpaar D

Operationszyklen: 2
Operationsschritte: 7
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Falls das Register B 3FH und das Register C 16H enthält, speichert der nachstehende Befehl eine Kopie des Akkumulatorinhalts in die Speicherstelle 3F16H ab:

STAX B

STC – Setzen des Carrybits

Der Befehl STC setzt das Carrybit auf 1. Andere Bedingungsbits werden nicht beeinflußt.

Operation:	1 → CY
Maschinencode:	Ø Ø 1 1 Ø 1 1 1
Operationszyklen:	1
Operationsschritte:	4
Veränderte Bedingungsbits:	Carrybit

Beispiel

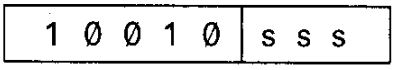
Wird der STC-Befehl gemeinsam mit den Befehlen zum Rotieren des Akkumulators durch Carrybit benutzt, ermöglicht er die Änderung einzelner Bits.

SUB – Subtrahieren eines Registers oder Speicherbytes vom Akkumulator

Der Befehl SUB subtrahiert ein Datenbyte vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert. SUB verwendet die Zweierkomplementdarstellung der Daten, d.h. er führt die Subtraktion durch Addition des Zweierkomplements durch.

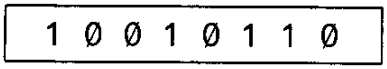
SUB reg – Subtrahieren eines Registers vom Akkumulator

Der Befehl subtrahiert den Inhalt des festgelegten Registers vom Akkumulatorinhalt und verwendet dazu die Zweierkomplementdarstellung. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) – (reg) → A
Maschinencode:	
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

SUB M – Subtrahieren eines Speicherbytes vom Akkumulator

Dieser Befehl subtrahiert den Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.

Operation:	(A) – ((H, L)) → A
Maschinencode:	
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, der Akkumulator enthält 3EH. Der Befehl SUB A subtrahiert den Akkumulatorinhalt vom Akkumulator und erzeugt das Resultat Ø wie folgt:

```
      3EH   = ØØ11 1110
+ (−3EH)    = 1100 ØØØ1   Einerkomplement
                     1    + 1 für Zweierkomplement
Überlauf = 1  ØØØØ ØØØØ   Resultat = Ø
```

Die Bedingungsbits werden wie folgt gesetzt:

Carrybit = Ø
Signbit = Ø
Zerobit = 1
Paritybit = 1
Hilfs-Carrybit = 1

Zu beachten ist, daß der SUB-Befehl den durch die Zweierkomplementaddition erzeugten Überlauf negiert, um ein Borrow-Kennzeichen zu schaffen. Das Hilfs-Carrybit wird auf 1 gesetzt, weil der in diesem Beispiel verwendete Wert einen Überlauf aus Bit 3 gebracht hat.

SUI – Subtrahieren einer Konstanten vom Akkumulator

Der Befehl SUI subtrahiert den Inhalt des zweiten Befehlsbytes vom Akkumulatorinhalt. Das Resultat wird im Akkumulator gespeichert.
Die Subtraktion wird durch Addition des Zweierkomplements durchgeführt.

Operation: (A) – (byte 2) → A

Maschinencode:

1 1 Ø 1 Ø 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, der Akkumulator enthält den Wert 9, wenn der Befehl SUI 1 ausgeführt wird:

Akkumulator	=	ØØØØ 1ØØ1	= 9H
Zweierkomplement der Konstante	=	1111 1111	= –1H
Überlauf = 1		ØØØØ 1ØØØ	= 8H

Zu beachten ist, daß die Zweierkomplementaddition einen Überlauf ergibt. Der SUI-Befehl negiert den bei der Addition erzeugten Überlauf, um ein Borrow-Kennzeichen zu schaffen. Aus der Operation ergeben sich folgende Bedingungsbits:

Carrybit = Ø
Signbit = Ø
Zerobit = Ø
Paritybit = Ø
Hilfs-Carrybit = 1

XCHG – Austauschen der Register H und L mit D und E

Der Befehl XCHG tauscht den Inhalt der Register H und L mit dem der Register D und E aus.

Operation:	(H) ↔ (D) (L) ↔ (E)
Maschinencode:	1 1 1 Ø 1 Ø 1 1
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register
Veränderte Bedingungsbits:	keine

Beispiel

Der Befehl XCHG stellt den augenblicklichen Inhalt der Register H und L sicher und lädt sie mit einer neuen Adresse. Da XCHG ein „Register-zu-Register-Befehl" ist, liefert er den schnellsten Weg, die Register H und L sicherzustellen und/oder zu ändern.

Angenommen, die Register H und L enthalten 1234H und die Register D und E enthalten ABCDH. Nach Ausführung des Befehls XCHG enthalten die Register H und L ABCDH und die Register D und E 1234H.

XRA – Exklusives ODERieren eines Registers oder Speicherbytes mit dem Akkumulator

Der Befehl XRA führt eine logische Exklusiv-ODER-Operation aus und verwendet dazu die Inhalte des festgelegten Bytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert.

Zusammenfassung der logischen Operationen siehe Erklärung bei der Beschreibung des Befehls ANA.

XRA reg – Exklusives ODERieren eines Registers mit dem Akkumulator

Dieser Befehl führt ein exklusives ODER aus und verwendet dazu die Inhalte des festgelegten Registers und des Akkumulators. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Operation:	(A) ⊻ (reg) → A
Maschinencode:	1 Ø 1 Ø 1 | s s s
Operationszyklen:	1
Operationsschritte:	4
Adressierung:	Register (sss = Quellregister)
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

XRA M – Exklusives ODERieren eines Speicherbytes mit dem Akkumulator

Der Inhalt der Speicherstelle, die durch die Register H und L adressiert wird, wird mit dem Akkumulatorinhalt exklusiv ODERiert. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Operation:	(A) ⊻ ((H, L)) → A
Maschinencode:	1 Ø 1 Ø 1 1 1 Ø
Operationszyklen:	2
Operationsschritte:	7
Adressierung:	Register, indirekt
Veränderte Bedingungsbits:	Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Da jedes mit sich selbst exklusiv ODERierte Bit eine Null erzeugt, wird XRA häufig dazu verwendet, den Akkumulator auf Null zu setzen. Die folgenden Befehle setzen den Akkumulator und die Register B und C, sowie das Carrybit auf Null:

```
XRA     A
MOV     B, A
MOV     C, A
```

Jedes, mit einem Einerbit exklusiv ODERierte Bit wird negiert. Falls also der Akkumulator nur Einsen (FFH) enthält, erzeugt der Befehl XRA B das Einerkomplement des B-Registers im Akkumulator.

XRI – Exklusives ODERieren einer Konstanten mit dem Akkumulator

Der Befehl XRI führt eine logische Exklusiv-ODER-Operation aus und verwendet dazu die Inhalte des zweiten Befehlsbytes und des Akkumulators. Das Resultat wird im Akkumulator gespeichert. **Das Carrybit und das Hilfs-Carrybit werden auf Ø zurückgesetzt.**

Eine Zusammenfassung der logischen Operationen ist bei der Beschreibung des Befehls ANA gegeben.

Operation: (A) $\veebar$ (byte 2) $\rightarrow$ A

Maschinencode:

1 1 1 Ø 1 1 1 Ø
daten

Operationszyklen: 2
Operationsschritte: 7
Adressierung: unmittelbar
Veränderte Bedingungsbits: Zero-, Sign-, Parity-, Carry- und Hilfs-Carrybit.

Beispiel

Angenommen, ein Programm verwendet die Bits 7 und 6 eines Bytes als Kennzeichen, um die Aufrufe zweier Unterprogramme zu steuern. Das Programm testet die Bits, indem es den Akkumulatorinhalt rotiert, bis das gewünschte Bit im Carrybit steht; dann testet ein CC-Befehl (Unterprogrammsprung, falls Carrybit=1) das Kennzeichen und ruft das Unterprogramm auf, falls erforderlich.

Angenommen, das Steuerzeichenbyte steht normal im Akkumulator und das Programm muß das Bit 6 auf Ø und das Bit 7 auf 1 setzen. Die übrigen Bits sind Zustandsbits für andere Zwecke und dürfen nicht verändert werden. Da jedes mit 1 exklusiv ODERierte Bit negiert wird und jedes mit Ø exklusiv ODERierte Bit unverändert bleibt, wird der folgende Befehl verwendet:

XRI 1100 0000B

Der Befehl hat folgendes Ergebnis:

Akkumulator = Ø1ØØ 11ØØ
unmittelbare Daten = 11ØØ ØØØØ
1ØØØ 11ØØ

XTHL – Austauschen der Register H und L mit dem Stack

Der Befehl XTHL tauscht den Inhalt des L-Registers mit dem Inhalt der Speicherstelle aus, die durch den Stackpointer SP adressiert wird. Der Inhalt des Registers H wird mit dem Inhalt der Speicherstelle ausgetauscht, die durch den Stackpointer +1 adressiert wird.

Operation: (L) ↔ ((SP))
(H) ↔ ((SP)+1)

Maschinencode: 1 1 1 Ø Ø Ø 1 1

Operationszyklen: 5
Operationsschritte: 18
Adressierung: Register, indirekt
Veränderte Bedingungsbits: keine

Beispiel

Angenommen, der Stackpointer enthält 1ØADH, Register H enthält ØBH, Register L enthält 3CH, die Speicherstelle 1ØADH enthält FØH und die Speicherstelle 1ØAEH enthält ØDH. Das folgende Bild zeigt die Wirkung des XTHL-Befehls:

	Speicheradressen				H	L
	1ØAC	1ØAD	1ØAE	1ØAF		
vor XTHL	FF	FØ	ØD	FF	ØB	3C
nach XTHL	FF	3C	ØB	FF	ØD	FØ

Nach der Ausführung des XTHL-Befehls ist der Stackpointer unverändert.

4.3. Befehlsablauf

In diesem Kapitel werden die einzelnen Operationsschritte, die für den Ablauf eines Befehls vom Prozessor durchzuführen sind, dargestellt. Das Wissen um diese Vorgänge ist für einen Systementwickler nicht zwingend notwendig, *jedoch für das* Verständnis des Prozessors SAB 8080A hilfreich. Für den Ablauf der internen Operationen sowie den Datenfluß innerhalb des Prozessors sei auf Bild 2 verwiesen. Hier einige verwendete Abkürzungen:

PC	=	Befehlszähler
PCL	=	niederwertige 8 Bit des PC
PCH	=	höherwertige 8 Bit des PC
SP	=	Stackpointer
IR	=	Befehlsregister
A	=	Akkumulator
ACT	=	Akkumulator-Zwischenspeicher
TMP	=	Zwischenspeicher-Register
FLAGS	=	Bedingungs-Flipflops
r	=	8-Bit-Register, adressierbar
rp	=	16-Bit-Registerpaar, adressierbar
rl	=	niederwertige 8 Bit eines rp
rh	=	höherwertige 8 Bit eines rp
INST	=	Befehls-Opcode
B2	=	2. Byte eines Befehls
B3	=	3. Byte eines Befehls
JUDGE CONDITION	=	Abfrage der Bedingung

Befehlssatz

Befehlsablauf

MNEMONIC	OP CODE		M1 [1]					M2		
	$D_7 D_6 D_5 D_4$	$D_3 D_2 D_1 D_0$	T1	T2 [2]	T3	T4	T5	T1	T2 [2]	T3
MOV r1, r2	0 1 D D	D S S S	PC OUT STATUS	PC = PC + 1	INST → TMP/IR	(SSS) → TMP	(TMP) → DDD			
MOV r, M	0 1 D D	D 1 1 0				X [3]		HL OUT STATUS [6]	DATA → DDD	
MOV M, r	0 1 1 1	0 S S S				(SSS) → TMP		HL OUT STATUS [7]	(TMP) → DATA BUS	
SPHL	1 1 1 1	1 0 0 1				(HL) → SP				
MVI r, data	0 0 D D	D 1 1 0				X		PC OUT STATUS [6]	PC = PC + 1 B2 → DDD	
MVI M, data	0 0 1 1	0 1 1 0				X			PC = PC + 1 B2 → TMP	
LXI rp, data	0 0 R P	0 0 0 1				X			PC = PC + 1 B2 → r1	
LDA addr	0 0 1 1	1 0 1 0				X			PC = PC + 1 B2 → Z	
STA addr	0 0 1 1	0 0 1 0				X			PC = PC + 1 B2 → Z	
LHLD addr	0 0 1 0	1 0 1 0				X			PC = PC + 1 B2 → Z	
SHLD addr	0 0 1 0	0 0 1 0				X		PC OUT STATUS [6]	PC = PC + 1 B2 → Z	
LDAX rp [4]	0 0 R P	1 0 1 0				X		rp OUT STATUS [6]	DATA → A	
STAX rp [4]	0 0 R P	0 0 1 0				X		rp OUT STATUS [7]	(A) → DATA BUS	
XCHG	1 1 1 0	1 0 1 1				(HL) ↔ (DE)				
ADD r	1 0 0 0	0 S S S				(SSS) → TMP (A) → ACT		[9]	(ACT) + (TMP) → A	
ADD M	1 0 0 0	0 1 1 0				(A) → ACT		HL OUT STATUS [6]	DATA → TMP	
ADI data	1 1 0 0	0 1 1 0				(A) → ACT		PC OUT STATUS [6]	PC = PC + 1 B2 → TMP	
ADC r	1 0 0 0	1 S S S				(SSS) → TMP (A) → ACT		[9]	(ACT) + (TMP) + CY → A	
ADC M	1 0 0 0	1 1 1 0				(A) → ACT		HL OUT STATUS [6]	DATA → TMP	
ACI data	1 1 0 0	1 1 1 0				(A) → ACT		PC OUT STATUS [6]	PC = PC + 1 B2 → TMP	
SUB r	1 0 0 1	0 S S S				(SSS) → TMP (A) → ACT		[9]	(ACT) − (TMP) → A	
SUB M	1 0 0 1	0 1 1 0				(A) → ACT		HL OUT STATUS [6]	DATA → TMP	
SUI data	1 1 0 1	0 1 1 0				(A) → ACT		PC OUT STATUS [6]	PC = PC + 1 B2 → TMP	
SBB r	1 0 0 1	1 S S S				(SSS) → TMP (A) → ACT		[9]	(ACT) − (TMP) − CY → A	
SBB M	1 0 0 1	1 1 1 0				(A) → ACT		HL OUT STATUS [6]	DATA → TMP	
SBI data	1 1 0 1	1 1 1 0				(A) → ACT		PC OUT STATUS [6]	PC = PC + 1 B2 → TMP	
INR r	0 0 D D	D 1 0 0				(DDD) → TMP (TMP) + 1 → ALU	ALU → DDD			
INR M	0 0 1 1	0 1 0 0				X		HL OUT STATUS [6]	DATA → TMP (TMP) + 1 → ALU	
DCR r	0 0 D D	D 1 0 1				(DDD) → TMP (TMP) + 1 → ALU	ALU → DDD			
DCR M	0 0 1 1	0 1 0 1				X		HL OUT STATUS [6]	DATA → TMP (TMP) − 1 → ALU	
INX rp	0 0 R P	0 0 1 1				(RP) + 1 → RP				
DCX rp	0 0 R P	1 0 1 1				(RP) − 1 → RP				
DAD rp [8]	0 0 R P	1 0 0 1				X		(r) → ACT	(L) → TMP (ACT) + (TMP) → ALU	ALU → L, CY
DAA	0 0 1 0	0 1 1 1				DAA → A, FLAGS [10]				
ANA r	1 0 1 0	0 S S S				(SSS) → TMP (A) → ACT		[9]	(ACT) + (TMP) → A	
ANA M	1 0 1 0	0 1 1 0	PC OUT STATUS	PC = PC + 1	INST → TMP/IR	(A) → ACT		HL OUT STATUS [6]	DATA → TMP	

M3			M4			M5				
T1	T2 [2])	T3	T1	T2 [2])	T3	T1	T2 [2])	T3	T4	T5
HL OUT STATUS [7])	(TMP) →	DATA BUS								
PC OUT STATUS [6])	PC = PC + 1 B3 →	rh								
	PC = PC + 1 B3 →	W	WZ OUT STATUS [6])	DATA →	A					
	PC = PC + 1 B3 →	W	WZ OUT STATUS [7])	(A) →	DATA BUS					
	PC = PC + 1 B3 →	W	WZ OUT STATUS [6])	DATA → WZ = WZ + 1	L	WZ OUT STATUS [6])	DATA →	H		
PC OUT STATUS [6])	PC = PC + 1 B3 →	W	WZ OUT STATUS [7])	(L) → WZ = WZ + 1	DATA BUS	WZ OUT STATUS [7])	(H) →	DATA BUS		
[5])	(ACT) + (TMP) → A									
[5])	(ACT) + (TMP) → A									
[5])	(ACT) + (TMP) + CY → A									
[5])	(ACT) + (TMP) + CY → A									
[5])	(ACT) − (TMP) → A									
[5])	(ACT) − (TMP) → A									
[5])	(ACT) − (TMP) − CY → A									
[5])	(ACT) − (TMP) − CY → A									
HL OUT STATUS [7])	ALU →	DATA BUS								
HL OUT STATUS [7])	ALU →	DATA BUS								
(rh) → ACT	(H) → TMP (ACT) + (TMP) + CY → ALU	ALU → H, CY								
[5])	(ACT) + (TMP) → A									

Befehlsablauf (Fortsetzung)

MNEMONIC	OP CODE		M1 [1]					M2		
	$D_7\ D_6\ D_5\ D_4$	$D_3\ D_2\ D_1\ D_0$	T1	T2 [2]	T3	T4	T5	T1	T2 [2]	T3
ANI data	1 1 1 0	0 1 1 0	PC OUT STATUS	PC=PC+1	INST→TMP/IR	(A)→ACT		PC OUT STATUS [6]	PC=PC+1	B2→TMP
XRA r	1 0 1 0	1 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)+(TPM)→A	
XRA M	1 0 1 0	1 1 1 0				(A)→ACT		HL OUT STATUS [6]		DATA→TMP
XRI data	1 1 1 0	1 1 1 0				(A)→ACT		PC OUT STATUS [6]	PC=PC+1	B2→TMP
ORA r	1 0 1 1	0 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)+(TMP)→A	
ORA M	1 0 1 1	0 1 1 0				(A)→ACT		HL OUT STATUS [6]		DATA→TMP
ORI data	1 1 1 1	0 1 1 0				(A)→ACT		PC OUT STATUS [6]	PC=PC+1	B2→TMP
CMP r	1 0 1 1	1 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)−(TMP); FLAGS	
CMP M	1 0 1 1	1 1 1 0				(A)→ACT		HL OUT STATUS [6]		DATA→TMP
CPI data	1 1 1 1	1 1 1 0				(A)→ACT		PC OUT STATUS [6]	PC=PC+1	B2→TMP
RLC	0 0 0 0	0 1 1 1				(A)→ALU ROTATE		[9]	ALU→A, CY	
RRC	0 0 0 0	1 1 1 1				(A)→ALU ROTATE		[9]	ALU→A, CY	
RAL	0 0 0 1	0 1 1 1				(A), CY→ALU ROTATE		[9]	ALU→A, CY	
RAR	0 0 0 1	1 1 1 1				(A), CY→ALU ROTATE		[9]	ALU→A, CY	
CMA	0 0 1 0	1 1 1 1				$(\bar{A})\rightarrow A$				
CMC	0 0 1 1	1 1 1 1				$\overline{CY}\rightarrow CY$				
STC	0 0 1 1	0 1 1 1				1→CY				
JMP addr	1 1 0 0	0 0 1 1				X		PC OUT STATUS [6]	PC=PC+1	B2→Z
J cond addr [17]	1 1 C C	C 0 1 0				JUDGE CONDITION		PC OUT STATUS [6]	PC=PC+1	B2→Z
CALL addr	1 1 0 0	1 1 0 1				SP=SP−1		PC OUT STATUS [6]	PC=PC+1	B2→Z
C cond addr [17]	1 1 C C	C 1 0 0				JUDGE CONDITION IF TRUE, SP=SP−1		PC OUT STATUS [6]	PC=PC+1	B2→Z
RET	1 1 0 0	1 0 0 1				X		SP OUT STATUS [15]	SP=SP+1	DATA→Z
R cond addr [17]	1 1 C C	C 0 0 0			INST→TMP/IR	JUDGE CONDITION [14]		SP OUT STATUS [15]	SP=SP+1	DATA→Z
RST n	1 1 N N	N 1 1 1			0→W INST→TMP/IR	SP=SP−1		SP OUT STATUS [16]	SP=SP−1	(PCH)→DATA BUS
PCHL	1 1 1 0	1 0 0 1			INST→TMP/IR	(HL)→PC				
PUSH rp	1 1 R P	0 1 0 1				SP=SP−1		SP OUT STATUS [16]	SP=SP−1	(rh)→DATA BUS
PUSH PSW	1 1 1 1	0 1 0 1				SP=SP−1		SP OUT STATUS [16]	SP=SP−1	(A)→DATA BUS
POP rp	1 1 R P	0 0 0 1				X		SP OUT STATUS [15]	SP=SP+1	DATA→r1
POP PSW	1 1 1 1	0 0 0 1				X		SP OUT STATUS [15]	SP=SP+1	DATA→FLAGS
XTHL	1 1 1 0	0 0 1 1				X		SP OUT STATUS [15]	SP=SP+1	DATA→Z
IN port	1 1 0 1	1 0 1 1				X		PC OUT STATUS [6]	PC=PC+1	B2→Z, W
OUT port	1 1 0 1	0 0 1 1				X		PC OUT STATUS [6]	PC=PC+1	B2→Z, W
EI	1 1 1 1	1 0 1 1				SET INTE F/F				
DI	1 1 1 1	0 0 1 1				RESET INTE F/F				
HLT	0 1 1 1	0 1 1 0				X		PC OUT STATUS	HALT MODE [20]	
NOP	0 0 0 0	0 0 0 0	PC OUT STATUS	PC=PC+1	INST→TMP/IR	X				

M3			M4			M5					M6	
T1	T2[2])	T3	T1	T2[2])	T3	T1	T2[2])	T3	T4	T5	T1	T2
[9])	(ACT) + (TMP) → A											
[9])	(ACT) + (TMP) → A											
[9])	(ACT) + (TMP) → A											
[9])	(ACT) + (TMP) → A											
[9])	(ACT) + (TMP) → A											
[9])	(ACT) − (TMP); FLAGS											
[9])	(ACT) − (TMP); FLAGS											
PC OUT STATUS[6])	PC = PC + 1 B3	→ W									WZ OUT STATUS[11])	(WZ) + 1 → PC
PC OUT STATUS[6])	PC = PC + 1 B3	→ W									WZ OUT STATUS[11, 12])	(WZ) + 1 → PC
PC OUT STATUS[6])	PC = PC + 1 B3	→ W	SP OUT STATUS[16])	(PCH) → DATA BUS SP = SP − 1		SP OUT STATUS[16])	(PCL) → DATA BUS				WZ OUT STATUS[11])	(WZ) + 1 → PC
PC OUT STATUS[6])	PC = PC + 1 B3	→ W[13])	SP OUT STATUS[16])	(PCH) → DATA BUS SP = SP − 1		SP OUT STATUS[16])	(PCL) → DATA BUS				WZ OUT STATUS[11, 12])	(WZ) + 1 → PC
SP OUT STATUS[15])	SP = SP + 1 DATA	→ W									WZ OUT STATUS[11])	(WZ) + 1 → PC
SP OUT STATUS[15])	SP = SP + 1 DATA	→ W									WZ OUT STATUS[11, 12])	(WZ) + 1 → PC
SP OUT STATUS[16])	(TMP = 00NNN000) (PCL)	→ Z → DATA BUS									WZ OUT STATUS[11])	(WZ) + 1 → PC
SP OUT STATUS[16])	(rl)	→ DATA BUS										
SP OUT STATUS[16])	FLAGS	→ DATA BUS										
SP OUT STATUS[15])	SP = SP + 1 DATA	→ rh										
SP OUT STATUS[15])	SP = SP + 1 DATA	→ A										
SP OUT STATUS[15])	DATA	→ W	SP OUT STATUS[16])	(H) → DATA BUS SP = SP − 1		SP OUT STATUS[16])	(L) → DATA BUS		(WZ)	→ HL		
WZ OUT STATUS[18])	DATA	→ A										
WZ OUT STATUS[18])	(A)	→ DATA BUS										

Anmerkungen

1. Der erste Operationszyklus (M1) ist immer ein Befehlsabruf. Während dieses Zyklus wird das erste (u.U. einzige) Byte, das den Operationscode enthält, aus dem Speicher abgerufen.
2. Wenn der READY-Eingang vom Speicher sich nicht während T2 eines jeden Speicherzyklus auf einem H-Pegel befindet, geht der Prozessor in einen Wartezustand (Tw) über, bis das READY-Signal H-Pegel zeigt.
3. Die Zustände T4 und T5 werden nur bei Bedarf durchlaufen, und zwar nur bei Operationen, die sich vollständig Mikroprozessor-intern abspielen. Der Inhalt des internen Datenbusses ist während T4 und T5 am Datenbus verfügbar, jedoch nur für Prüfzwecke. Ein „X" besagt, daß der betroffene Zustand verwendet wird, jedoch nur für gewisse interne Operationen, wie Befehlsdekodierung.
4. Nur die Registerpaare rp = B (Register B und C) oder rp = D (Register DE) dürfen spezifiziert werden.
5. Diese Zustände werden übersprungen.
6. Speicher-Lese-Zyklen. Ein Befehls- oder Datenwort wird gelesen.
7. Speicher-Schreib-Zyklus.
8. Das READY-Signal wird während des zweiten und dritten Operationszyklus (M 2 und M 3) nicht benötigt. Das HOLD-Signal wird während M 2 und M 3 aufgenommen. Das SYNC-Signal wird während M 2 und M 3 nicht erzeugt. Während der Ausführung eines DAD-Befehls werden M 2 und M 3 für eine interne Registerpaar-Addition benötigt; der Speicher wird nicht angesprochen.
9. Das Ergebnis dieser arithmetischen, logischen oder Schiebebefehle wird vor T2 des nächsten Befehlszyklus nicht zum Akkumulator (A) übertragen. Der Akkumulator wird somit gleichzeitig mit dem nächsten Befehlsabruf geladen. Eine solche Überlappung von Operationen ermöglicht einen schnelleren Ablauf.
10. Wenn der Wert der niederwertigen 4 Bits im Akkumulator größer als 9 ist, oder wenn das Hilfsübertragsbit gesetzt ist, wird die Zahl 6 zum Akkumulator addiert. Ist der Wert der 4 Bits mit der höchsten Wertigkeit im Akkumulator dann größer als 9, oder wenn das Übertragsbit gesetzt ist, wird die Zahl 6 zu den 4 Bits mit der höchsten Wertigkeit addiert.
11. Dieses ist der erste Operationszyklus (der Befehlsabruf) des folgenden Befehlszyklus.
12. Wenn die Bedingung erfüllt wurde, wird anstelle des Inhalts des Befehlszählers (PC) der Inhalt des Registerpaares WZ auf den Adressenleitungen ausgegeben.
13. Wenn die Bedingung nicht erfüllt wurde, werden die Operationszyklen M 4 und M 5 übersprungen. Der Prozessor schreitet in diesem Fall sofort weiter zum Befehlsabruf (M 1) des nächsten Befehlszyklus.
14. Wenn die Bedingung nicht erfüllt wurde, werden die Operationszyklen M 2 und M 3 übersprungen. Der Prozessor geht in diesem Fall sofort zum Befehlsabruf (M 1) des nächsten Befehlszyklus über.

15. Stack-Lese-Operationszyklus.

16. Stack-Schreib-Operationszyklus.

17. Bedingungen

		CCC
NZ – nicht Null (Not Zero)	(Z = Ø)	ØØØ
Z – Null (Zero)	(Z = 1)	ØØ1
NC – kein Übergang (No Carry)	(CY = Ø)	Ø1Ø
C – Übertrag (Carry)	(CY = 1)	Ø11
PO – Parität ungerade (Parity odd)	(P = Ø)	1ØØ
PE – Parität gerade (Parity even)	(P = 1)	1Ø1
P – plus	(S = Ø)	11Ø
M – minus	(S = 1)	111

18. E/A-Zyklus: Die 8-Bit-Adresse des E/A-Bausteins wird auf den Adressenleitungen Ø–7 ($A_{Ø-7}$) und 8–15 (A_{8-15}) dupliziert ausgegeben.

19. Ausgabe-Zyklus.

20. Der Prozessor bleibt im HALT-Zustand untätig, bis er ein Unterbrechungs-, ein Rücksetz- oder ein HOLD[1])-Signal empfängt.
Im Fall eines HOLD[1])-Signals geht der SAB 8080A in den HOLD-Zustand über. Nach dessen Beendigung kehrt der Prozessor in den HALT-Zustand zurück.

Nach dem Empfang eines Rücksetz-Signals beginnt der Prozessor die Programmausführung mit dem Speicherplatz Ø.

Im Anschluß an einen Unterbrechungsbefehl führt der Prozessor den Befehl aus, der in den Datenbus eingegeben worden ist (gewöhnlich ein RESTART-Befehl).

[1]) HOLD bedeutet das Anhalten des SAB 8080A für DMA (direkter Speicherzugriff).

TTL-kompatibel
2 µs Befehlszyklus
Leistungsfähiger Befehlssatz
Sechs Register zur allgemeinen Verwendung und ein Akkumulator
16-Bit-Befehlszähler für die direkte Adressierung einer Speicherkapazität bis zu 64 KBytes
16-Bit-Stackpointer und Stack-Bearbeitungsbefehle, zur schnellen Anpassung an die Programm-Erfordernisse
Dezimale, binäre und doppelt-genaue Arithmetik
Möglichkeit zur Verwendung prioritätsbestimmter Vektor-Unterbrechungen
512 direkt adressierbare E/A-Kanäle

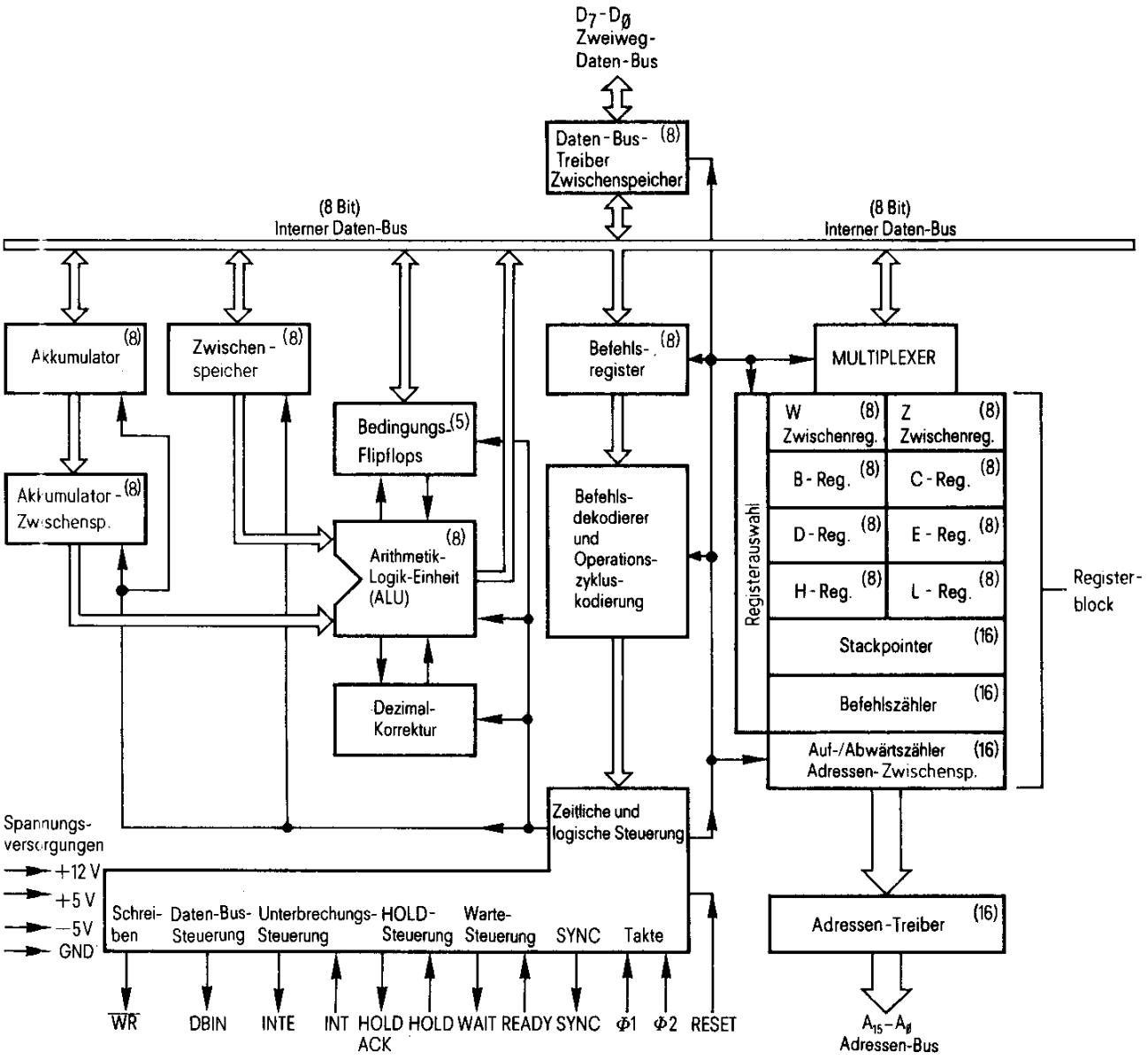
Der SAB 8080A ist ein 8-Bit-paralleler Mikroprozessor. Er wird in n-Kanal-Silizium-Gate-MOS-Technologie auf einem einzigen LSI-Chip hergestellt. Er enthält sechs 8-Bit-Register zur allgemeinen Verwendung und einen Akkumulator; die sechs allgemeinen Register können einzeln oder paarweise adressiert werden und ermöglichen somit entweder einfache oder doppelte Rechen-Genauigkeit. Durch arithmetische und logische Befehle werden vier verschiedene Bedingungsbits gesetzt bzw. rückgesetzt. Ein zusätzliches Bedingungsbit ist für dezimale Arithmetik vorgesehen. Der SAB 8080A kann auf einem externen Stack (Stapelspeicher) zugreifen. Solche externen Speicherbereiche können als „LIFO"-Stack [1]) zur Zwischenspeicherung von Akkumulator, Bedingungsbits, Befehlszähler und sämtlicher Register verwendet werden.

Der 16-Bit-Stackpointer (Stapelspeicher-Zeiger) steuert die Adressierung dieses externen Stack; er ermöglicht eine einfache Handhabung mehrstufiger Prioritätsunterbrechungen, indem er den Zustand des Prozessors ggf. schnell abspeichern und wieder herstellen kann, und eine praktisch unbegrenzte Verschachtelungstiefe von Unterprogrammen.

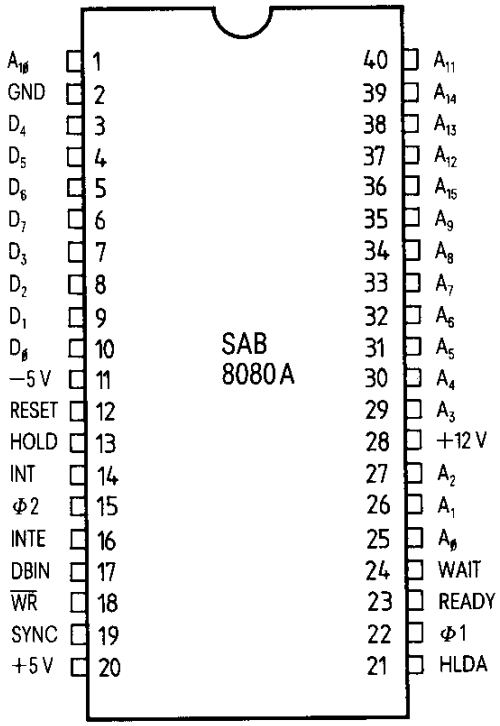
Dieser Mikroprozessor wurde so ausgelegt, daß er die Systementwicklung vereinfacht. Durch den 16-Bit-Adreß-Bus und den getrennten 8-Bit-Zweiweg-Daten-Bus ist eine einfache Anpassung an Speicher- und E/A-Bausteine möglich. Die zur Speicher- und E/A-Anpassung benötigten Steuersignale werden vom SAB 8080A geliefert. Durch ein HOLD-Signal kann der Prozessor in seiner Arbeit unterbrochen und die Adreß- und Daten-Busse in den hochohmigen Zustand gebracht werden. Dies ermöglicht WIRED-OR-Verknüpfung der Busse mit anderen Einheiten, z.B. für Operationen mit direktem Speicherzugriff (DMA) oder Multiprozessor-Operationen.

[1]) LIFO (last in/first out): Die zuletzt eingegebenen Daten werden zuerst wieder ausgegeben.

Funktions-Blockschaltbild



Anschlußbelegung



Abmessungen am Schluß des Buches

Anschlußbezeichnungen

Im folgenden werden die Funktionen aller Gehäuse-Anschlüsse des SAB 8080A beschrieben. Einige Erklärungen beziehen sich auf die internen zeitlichen Abläufe.

A_{15}–$A_{\emptyset}$ (Ausgang mit drei Zuständen)

„Adressen-Bus": Der Adressen-Bus überträgt die Adressen zum Speicher (bis zu 64 KByte a 8-Bit) oder übermittelt die E/A-Adresse für bis zu 256 Eingabe- und 256 Ausgabebausteinen. $A_{\emptyset}$ ist das Adreß-Bit mit der niedrigsten Wertigkeit.

D_7–$D_{\emptyset}$ (Eingang/Ausgang mit drei Zuständen)

„Daten-Bus": Über den Daten-Bus erfolgt der Informationsaustausch zwischen dem Mikroprozessor, Speicher und E/A-Einheiten (Befehle und Daten). Darüber hinaus gibt der SAB 8080A während der ersten Taktperiode eines jeden Befehlszyklus ein Zustandswort mit Informationen über den jeweiligen Zyklus an den Daten-Bus aus. $D_{\emptyset}$ ist das Bit mit der niedrigsten Wertigkeit.

SYNC (Ausgang)

„Synchronisations-Signal": Der SYNC-Anschluß liefert ein Signal, das den Anfang eines jeden Operationszyklus anzeigt.

DBIN (Ausgang)

„Daten-Bus-Eingabe": Das DBIN-Signal zeigt den externen Schaltungen an, daß sich der Daten-Bus in einem Eingabe-Zustand befindet. Dieses Signal wird verwendet, Daten vom Speicher oder von E/A-Einheiten auf den SAB 8080A Daten-Bus zu schleusen.

READY (Eingang)

„Bereit": Das Bereit-Signal zeigt dem SAB 8080A an, daß auf seinem Daten-Bus Speicher- oder Eingabedaten anstehen. Dieses Signal wird dazu verwendet, den Mikroprozessor mit langsameren Speicher- oder E/A-Einheiten zu synchronisieren. Wenn der SAB 8080A nach dem Aussenden einer Adresse nicht sofort ein Bereit-Signal erhält, geht er in einen Wartezustand über, der so lange dauert, wie die Bereit-Leitung auf einem L-Pegel verbleibt.

Das Bereit-Signal kann auch dazu benutzt werden, den Ablauf im Mikroprozessor jeweils um einen einzelnen Operationszyklus weiterzubewegen.

WAIT (Ausgang)

„Warten": Das Warte-Signal bestätigt, daß sich der Mikroprozessor im Wartezustand befindet.

$\overline{WR}$ (Ausgang)

„Schreiben": Das $\overline{WR}$-Signal wird für den Speicher-Schreibvorgang oder zur Ausgabe Steuerung verwendet. Während das $\overline{WR}$-Signal „Low" ist ($\overline{WR}=Ø$), stehen auf dem Daten-Bus Daten an.

HOLD (Eingang)

„Halten": Das HOLD-Signal ist eine Anforderung an den SAB 8080A, in den HOLD-Zustand überzugehen. Dieser Zustand ermöglicht es einer externen Schaltung, die Kontrolle über Adreß- und Daten-Bus zu übernehmen, sobald der SAB 8080A seinen Verkehr mit den Bussen innerhalb des laufenden Befehlszyklus beendet hat. Das HOLD-Signal wird unter den folgenden Bedingungen akzeptiert:

- **Der Mikroprozessor befindet sich im HALT-Zustand**
- **Der Mikroprozessor befindet sich im Operationsschritt T2 oder TW und das Bereit-Signal ist aktiv**

Beim Übergang in den HOLD-Zustand gehen der Adressen-Bus (A_{15}–$A_Ø$) und Daten-Bus (D_7–$D_Ø$) des Mikroprozessor in den hochohmigen Zustand über. Der Mikroprozessor bestätigt seinen HOLD-Zustand am HLDA-(HOLD-Quittungs-)Anschluß.

HLDA (Ausgang)

„Quittierung der HOLD-Anforderung": Das HLDA-Signal erscheint als Antwort auf ein HOLD-Signal. Es zeigt an, daß Daten- und Adreß-Bus in den hochohmigen Zustand gehen. Das HLDA-Signal beginnt:

- zum Zeitpunkt T3 bei einem Speicher-Lesevorgang oder einer Eingabe
- mit der Taktperiode, die auf T3 folgt, bei einer Speicher-Schreib- oder Ausgabe-Operation.

In jedem Fall tritt das HLDA-Signal nach der ansteigenden Flanke von $\Phi 1$ auf, die Hochohmigkeit erfolgt nach der Anstiegsflanke von $\Phi 2$.

INTE (Ausgang)

„Unterbrechungs-Freigabe": Dieses Signal zeigt den Zustand des internen Unterbrechungs-Freigabe-Flipflops an. Dieses Flipflop kann durch die Befehle zur Freigabe oder zum Sperren von Unterbrechungen gesetzt bzw. zurückgesetzt werden. Es verhindert im rückgesetzten Zustand die Annahme von Unterbrechungen (INTERRUPTS) durch den Mikroprozessor. Das Flipflop wird automatisch rückgesetzt (und verhindert damit Unterbrechungen) während eines T1-Abrufzyklus (M1), wenn ein Unterbrechungsbefehl bereits angenommen wurde. Es wird auch durch ein RESET-Signal am Rückstelleingang des SAB 8080A rückgesetzt.

INT (Eingang)

„Unterbrechungs-Anforderung": Der SAB 8080A akzeptiert eine Unterbrechungs-Anforderung auf dieser Leitung entweder am Ende eines laufenden Befehlszyklus oder während eines HALT-Zustands. Wenn sich der Mikroprozessor im HOLD-Zustand befindet, oder wenn das Unterbrechungs-Freigabe-Flipflop rückgesetzt ist, wird die Unterbrechungs-Anforderung nicht angenommen.

RESET [1]) (Eingang)

„Rücksetzen": Durch ein Rücksetz-Signal wird der Inhalt des Befehlszählers auf Null gesetzt. Nach einem Rücksetz-Signal beginnt das Programm infolgedessen mit der Speicherstelle Null. Die Flipflops INTE und HLDA werden ebenfalls rückgesetzt. Hingegen ist zu beachten, daß Bedingungs-Flipflop, Akkumulator, Stackpointer und die Register nicht rückgesetzt werden.

V_{SS}: Bezugspotential (Masse)
V_{DD}: $+12\,V \pm 5\%$
V_{CC}: $+5\,V \pm 5\%$
V_{BB}: $-5\,V \pm 5\%$ (Substratvorspannung)
$\Phi 1$, $\Phi 2$: Zwei von außerhalb zugeführte Takte (nicht TTL-kompatibel)

---

[1]) Das Rücksetz-Signal muß für die Dauer von mindestens drei Taktzyklen anliegen.

Grenzdaten [1])

Betriebstemperatur	0	bis	+70 °C
Lagertemperatur	−65	bis	+150 °C
Alle Eingangs- oder Ausgangsspannungen (bezogen auf V_{BB})	−0,3	bis	+20 V
V_{CC}, V_{DD} und V_{SS} (bezogen auf V_{BB})	−0,3	bis	+20 V
Verlustleistung	1,5 W		

Statische Kenndaten und Betriebsbedingungen

T_U = 0 bis +70 °C, V_{DD} = +12 V ± 5%, V_{CC} = +5 V ± 5%, V_{BB} = −5 V ± 5%, V_{SS} = 0 V (wenn nicht anders angegeben)

Symbol	Bezeichnung	Grenzwerte min.	Grenzwerte typ.[2])	Grenzwerte max.	Einheit	Prüfbedingung
V_{ILC}	Takt-L-Eingangsspannung	$V_{SS}-1$		$V_{SS}+0{,}8$		
V_{IHC}	Takt-H-Eingangsspannung	9		$V_{DD}+1$		–
V_{IL}	L-Eingangsspannung	$V_{SS}-1$		$V_{SS}+0{,}8$		
V_{IH}	H-Eingangsspannung	3,3	–	$V_{CC}+1$	V	
V_{OL}	L-Ausgangsspannung	–		0,45		I_{OL} = 1,9 mA alle Ausgänge;
V_{OH}	H-Ausgangsspannung	3,7		–		I_{OH} = −150 µA
$I_{DD(AV)}$	Mittl. Stromaufnahme (V_{DD})		40	70		
$I_{CC(AV)}$	Mittl. Stromaufnahme (V_{CC})		60	80	mA	in Betrieb t_{CY} = 0,48 µs
$I_{BB(AV)}$	Mittl. Stromaufnahme (V_{BB})		0,01	1		
I_{IL}	Eingangs-Reststrom			±10	µA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{CL}	Takteingangs-Reststrom	–				$V_{SS} \le V_{CLOCK} \le V_{DD}$
I_{DL} [3])	Daten-Bus-Reststrom (Eingabezustand)		–	−100	µA	$V_{SS} \le V_{IN} \le V_{SS}+0{,}8$ V
				−2	mA	$V_{SS}+0{,}8$ V $\le V_{IN} \le V_{CC}$
I_{FL}	Adress- und Daten-Bus-Reststrom (HOLD-Zustand)			+10 −100	µA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS}+0{,}45$ V

[1]) Die Überschreitung der angegebenen Grenzdaten kann zu Dauerschäden am Baustein führen.
[2]) Typische Werte gelten für T_U = 25 °C und Nenn-Versorgungsspannung.
[3]) Wenn das DBIN-Signal „H" ist und $V_{IN} > V_{IL}$, wird eine interne Vorspannung auf den Datenbus gegeben.

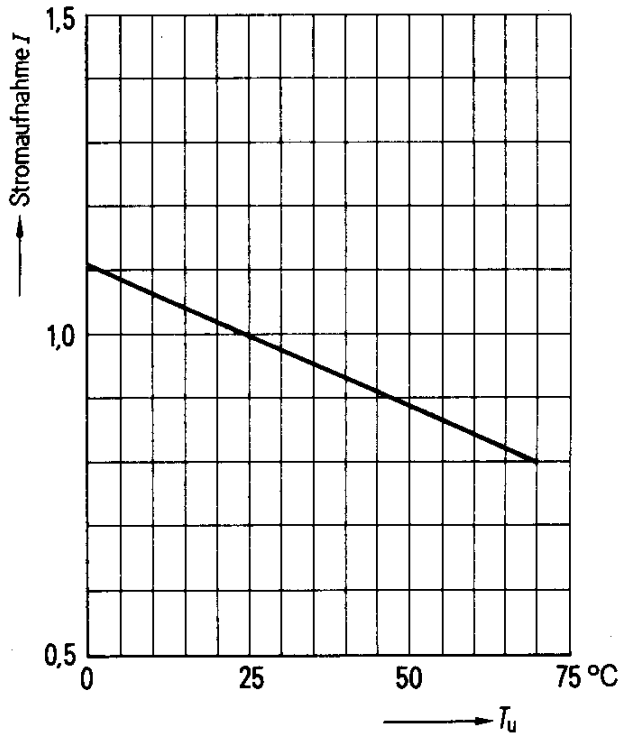
Kapazitäten[1])

$T_U = 25\,°C$, $V_{CC} = V_{DD} = V_{SS} = 0\,V$, $V_{BB} = -5\,V$

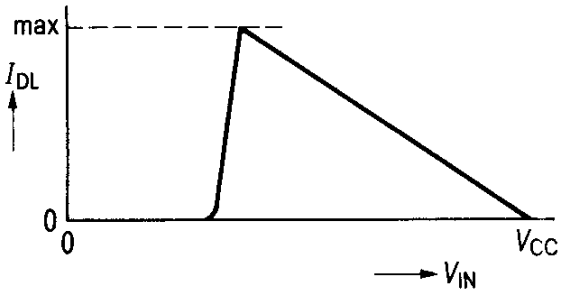
Symbol	Bezeichnung	Grenzwerte typ.	Grenzwerte max.	Einheit	Prüfbedingung
C_Φ	Takteingangs-Kapazität	17	25	pF	$f_c = 1$ MHz
C_{IN}	Eingangs-Kapazität	6	10		nicht gemessene Anschlüsse verbunden mit V_{SS}
C_{OUT}	Ausgangs-Kapazität	10	20		

[1]) Dieser Parameter wird nur stichprobenmäßig kontrolliert und nicht zu 100% geprüft.

Typische Stromaufnahme in Abhängigkeit von der Temperatur (normalisiert)*)



Daten-Bus-Kennlinie während „DBIN"

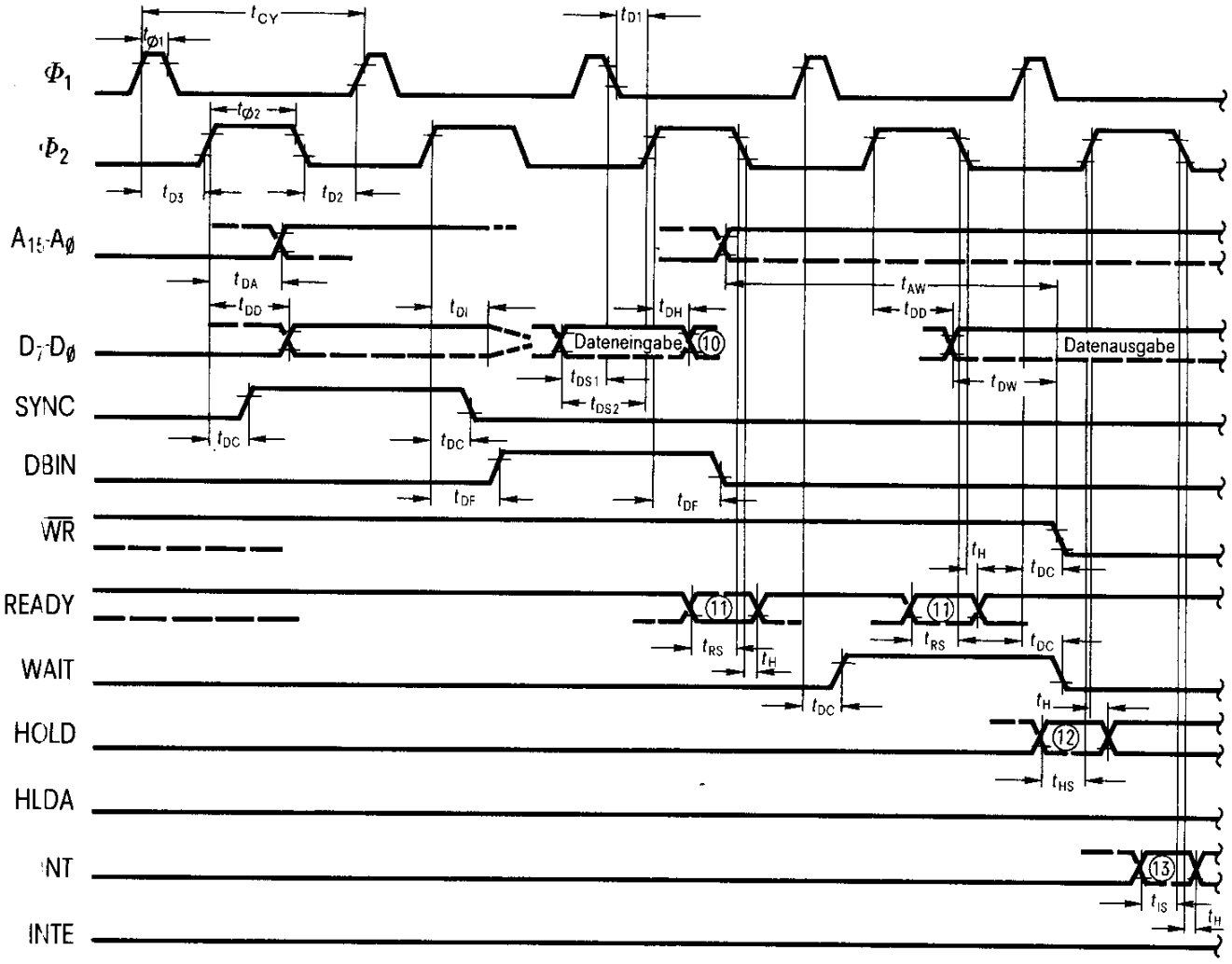


*) Stromaufnahme/Umgebungstemperatur $\Delta I/\Delta T_U = -0{,}45\%/°C$

Impulsdiagramm [1])

Die zeitlichen Messungen sind für die folgenden Bezugsspannungen gültig:

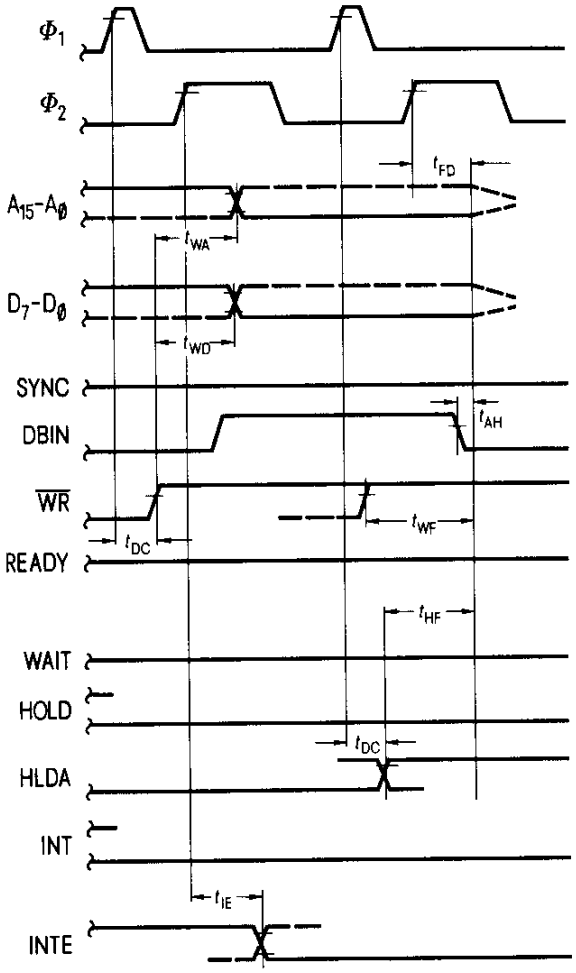
Takt „1" = 8 V; „Ø" = 1 V;
Eingänge „1" = 3,3 V; „Ø" = 0,8 V;
Ausgänge „1" = 2 V; „Ø" = 0,8 V;



Ⓝ -Anmerkungen siehe Seite 158

[1]) Dieses Impuls-Diagramm stellt nur die zeitlichen Relationen von Signalen dar, nicht einen speziellen Operationszyklus.

Impulsdiagramm (Fortsetzung)



Schaltzeiten

$T_U = 0$ bis $+70$ °C, $V_{DD} = +12\ V \pm 5\%$, $V_{CC} = +5\ V \pm 5\%$, $V_{BB} = -5\ V \pm 5\%$, $V_{SS} = 0\ V$, (wenn nicht anders angegeben)

Symbol	Bezeichnung	Grenzwerte		Einheit	Prüfbedingung
		min.	max.		
t_{CY} [3]	Taktperiode	0,48	2	µs	–
t_r, t_f	Takt-Anstiegs- und Abfallzeit	0	50	ns	
$t_{\Phi 1}$	Φ_1-Impulsdauer	60	–		
$t_{\Phi 2}$	Φ_2-Impulsdauer	220	–		
t_{D1}	Verzögerung Φ_1 bis Φ_2	0	–		
t_{D2}	Verzögerung Φ_2 bis Φ_1	70	–		
t_{D3}	Verzögerung Φ_1 bis Φ_2, vordere Flanken	80	–		
t_{DA} [2]	Adressenausgabe-Verzögerung von Φ_2	–	200		$C_L = 100$ pF
t_{DD} [2]	Datenausgabe-Verzögerung von Φ_2	–	220		
t_{DC} [2]	Signalausgabe-Verzögerung von Φ_1 oder Φ_2 (SYNC, $\overline{WR}$, WAIT, HLDA)	–	120		$C_L = 50$ pF
t_{DF} [2]	DBIN-Verzögerung von Φ_2	25	140		
t_{DI} [1]	Eingabezustands-Verzögerung des Daten-Bus	–	t_{DF}		–
t_{DS1}	Daten-Vorbereitungszeit während Φ_1 und DBIN	30	–		

Anmerkungen siehe Seite 158

Schaltzeiten (Fortsetzung)

Symbol	Bezeichnung	Grenzwerte		Einheit	Prüfbedingung
		min.	max.		
t_{DS2}	Daten-Vorbereitungszeit bis Φ_2 während DBIN	150	–	ns	–
t_{DH} [1])	Daten-Haltezeit von Φ_2 während DBIN	[1])	–		
t_{IE} [2])	Verzögerung der Unterbrechungs-Freigabe (INTE) von Φ_2	–	200		$C_L = 50$ pF
t_{RS}	Bereit-(READY-)Vorbereitungszeit während Φ_2	120	–		–
t_{HS}	Anhalte-(HOLD-)Vorbereitungszeit bis Φ_2	140	–		
t_{IS}	Unterbrechungs-(INT-)Vorbereitungszeit während Φ_2 (während Φ_1 im HALT-Zustand)	120	–		
t_H	Haltezeit von Φ_2 (READY, INT, HOLD)	0	–		
t_{FD}	Verzögerung bis hochohmiger Zustand während HOLD (Adressen- und Daten-Bus)	–	120		
t_{AW} [2])	Adresse stabil vor $\overline{WR}$	[5])	–	ns	$C_L = 100$ pF: Adressen, Daten $C_L = 50$ pF: $\overline{WR}$, HLDA, DBIN
t_{DW} [2])	Ausgangs-Daten stabil vor $\overline{WR}$	[6])	–		
t_{WD} [2])	Ausgangs-Daten stabil nach $\overline{WR}$	[7])	–		
t_{WA} [2])	Adresse stabil nach $\overline{WR}$	[7])	–		
t_{HF} [2])	Verzögerung HLDA bis hochohmiger Zustand	[8])	–		
t_{WF} [2])	Verzögerung $\overline{WR}$ bis hochohmiger Zustand	[9])	–		
t_{AH} [2])	Adressen-Haltezeit nach DBIN während HLDA	−20	–		

Anmerkungen siehe Seite 158

Anmerkungen der Seiten 154 bis 157

[1]) Dateneingaben sind mit DBIN-Zustand freizugeben. Auf diese Weise werden Bus-Konflikte vermieden, und die Daten-Haltezeit ist sichergestellt. $t_{DH}=50$ ns oder t_{DF}, das Minimum von beiden.

[2]) Meßschaltung (Ausgangsbelastung, siehe nächste Seite).

[3]) $t_{CY}=t_{D3}+t_{r\Phi2}+t_{\Phi2}+t_{f\Phi2}+t_{D2}+t_{r\Phi1}\geqq 480$ ns.

[4]) Die folgenden Angaben gelten, wenn der SAB 8080A an Bausteine mit $V_{IH}=3{,}3$ V angeschlossen wird.
a) Max. Ausg.-Anstiegszeit von 0,8 V auf 3,3 V = 100 ns bei C_L = spez.
b) Ausg.-Verzögerung gemessen bis 3 V = spez. + 60 ns bei C_L = spez.
c) Wenn $C_L \neq$ spez.: Addiere 0,6 ns/pF, wenn $C_L >$ C spez.; subtrahiere 0,3 ns/pF (vom modifizierten Verzögerungswert), wenn $C_L <$ C spez. (Bild siehe nächste Seite).

[5]) $t_{AW}=2t_{CY}-t_{D3}-t_{r\Phi2}-140$ ns.

[6]) $t_{DW}=t_{CY}-t_{D3}-t_{r\Phi2}-170$ ns.

[7]) Wenn nicht HLDA, $t_{WD}=t_{WA}=t_{D3}+t_{r\Phi2}+10$ ns.
Wenn HLDA, $t_{WD}=t_{WA}=t_{WF}$.

[8]) $t_{HF}=t_{D3}+t_{r\Phi2}-50$ ns.

[9]) $t_{WF}=t_{D3}+t_{r\Phi2}-10$ ns.

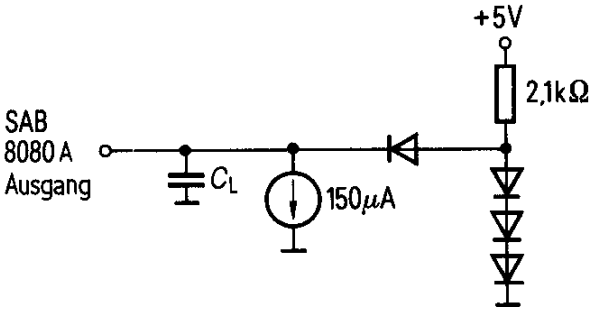
[10]) Eingabedaten müssen für dieses Zeitintervall während DBIN·T_3 stabil sein. Sowohl die t_{DS1}- als auch die t_{DS2}-Forderung muß erfüllt sein.

[11]) Das Bereit-Signal muß für dieses Zeitintervall während T_2 oder T_W stabil sein. (Muß extern synchronisiert werden.)

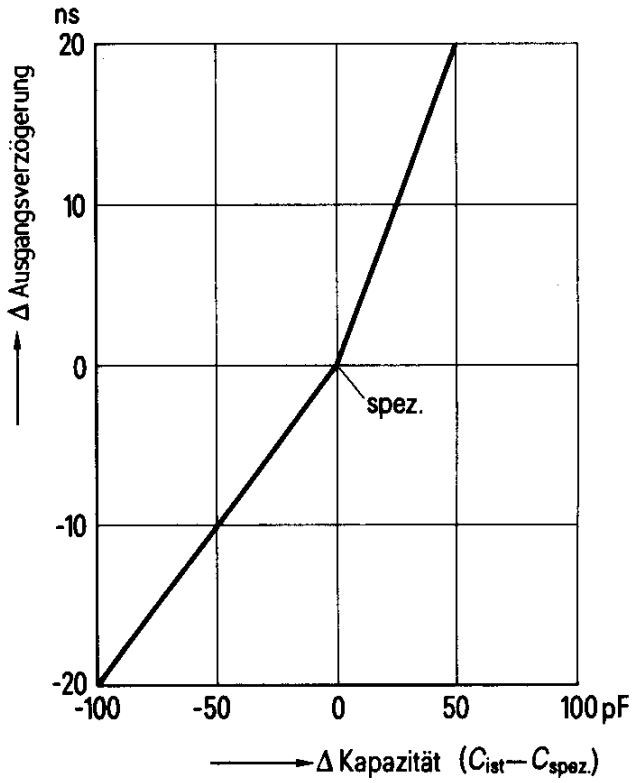
[12]) Das HOLD-Signal muß für dieses Zeitintervall stabil sein, und zwar bei Eintritt in den HOLD-Zustand während T_2 oder T_W und nach Erreichen des HOLD-Zustands während T_3, T_4, T_5 und T_{WH}.
(Externe Synchronisation nicht erforderlich.)

[13]) Das Unterbrechungs-Signal muß während dieses Zeitintervalls im letzten Taktzyklus eines jeden Befehls stabil sein, um im nachfolgenden Befehl erkannt zu werden.
(Externe Synchronisation nicht erforderlich.)

Meßschaltung



Typische Δ Ausgangsverzögerung in Abhängigkeit von der Δ Kapazität



Befehlssatz

Der Befehlssatz des SAB 8080A umfaßt folgende Anweisungen:

Datentransferbefehle zur Datenübertragung zwischen dem Akkumulator, den allgemeinen Registern, RAM-Speicher-Stellen und Ein-/Ausgabe-Bausteinen in direkter, indirekter und unmittelbarer Adressierung.

Arithmetische und logische Operationen ebenfalls mit direkter, indirekter und unmittelbarer Adressierung. Neben den Operationen mit 8-Bit-Argumenten stehen arithmetische Operationen doppelter Genauigkeit (16-Bit-Argumente) und Befehle zur Durchführung von Dezimalarithmetik zur Verfügung.

Befehle zur Programmverzweigung, wie bedingte und unbedingte Sprünge und Unterprogrammsprung.

Sonderbefehle, wie HALT (Anhalten des Prozessors), NOP (Leerbefehl), STC (Setzen des Übertragsbits), CMC (Komplementieren des Übertragsbits), CMA (Komplementieren des Akkumulatorinhalts), XCHG (Austausch zweier Registerpaarinhalte) und Anweisungen zum bitweisen Rotieren des Akkumulatorinhalts.

Daten- und Befehlsformate

Daten im SAB 8080A werden in der Form von 8-Bit binären ganzen Zahlen gespeichert. Alle Datenübertragungen zum Daten-Bus des Systems erfolgen mit dem gleichen Format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Daten-Wort

Die Befehle können 1, 2 oder 3 Bytes lang sein. Mehr-Byte-Befehle müsser an aufeinanderfolgenden Stellen im Programmspeicher gespeichert werden. Die Befehlsformate hängen in diesem Fall von der jeweils auszuführenden Operation ab.

1-Byte-Befehl

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

OP-Code

Typische Befehle

Register/Register-, Speicher-, arithmetische-, logische-, Rotations-, Rückkehr-, Stackeingabe-, Stackausgabe-, Unt.-Freigabe-, Sperren Unt.-Befehle

2-Byte-Befehle

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Op-Code

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Operand oder Adresse

Befehle für unmittelbare Adressierung oder E/A-Operationen

3-Byte-Befehle

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

OP-Code

Befehle zum Springen, Aufrufen eines Unterprogramms oder zum direkten Laden und Speichern

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Niederwertige Adresse oder Operand 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

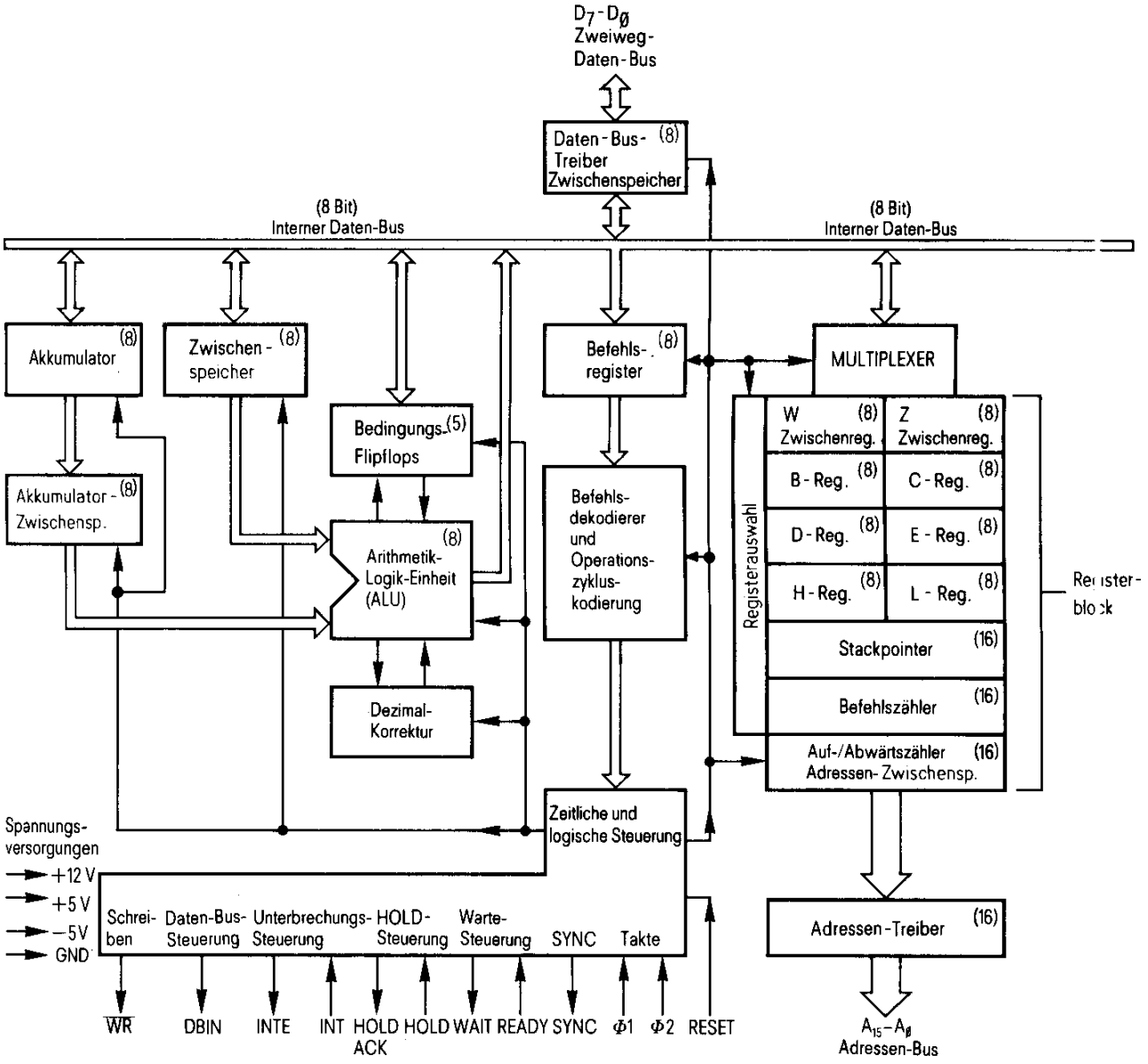
Höherwertige Adresse oder Operand 2

Bemerkung: Im System SAB 8080A entspricht der „logischen 1" der H-Pegel und der „logischen Ø" der L-Pegel.

Befehlszyklus 1,3 µs (SAB 8080A-1)
Befehlszyklus 1,5 µs (SAB 8080A-2)

Funktionsbeschreibung und Anwendung siehe SAB 8080A

Funktions-Blockschaltbild



Grenzdaten [1])

Betriebstemperatur	0 bis + 70 °C
Lagertemperatur	−65 bis +150 °C
Alle Eingangs- oder Ausgangsspannungen (bezogen auf V_{BB})	−0,3 bis + 20 V
V_{CC}, V_{DD} und V_{SS} (bezogen auf V_{BB})	−0,3 bis + 20 V
Verlustleistung	1,5 W

Statische Kenndaten und Betriebsbedingungen

T_U = 0 bis +70 °C; V_{DD} = +12 V ±5%; V_{CC} = +5 V ±5%; V_{BB} = −5 V ±5%; V_{SS} = 0 V (wenn nicht anders angegeben)

Symbol	Bezeichnung	Grenzwerte min.	Grenzwerte typ.[2])	Grenzwerte max.	Einheit	Prüfbedingung
V_{ILC}	Takt-L-Eingangsspannung	$V_{SS}-1$		$V_{SS}+0,8$		
V_{IHC}	Takt-H-Eingangsspannung	9		$V_{DD}+1$		
V_{IL}	L-Eingangsspannung	$V_{SS}-1$	–	$V_{SS}+0,8$	V	–
V_{IH}	H-Eingangsspannung	3,3		$V_{CC}+1$		
V_{OL}	L-Ausgangsspannung	–		0,45		I_{OL} = 1,9 mA alle Ausgänge
V_{OH}	H-Ausgangsspannung	3,7		–		I_{OH} = −150 µA
$I_{DD(AV)}$	Mittl. Stromaufnahme (V_{DD})		40	70		in Betrieb t_{CY} = 0,32 µs (8080A-1) 0,38 µs (8080A-2)
$I_{CC(AV)}$	Mittl. Stromaufnahme (V_{BB})		60	80	mA	
$I_{BB(AV)}$	Mittl. Stromaufnahme (V_{CC})		0,01	1		
I_{IL}	Eingangs-Reststrom	–		±10	µA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{CL}	Takteingangs-Reststrom					$V_{SS} \le V_{CLOCK} \le V_{DD}$
I_{DL} [3])	Daten-Bus-Reststrom (Eingabezustand)		–	−100	µA	$V_{SS} \le V_{IN} \le V_{SS}+0,8$ V
				−2	mA	$V_{SS}+0,8$ V $\le V_{IN} \le V_{CC}$
I_{FL}	Adreß- und Daten-Bus-Reststrom (HOLD-Zustand)			+ 10 −100	µA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS}$ + 0,45 V

[1]) Die Überschreitung der angegebenen Grenzdaten kann zu Dauerschäden am Baustein führen.
[2]) Typische Werte für T_U = 25 °C und Nenn-Versorgungsspannung.
[3]) Wenn das DBIN-Signal „H" ist und $V_{IN} > V_{IL}$, wird eine interne Vorspannung auf den Daten-Bus gegeben.

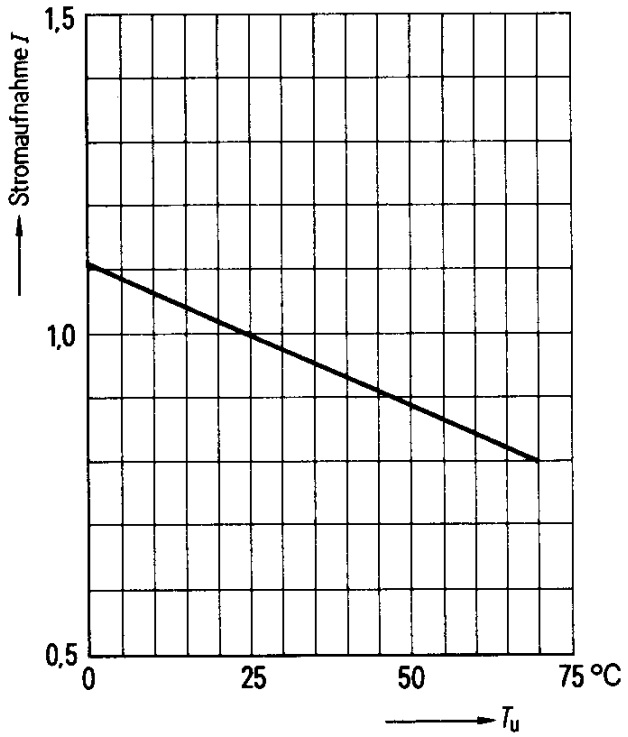
Kapazitäten[1])

$T_U = 25\,°C$; $V_{CC} = V_{DD} = V_{SS} = 0\,V$; $V_{BB} = -5\,V$

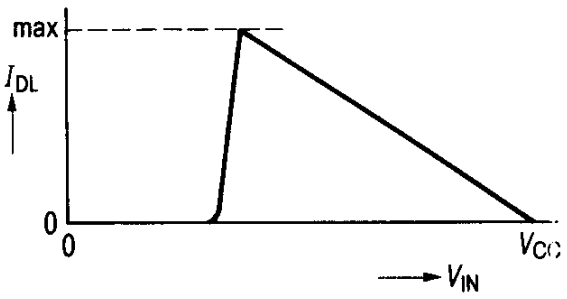
Symbol	Bezeichnung	Grenzwerte		Einheit	Prüfbedingung
		typ	max.		
C_Φ	Takteingangs-Kapazität	17	25	pF	$f_c = 1$ MHz
C_{IN}	Eingangs-Kapazität	6	10		nicht gemessene Anschlüsse verbunden mit V_{SS}
C_{OUT}	Ausgangs-Kapazität	10	20		

[1]) Dieser Parameter wird nur stichprobenmäßig kontrolliert und nicht zu 100% geprüft.

Typische Stromaufnahme in Abhängigkeit von der Temperatur (normalisiert)*)



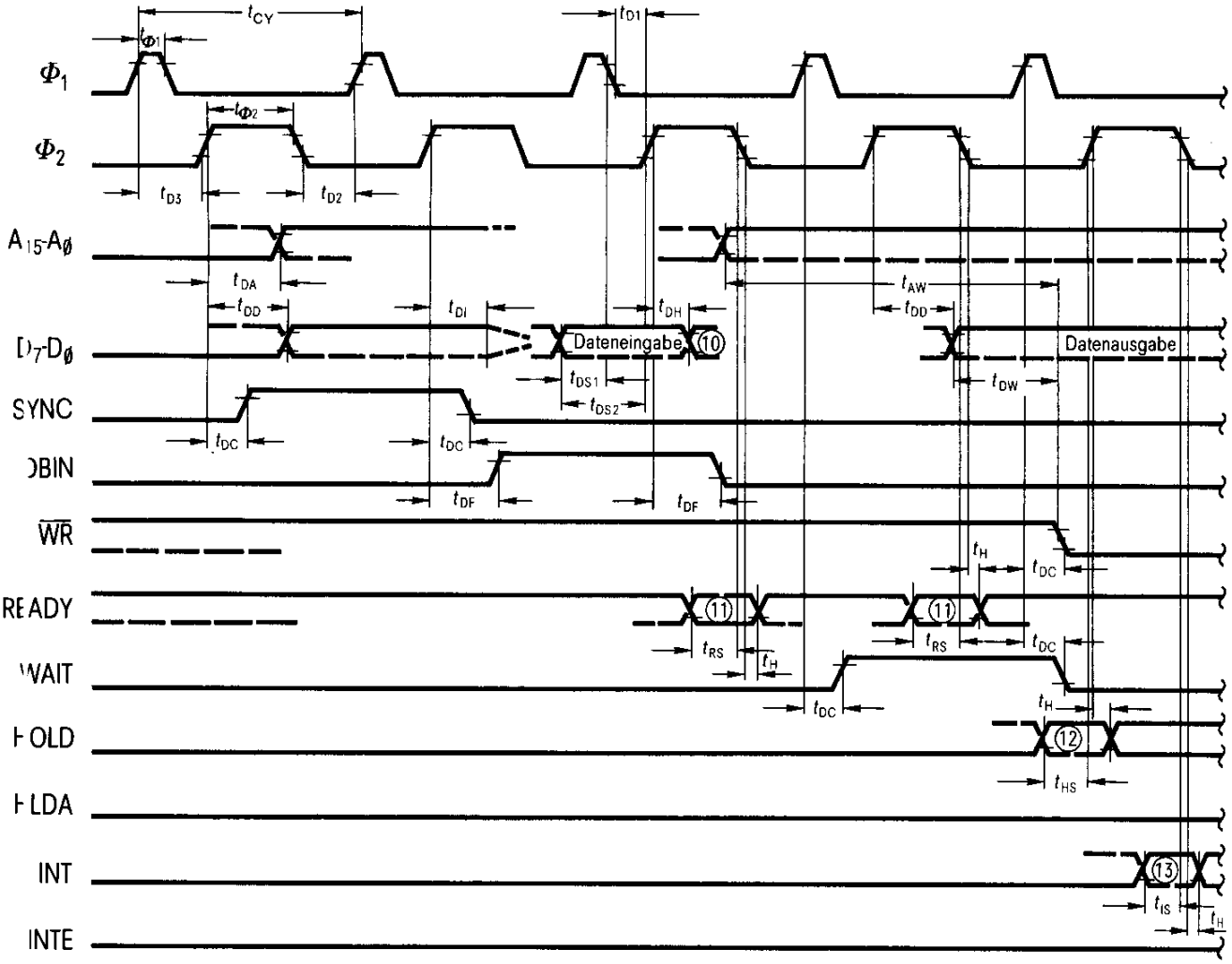
Daten-Bus-Kennlinie während „DBIN"



*) Stromaufnahme/Umgebungstemperatur: $\Delta I / \Delta T_U = -0{,}45\%/°C$

Impulsdiagramm [1])

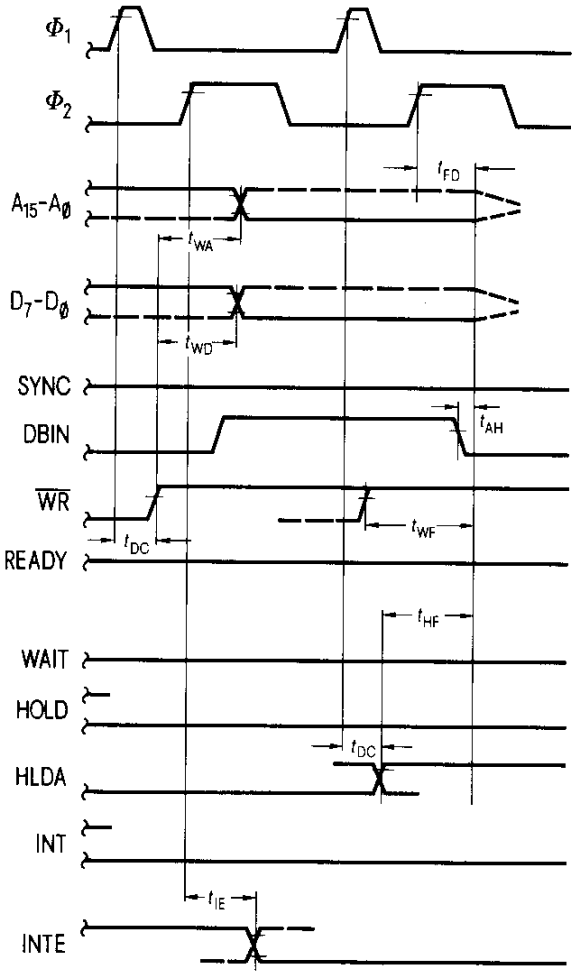
Die zeitlichen Messungen sind für die folgenden Bezugsspannungen gültig:
Takt „1" = 8 V; „Ø" = 1 V;
Eingänge „1" = 3,3 V; „Ø" = 0,8 V;
Ausgänge „1" = 2 V; „Ø" = 0,8 V.



Ⓝ-Anmerkungen siehe Seite 170

[1]) Dieses Impuls-Diagramm stellt nur die zeitlichen Relationen von Signalen dar, nicht einen speziellen Operationszyklus.

Impulsdiagramm (Fortsetzung)



Wenn der SAB 8080A-1 mit voller oder nahezu voller Geschwindigkeit betrieben wird, muß sichergestellt sein, daß die Zeitabläufe der Bausteine SAB 8080A-1, SAB 8224 und SAB 8228 genau aufeinander abgestimmt sind.

Schaltzeiten

$T_U = 0$ bis $+70\,°C$; $V_{DD} = +12\,V \pm 5\%$; $V_{BB} = -5\,V \pm 5\%$; $V_{SS} = 0\,V$
(wenn nicht anders angegeben)

Symbol	Bezeichnung	Typ	Grenzwerte		Einheit	Prüfbedingung
			min.	max.		
t_{CY} [3]	Taktperiode	8080A-1	0,32	2	µs	
		8080A-2	0,38			
t_r, t_f	Takt-Anstiegs- und Abfallzeit	8080A-1	0	25	ns	
		8080A-2		50		
$t_{\Phi 1}$	Φ_1-Impulsdauer	8080A-1	50	–		–
		8080A-2	60			
$t_{\Phi 2}$	Φ_2-Impulsdauer	8080A-1	145			
		8080A-2	175			
t_{D1}	Verzögerung Φ_1 bis Φ_2	beide	0			
t_{D2}	Verzögerung Φ_2 bis Φ_1	8080A-1	60			
		8080A-2	70			
t_{D3}	Verzögerung Φ_1 bis Φ_2 vordere Flanken	8080A-1	60			
		8080A-2	70			
t_{DA} [2]	Adressenausgabe – Verzögerung von Φ_2	8080A-1	–	150		(8080A-1) $C_L = 50$ pF
		8080A-2		175		
t_{DD} [2]	Datenausgabe – Verzögerung von Φ_2	8080A-1		180		(8080A-2) $C_L = 100$ pF
		8080A-2		200		
t_{DC} [2]	Signalausgabe – Verzögerung von Φ_1 oder Φ_2 (SYNC, $\overline{WR}$, WAIT, HLDA)	8080A-1		110		$C_L = 50$ pF
		8080A-2		120		
t_{DC} [2]	DBIN – Verzögerung von Φ_2	8080A-1	25	130		
		8080A-2		140		
t_{DI} [1]	Eingabezustands-Verzögerung des Daten-Bus	beide	–	t_{DF}		–
t_{DS1}	Daten-Vorbereitungszeit während Φ_1 und DBIN	8080A-1	10	–		
		8080A-2	20			

Anmerkungen siehe Seite 170

Schaltzeiten (Fortsetzung)

Symbol	Bezeichnung	Typ	Grenzwerte min.	Grenzwerte max.	Einheit	Prüfbedingung
t_{DS2}	Daten-Vorbereitungszeit bis Φ_2 während DBIN	8080A-1	120	–	ns	–
		8080A-2	130			
t_{DH} [1]	Daten-Haltezeit von Φ_2 während DBIN		[1]			
t_{IE} [2]	Verzögerung der Unterbrechungs-Freigabe (INTE) von Φ_2		–	200		C_L=50 pF
t_{RS}	Bereit-(READY-)Vorbereitungszeit während Φ_2		90	–		–
t_{HS}	Anhalte-(HOLD-)Vorbereitungszeit bis Φ_2		120			
t_{IS}	Unterbrechungs-(INT-) Vorbereitungszeit während Φ_2 (während Φ_1 im HALT-Zustand)		100			
t_H	Haltezeit von Φ_2 (READY, INT, HOLD)		0			
t_{FD}	Verzögerung bis hochohmiger Zustand während HOLD (Adressen- und Daten-Bus)		–	120		
t_{AW} [2]	Adresse stabil vor $\overline{WR}$		[5]	–		C_L=50 pF (8080A-1) C_L=100 pF (8080A-2) Adressen, Daten C_L=50 pF $\overline{WR}$, HLDA, DBIN
t_{DW} [2]	Ausgangs-Daten stabil vor $\overline{WR}$		[6]			
t_{WD} [2]	Ausgangs-Daten stabil nach $\overline{WR}$		[7]			
t_{WA} [2]	Adresse stabil nach $\overline{WR}$		[7]			
t_{HF} [2]	Verzögerung HLDA bis hochohmiger Zustand		[8]			
t_{WF} [2]	Verzögerung $\overline{WR}$ bis hochohmiger Zustand		[9]			
t_{AH} [2]	Adressen-Haltezeit nach DBIN während HLDA		−20			

Anmerkungen siehe Seite 170

Anmerkungen der Seiten 166 bis 169

[1]) Dateneingaben sind mit DBIN-Zustand freizugeben. Auf diese Weise werden Bus-Konflikte vermieden, und die Daten-Haltezeit ist sichergestellt. $t_{DH}=50$ ns oder t_{DF} das Minimum von beiden.

[2]) Meßschaltung (Ausgangsbelastung, siehe nächste Seite).

[3]) $t_{CY}=t_{D3}+t_{r\Phi2}+t_{\Phi2}+t_{f\Phi2}+t_{D2}+t_{r\Phi1}\geqq 320$ ns (SAB 8080A-1).
$t_{CY}=t_{D3}+t_{r\Phi2}+t_{\Phi2}+t_{f\Phi2}+t_{D2}+t_{r\Phi1}\geqq 380$ ns (SAB 8080A-2).

[4]) Die folgenden Angaben gelten, wenn die Prozessoren an Bausteine mit $V_{IH}=3{,}3$ V angeschlossen wird:
a) Max. Ausg.-Anstiegszeit von 0,8 auf 3,3 V = 100 ns bei C_L = spez.
b) Ausg.-Verzögerung gemessen bis 3 V = spez. + 60 ns bei C_L = spez.
c) Wenn $C_L \neq$ spez.: Addiere 0,6 ns/pF, wenn $C_L >$ C spez.; subtrahiere 0,3 ns/pF (vom modifizierten Verzögerungswert), wenn $C_L <$ C spez. (Bild siehe nächste Seite).

[5]) $t_{AW}=2t_{CY}-t_{D3}-t_{r\Phi2}-110$ ns (SAB 8080A-1).
$t_{AW}=2t_{CY}-t_{D3}-t_{r\Phi2}-130$ ns (SAB 8080A-2).

[6]) $t_{DW}=t_{CY}-t_{D3}-t_{r\Phi2}-150$ ns (SAB 8080-A1).
$t_{DW}=t_{CY}-t_{D3}-t_{r\Phi2}-170$ ns (SAB 8080-A2).

[7]) Wenn nicht HLDA, $t_{WD}=t_{WA}=t_{D3}+t_{r\Phi2}+10$ ns.
Wenn HLDA, $t_{WD}=t_{WA}=t_{WF}$.

[8]) $t_{HF}=t_{D3}+t_{r\Phi2}-50$ ns.

[9]) $t_{WF}=t_{D3}+t_{r\Phi2}-10$ ns.

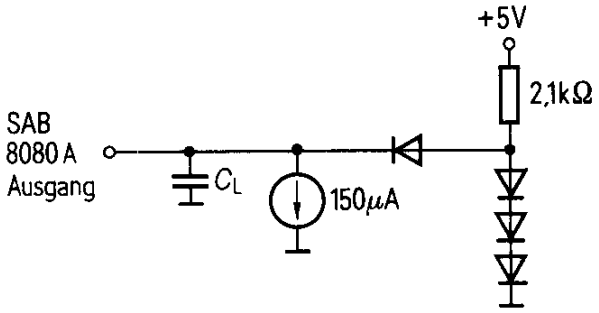
[10]) Eingabedaten müssen für dieses Zeitintervall während DBIN·T_3 stabil sein. Sowohl die t_{DS1}- als auch die t_{DS2}-Forderung muß erfüllt sein.

[11]) Das Bereit-Signal muß für dieses Zeitintervall während T_2 oder T_W stabil sein. (Muß extern synchronisiert werden.)

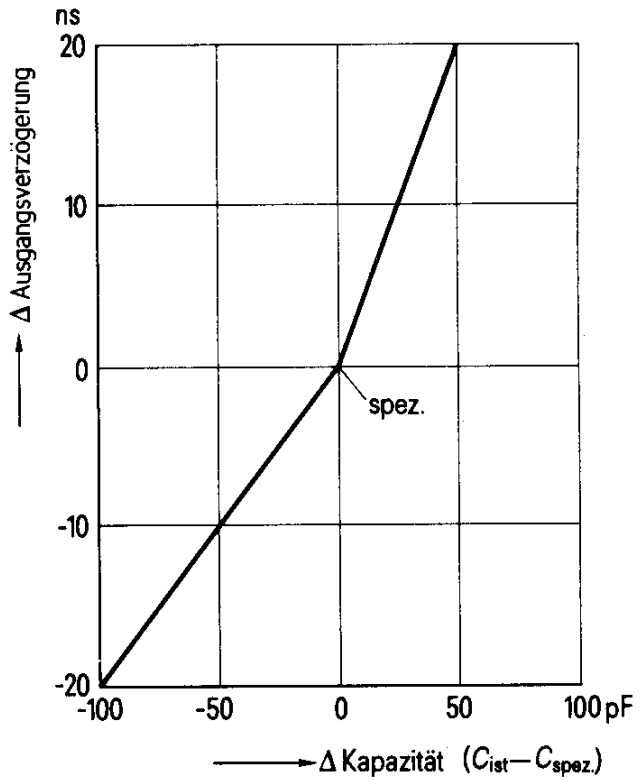
[12]) Das HOLD-Signal muß für dieses Zeitintervall stabil sein, und zwar bei Eintritt in den HOLD-Zustand während T_2 oder T_W und nach Erreichen des HOLD-Zustands während T_3, T_4, T_5 und T_{WH}.
(Externe Synchronisation nicht erforderlich.)

[13]) Das Unterbrechungs-Signal muß während dieses Zeitintervalls im letzten Taktzyklus eines jeden Befehls stabil sein, um im nachfolgenden Befehl erkannt zu werden.
(Externe Synchronisation nicht erforderlich.)

Meßschaltung



Typische Δ Ausgangsverzögerung in Abhängigkeit von der Δ Kapazität



Taktgeber- und Treiber-Baustein SAB 8224

Taktgeber/Treiber für den Mikroprozessor SAB 8080A

Erzeugung eines Rücksetzsignals für den SAB 8080A beim Einschalten der Versorgungsspannung

Flipflop zur Synchronisierung des Bereit-Signals

Oszillator-Ausgang für Zeitabläufe externer Systeme

Quarz-gesteuert

Vorverlegtes Zustandsübernahmesignal

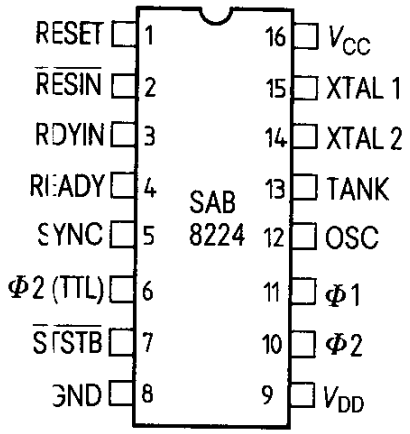
Der SAB 8224 ist ein Taktgeber und Treiber für den SAB 8080A. Die Arbeitsgeschwindigkeit kann durch Auswahl des externen Quarzes vom Entwickler in weiten Grenzen festgelegt werden.

Er beinhaltet eine Logik, die beim Einschalten der Versorgungsspannung ein Rücksetzsignal für den SAB 8080A erzeugt, sowie für ein vorverlegtes Zustandsübernahmesignal und für eine Synchronisierung des Bereit-Signals sorgt.

Durch den Baustein SAB 8224 wird die Anzahl nötiger Bausteine in einem System wesentlich reduziert.

SAB 8224

Anschlußbelegung

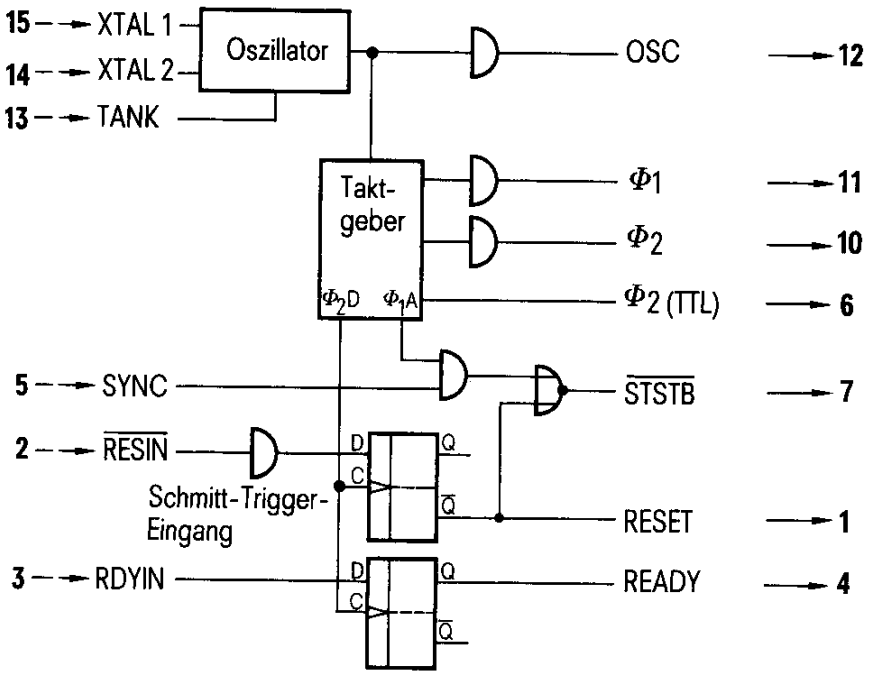


Abmessungen am Schluß des Buches

Anschlußbezeichnungen

$\overline{\text{RESIN}}$	Rücksetz-Eingang
RESET	Rücksetz-Ausgang
RDYIN	Bereit-Eingang
READY	Bereit-Ausgang
SYNC	Synchr. Eingang
$\overline{\text{STSTB}}$	Zustandsübernahme („Low" aktiv)
$\Phi 1$	Takt für SAB 8080
$\Phi 2$	
XTAL 1	Anschlüsse für externen Quarz
XTAL 2	
TANK	Eingang für Oberwellen-Quarz
OSC	Oszillator-Ausgang
$\Phi 2$ (TTL)	Takt $\Phi 2$ (TTL-Pegel)
V_{CC}	Versorgungsspannung (+5 V)
V_{DD}	Versorgungsspannung (+12 V)
GND	Masse (0 V)

Blockschaltbild



Funktionsbeschreibung

Allgemeines

Der SAB 8224 ist ein Taktgeber- und Treiber-Baustein für den SAB 8080. Er besteht aus einem Oszillator (extern quarz-gesteuert), einem Zählerteiler 1:9, zwei Treibern mit hoher Ausgangsspannung und verschiedenen logischen Hilfsfunktionen.

Oszillator

Die Basisbetriebsfrequenz des Schwingkreises kommt aus einem externen Reihen-Resonanz-Quarz vom Grundwellentyp. Für die Quarzanschlüsse sind zwei Eingänge vorgesehen (XTAL1 und XTAL2).

Die Auswahl der Frequenz des externen Quarzes hängt hauptsächlich von der Geschwindigkeit ab, mit der der SAB 8080 betrieben werden soll. Grundsätzlich arbeitet der Oszillator mit dem 9fachen der gewünschten Prozessor-Geschwindigkeit.

Eine einfache Formel zur Auswahl des Quarzes ist

$$\text{Quarzfrequenz} = \frac{1}{t_{CY}} \times 9$$

Beispiel 1: (500 ns t_{CY})
2 MHz × 9 = 18 MHz[1])

Beispiel 2: (800 ns t_{CY})
1,25 MHz × 9 = 11,25 MHz

Ein weiterer Eingang des Oszillators ist TANK. Dieser Eingang ermöglicht die Anwendung von Oberwellenquarzen. Diese Art von Quarz erzeugt im allgemeinen sehr viel weniger „Leistung" als der Grundwellentyp; aus diesem Grunde ist eine externe LC-Schaltung erforderlich. Die externe LC-Schaltung wird mit dem TANK-Eingang verbunden und wechselstrommäßig mit Masse gekoppelt.

Die Formel für die LC-Schaltung lautet:

$$F = \frac{1}{2\pi\sqrt{LC}}.$$

Der Ausgang des Oszillators ist gepuffert und am OSC (Anschluß 12) herausgeführt, so daß auch andere Zeitsignale von dieser konstanten, quarzgesteuerten Quelle abgeleitet werden können.

[1]) Wenn Quarze bei Frequenzen über 10 MHz eingesetzt werden, kann eine geringfügige Abgleichung der Frequenz erforderlich sein, um genau den gewünschten Wert zu erhalten (Kapazität von 3 pF bis 10 pF in Reihe zum Quarz).

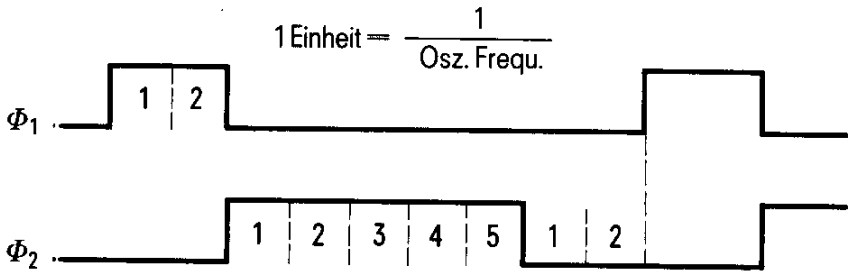
Taktgeber

Der Taktgeber besteht aus einem synchronen Zählerteiler im Verhältnis 1:9 und den zugehörigen Dekodier-Gattern zur Erzeugung der Impulse der beiden SAB 8080A-Takte und der Hilfssignale für den Zeitablauf.

Die vom Dekodier-Gatter erzeugten Impulsformen folgen einem 2-5-2-Digital-Muster (siehe Impulsdiagramm). Die erzeugten Takte – Phase 1 und 2 – stellt man sich am besten so vor, als bestünden sie aus „Einheiten", die von der Oszillator-Frequenz abgeleitet sind. Nehmen wir an, eine „Einheit" sei gleich der Periode der Oszillator-Frequenz. Multipliziert man nun die Anzahl der „Einheiten", die während einer Impulsdauer oder Impulsverzögerung auftreten, mit der Periode der Oszillator-Frequenz, so erhält man die ungefähre Zeit in ns.

Die Ausgänge des Taktgebers sind mit 2 Treibern mit hoher Ausgangsspannung zum direkten Anschluß an den SAB 8080A verbunden. Ferner ist für externe Zeitabläufe ein Takt $\Phi 2$ mit TTL-Pegeln herausgeführt: Φ_2 TTL. Er ist besonders für DMA-abhängige Vorgänge von Bedeutung. Dieses Signal dient dazu, dem anfragenden Baustein den Bus zuzuteilen, sobald der SAB 8080A die HOLD-Quittung (HLDA) ausgibt. Verschiedene andere Signale werden ebenfalls intern erzeugt, so daß ein optimaler Zeitablauf bei den Hilfs-Flipflops und dem Zustandsübernahmesignal ($\overline{STSTB}$) erzielt wird.

Impulsdiagramm



Beispiel: SAB 8080A t_{CY} (Taktperiode) = 500 ns
OSC = 18 MHz ≙ 55 ns
Φ_1 = 110 ns (2 × 55 ns)
Φ_2 = 275 ns (5 × 55 ns)
$\Phi_2 - \Phi_1$ = 110 ns (2 × 55 ns)

$\overline{\text{STSTB}}$ (Zustandsübernahmesignal)

Zu Beginn jedes Operationszyklus gibt der SAB 8080A Zustandsinformationen auf dem Daten-Bus aus.

Vom SAB 8080A wird das SYNC-Signal eingegeben und mit einem internen Zeitablaufsignal (Φ1A) so verknüpft, daß sich eine aktive „Low"-Übernahme ergibt, die zu Beginn jedes Operationszyklus zum frühestmöglichen Augenblick erscheint, an dem die Zustandsdaten auf dem Bus konstant sind. Das $\overline{\text{STSTB}}$-Signal kann direkt mit dem Systemsteuer-Baustein SAB 8228 verbunden werden.

Das Rücksetzsignal beim Einschalten der Versorgungsspannung erzeugt auch das $\overline{\text{STSTB}}$-Signal, jedoch selbstverständlich für eine längere Zeit. Diese Eigenschaft ermöglicht ein automatisches Rücksetzen des SAB 8228, ohne einen zusätzlichen Anschluß.

Rücksetzsignal für den SAB 8080A beim Anlegen der Versorgungsspannung

Der SAB 8224 ermöglicht ein automatisches System-Rücksetzen und Anlaufen bei Anlegen der Versorgungsspannung, was bei allen SAB 8080-Computersystemen nötig ist.

Eine externe RC-Schaltung ist mit dem $\overline{\text{RESIN}}$-Eingang verbunden. Der langsame Anstieg der Versorgungsspannung wird von einem internen Schmitt-Trigger in einen schnellen Spannungssprung umgewandelt. Der Ausgang des Schmitt-Triggers ist an ein „D"-Flipflop geschaltet, das mit Φ2D (Φ2DELAYED, ein verzögertes, internes Zeitsignal) getaktet ist. Das Flipflop wird synchron zurückgesetzt und es wird ein aktives „High"-Signal RESET erzeugt, das die Eingangsspezifikation des SAB 8080A erfüllt. Für manuelles Rücksetzen kann zusätzlich zur RC-Schaltung ein Schalter (Arbeitskontakt) zwischen $\overline{\text{RESIN}}$-Eingang und Masse geschaltet werden.

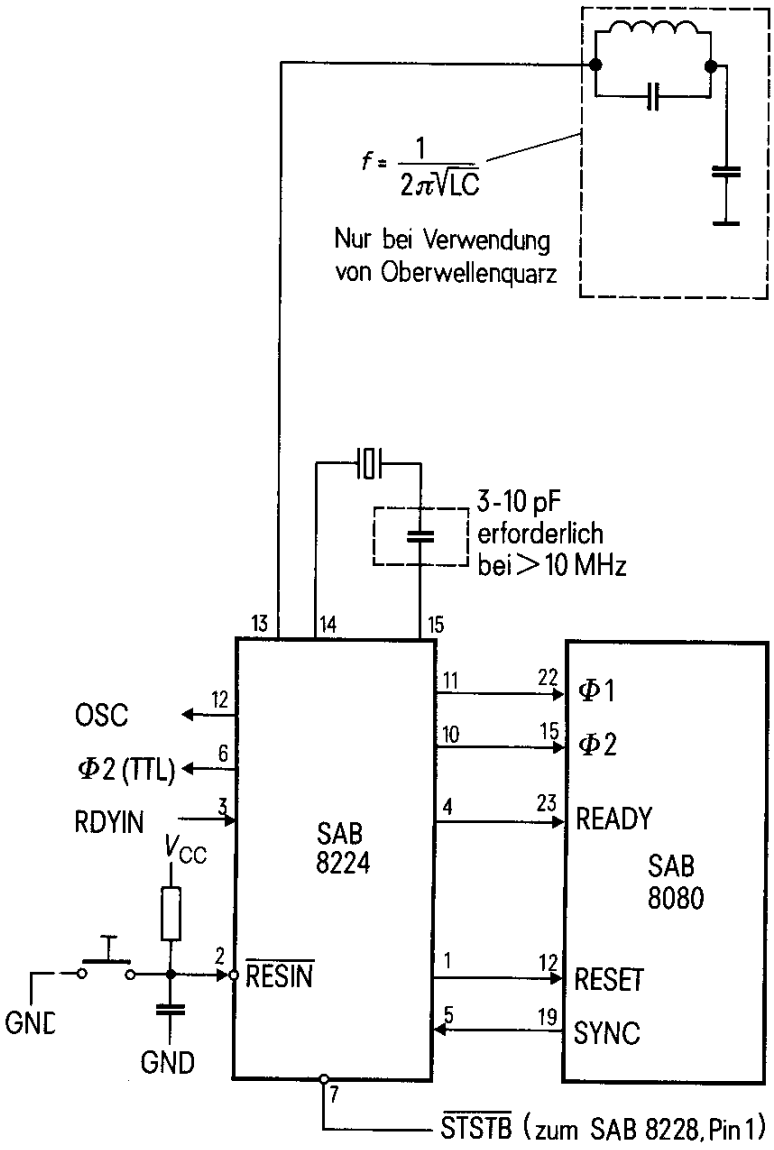
Bereit Flipflop

Der Eingang „READY" des SAB 8080A verlangt bestimmte, definierte Zeitabläufe wie z.B. „Vorbereiten und Halten". Hierfür ist ein externes Synchronisier-Flipflop erforderlich. Dies ist im SAB 8224 eingebaut. Der RDYIN-Eingang gibt die asynchrone „Warteanforderung" an ein „D"-Flipflop, das mit Φ2D getaktet wird, so daß ein synchronisiertes Bereit-Signal mit dem richtigen Eingangspegel direkt mit dem SAB 8080A verbunden werden kann.

Ein externes Flipflop zur Synchronisierung der Anforderung „Warten" ist erforderlich, weil ein im SAB 8080A eingebautes Flipflop durch die relativ langen Verzögerungszeiten der MOS-Technologie eine Zeit von ca. 200 ns brauchen würde.

Eine bipolare Schaltung, die in den Taktgeber eingebaut ist, eliminiert diese Verzögerung zum größten Teil und bewirkt keinen zusätzlichen Bausteinaufwand.

Anschluß an den SAB 8080A



Quarz-Anforderungen

Toleranz:	0,005% bei 0 bis 70 °C
Resonanz:	Reihenresonanz (Grundwellentyp)
Lastkapazität:	20 bis 35 pF
Ersatzwiderstand:	75 bis 20 Ω
Verlustleistung (min.):	4 mW

Grenzdaten [1])

Betriebstemperatur	0 bis + 70 °C
Lagertemperatur	−65 bis +150 °C
Versorgungsspannung, V_{CC}	−0,5 bis + 7 V
Versorgungsspannung, V_{DD}	−0,5 bis + 13,5 V
Eingangsspannung	−1,5 bis + 7 V
Ausgangsstrom	100 mA

Statische Kenndaten und Betriebsbedingungen

$T_U = 0$ bis $+70$ °C; $V_{CC} = +5\,V \pm 5\%$; $V_{DD} = +12\,V \pm 5\%$ (wenn nicht anders angegeben)

Symbol	Bezeichnung	Grenzwerte min.	Grenzwerte max.	Einheit	Prüfbedingung
I_F	L-Eingangsstrom	–	−0,25	mA	$V_F = 0{,}45$ V
I_R	H-Eingangsstrom	–	10	µA	$V_R = 5{,}25$ V
V_C	Eingangsklemmspannung	–	−1	V	$I_C = -5$ mA
V_{IL}	L-Eingangsspannung	–	0,8	V	$V_{CC} = 5$ V
V_{IH}	H-Eingangsspannung	2,6	–	V	Rücksetz-Eingang
		2	–	V	Alle anderen Eingänge
$V_{IH} - V_{IL}$	RESIN Eingangs-Hysterese-Spannung	0,25	–	V	$V_{CC} = 5$ V
V_{OL}	L-Ausgangsspannung	–	0,45	V	(Φ_1, Φ_2), READY, RESET, $\overline{\text{STSTB}}$ $I_{OL} = 2{,}5$ mA
		–	0,45	V	Alle anderen Ausgänge $I_{OL} = 15$ mA
V_{OH}	H-Ausgangsspannung Φ_1, Φ_2	9,4	–	V	$I_{OH} = -100$ µA
	READY, RESET	3,6	–	V	$I_{OH} = -100$ µA
	Alle anderen Ausgänge	2,4	–	V	$I_{OH} = -1$ mA
I_{SC} [2])	Kurzschlußausgangsstrom (Nur Ausgänge niederer Spannung, nicht beim Φ_1- und Φ_2-Ausgang)	−10	−60	mA	$V_O = 0$ V, $V_{CC} = 5$ V
I_{CC}	Stromaufnahme (V_{CC})	–	115	mA	–
I_{DD}	Stromaufnahme (V_{DD})	–	12	mA	–

[1]) Die Überschreitung der angegebenen Grenzdaten kann zu Dauerschäden am Baustein führen.

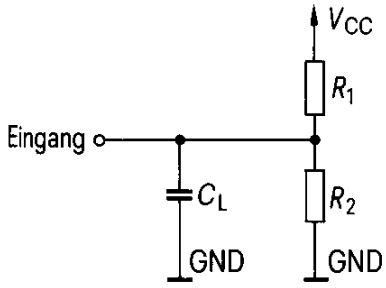
[2]) Vorsicht, Ausgangstreiber Φ_1 und Φ_2 haben keinen Kurzschlußschutz.

Schaltzeiten

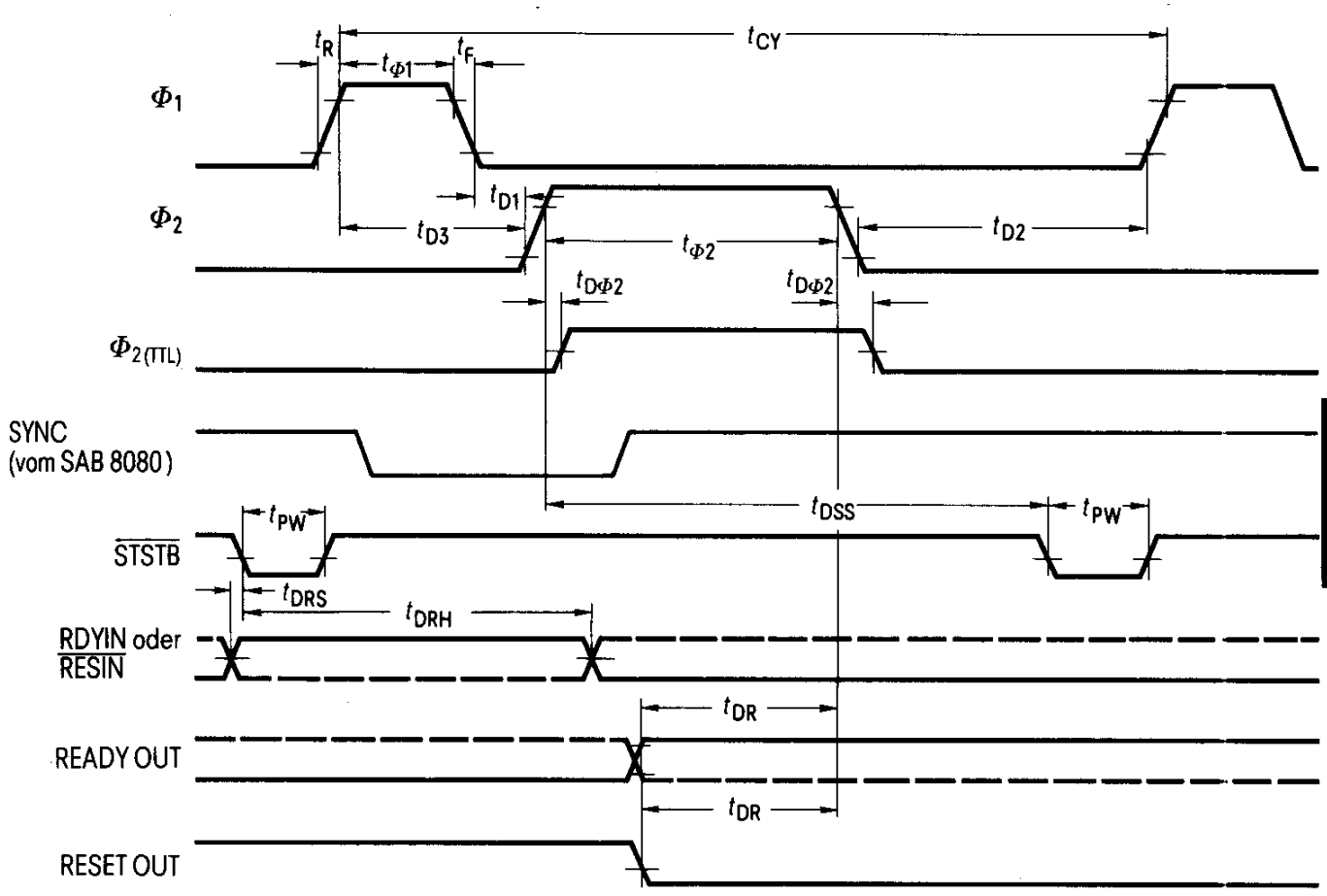
T_U = 0 bis +70 °C; V_{CC} = +5 V ±5%; V_{DD} = +12 V ±5%.

Symbol	Bezeichnung	Grenzwerte			Einheit	Prüfbedingung
		min.	typ.	max.		
$t_{\Phi 1}$	Impulsdauer Φ_1	$\frac{2tcy}{9}-20\,ns$				
$t_{\Phi 2}$	Impulsdauer Φ_2	$\frac{5tcy}{9}-35\,ns$		–		
t_{D1}	Verzögerung von Φ_1 bis Φ_2	0				
t_{D2}	Verzögerung von Φ_2 bis Φ_1	$\frac{2tcy}{9}-14\,ns$				C_L = 20 bis 50 pF
t_{D3}	Verzögerung von Φ_1 bis Φ_2	$\frac{2tcy}{9}$		$\frac{2tcy}{9}+20\,ns$		
t_R	Anstiegszeit von Φ_1 und Φ_2	–		20		
t_F	Abfallzeit von Φ_1 und Φ_2	–		20		
$t_{D\Phi 2}$	Verzögerung von Φ_2 bis $\Phi_{2(TTL)}$	−5	–	+15	ns	Φ_2(TTL), C_L = 30 pF R_1 = 300 Ω R_2 = 600 Ω
t_{DS1}	Verzögerung von Φ_2 bis $\overline{STSTB}$	$\frac{6tcy}{9}-30$ ns		$\frac{6tcy}{9}$		
t_{PW}	Impulsdauer $\overline{STSTB}$	$\frac{tcy}{9}-15$ ns				$\overline{STSTB}$, C_L = 15 pF R_1 = 2 kΩ R_2 = 4 kΩ
t_{DR1}	Vorbereitungszeit RDYIN bis zur Zustandsübernahme	$50\,ns-\frac{4tcy}{9}$				
t_{DR2}	Haltezeit RDYIN nach $\overline{STSTB}$	$\frac{4tcy}{9}$				
t_{DR}	Verzögerung READY oder RESET bis Φ_2	$\frac{4tcy}{9}-25\,ns$		–		READY und RESET C_L = 10 pF R_1 = 2 kΩ R_2 = 4 kΩ
t_{CLK}	CLK (Takt) Periode	–	$\frac{tcy}{9}$			–
f_{max}	Max. Schwingungsfrequenz	27			MHz	–
C_{in}	Eingangskapazität	–	–	8	pF	V_{CC} = +5 V V_{DD} = +12 V $V_{Vorsp.}$ = 2,5 V f = 1 MHz

Meßschaltung



Impulsdiagramm



Spannungsmeßpunkte: Φ_1, Φ_2 Logische „Ø" = 1 V; Logische „1" = 8 V.
Alle anderen Signale gemessen bei 1,5 V.

Beispiel

Schaltzeiten für $t_{CY}=488{,}28$ ns
$T_U=0$ bis $+70\,°C$; $V_{CC}=+5\,V\pm5\%$; $V_{DD}=+12\,V\pm5\%$

Symbol	Bezeichnung	Grenzwerte min.	Grenzwerte max.	Einheit	Prüfbedingung
$t_{\Phi 1}$	Impulsdauer Φ_1	89		ns	$t_{CY}=488{,}28$ ns
$t_{\Phi 2}$	Impulsdauer Φ_2	236	–		
t_{D1}	Verzögerung von Φ_1 bis Φ_2	0			Φ_1 und Φ_2 belastet mit $C_L=20$ bis 50 pF
t_{D2}	Verzögerung von Φ_2 bis Φ_1	95			
t_{D3}	Verzögerung von Φ_1 bis Φ_2,	109	129		
t_r	Anstiegszeit am Ausgang	–	20		
t_f	Abfallzeit am Ausgang	–	20		
t_{DSS}	Verzögerung von Φ_2 bis $\overline{STSTB}$	296	326	ns	READY und RESET belastet mit 2 mA/10 pF Alle Messungen bezogen auf 1,5 V, wenn nicht anders angegeben
$t_{D\Phi 2}$	Verzögerung von Φ_2 bis $\Phi_{2(TTL)}$	−5	+15		
t_{PW}	Impulsdauer der Zustandsübernahme $\overline{STSTB}$	40	–		
t_{DRS}	Vorbereitungszeit RDYIN bis zu $\overline{STSTB}$	−167			
t_{DRH}	Haltezeit RDYIN nach $\overline{STSTB}$	217			
t_{DR}	READY oder RESET Verzögerung bis Φ_2	192			
f_{max}	Oszillator-Frequenz	–	18,432	MHZ	

System-Steuer- und Bus-Treiber-Baustein

SAB 8228
SAB 8238

Systemsteuerung für SAB 8080-Systeme

Zweiweg-Bus-Treiber zur Daten-Bus-Trennung

Möglichkeit der Anwendung von Mehr-Byte-Befehlen, z.B. CALL-Befehl zur Unterbrechungsquittung

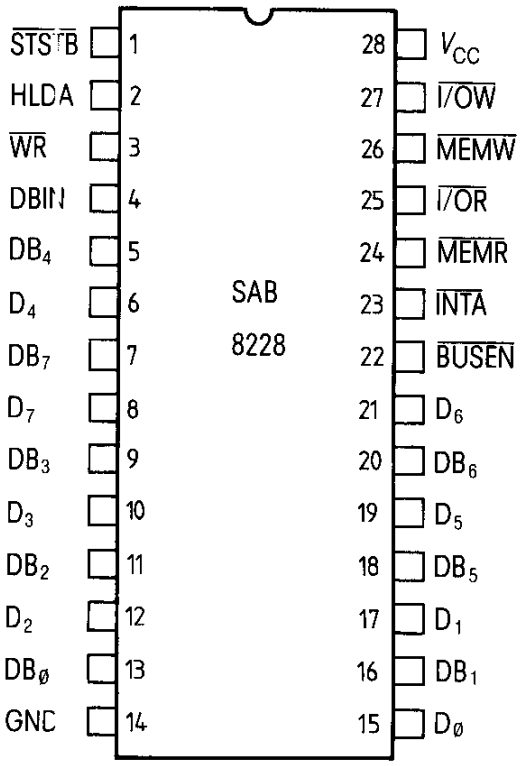
Möglichkeit zur Erzeugung eines RST 7-Befehls

Der SAB 8228 und der SAB 8238 sind System-Steuerbausteine und Bus-Treiber für SAB 8080 Mikrocomputer-Systeme. Sie erzeugen alle Signale, die zur direkten Kopplung von RAM, ROM und E/A-Bausteinen erforderlich sind und haben einen Bus-Treiber für höhere Ausgangsbelastungen. Er bewirkt außerdem die Trennung zwischen Mikroprozessor-Daten-Bus und Speicher sowie Ein- und Ausgabe-Bausteinen. Dadurch können langsamere Speicher- und E/A-Bausteine eingesetzt werden. Durch den Bus-Treiber ist überdies eine erhöhte Störsicherheit des Systems gewährleistet.

Für die einfache Realisierung von Systemen mit Programmunterbrechung („Interrupt"), liefert der SAB 8228/38 ohne zusätzlichen Aufwand einen Einfach-Unterbrechungs-Vektor (RST 7). Außerdem erzeugt der SAB 8228/38 die Steuersignale für Mehr-Byte-Befehle, d.h. CALL-Befehle für die Verzweigung auf ein Unterbrechungsbearbeitungsprogramm nach einer Unterbrechungsanforderung. Dadurch können große unterbrechungsgesteuerte Systeme mit einer praktisch unbegrenzten Anzahl von Unterbrechungs-Ebenen arbeiten.

SAB 8228
SAB 8238

Anschlußbelegung

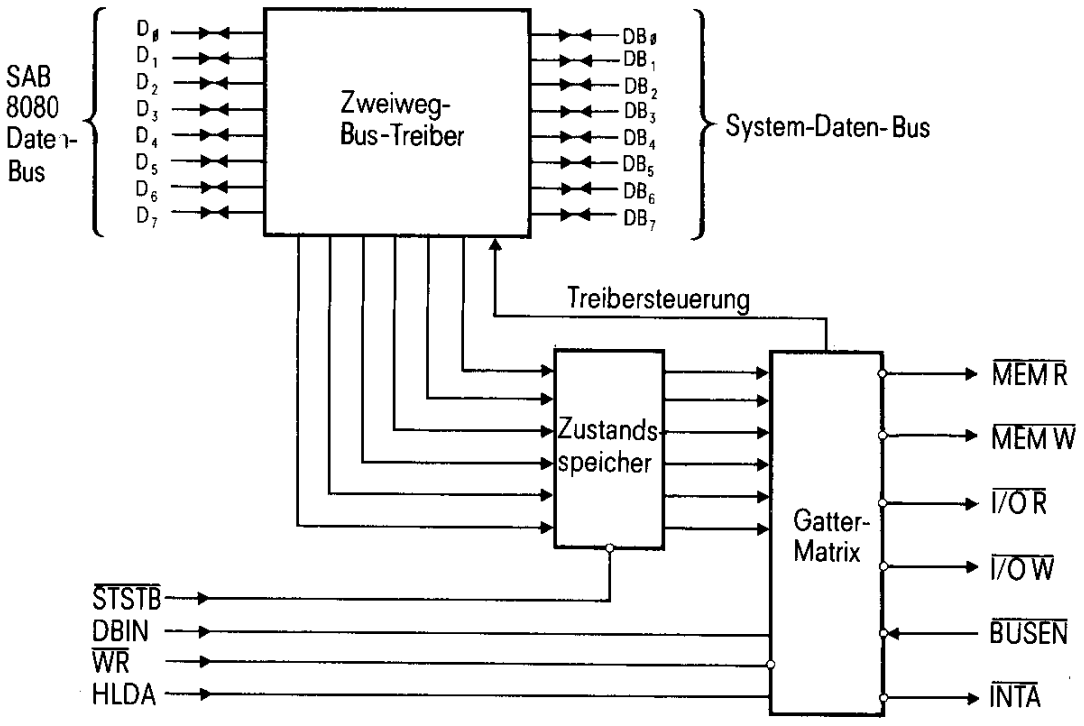


Abmessungen am Schluß des Buches

Anschlußbezeichnungen

$D_Ø$–D_7	Daten-Bus (SAB 8080A-Seite)
$DB_Ø$–DB_7	Daten-Bus (Systemseite)
$\overline{I/OR}$	E/A Lesen
$\overline{I/OW}$	E/A Schreiben
$\overline{MEMR}$	Speicher Lesen
$\overline{MEMW}$	Speicher Schreiben
DBIN	DBIN (vom SAB 8080A)
$\overline{INTA}$	Unterbrechungs-Quittung
HLDA	HLDA (vom SAB 8080A)
$\overline{WR}$	$\overline{WR}$ (vom SAB 8080A)
$\overline{BUSEN}$	Bus-Freigabe-Eingang
$\overline{STSTB}$	Zustandsübernahme (vom SAB 8224)
V_{CC}	Versorgungsspannung (+5 V)
GND	Masse (0 V)

Blockschaltbild



Funktionsbeschreibung

Allgemeines

Der SAB 8228/38 ist ein Systemsteuerbaustein und Datenbus-Treiber in einem Gehäuse. Er erzeugt alle Steuersignale, die zum direkten Anschluß der Bausteine der Mikroprozessor-Familie SAB 8080 wie RAM, ROM und E/A-Bausteine, erforderlich sind.

Die Schottky-Bipolar-Technologie gewährleistet kurze Verzögerungszeiten und hohe Ausgangs-Belastbarkeit.

Zweiweg-Bus-Treiber

Der eingebaute acht-Bit-Zweiweg-Bus-Treiber trennt den 8080A Daten-Bus von Speicher- und E/A-Bausteinen. Der Daten-Bus des SAB 8080A erfordert eine Eingangsspannung von min. 3,3 Volt und kann einen Maximalstrom von 1,9 mA schalten. Der Daten-Bus-Treiber des SAB 8228/38 gewährleistet darüber hinaus eine erhöhte Störsicherheit. Auf der Systemseite des Treibers steht ein Treiberstrom von 10 mA (Richtwert) zur Verfügung, so daß eine Vielzahl von Speichern und E/A-Bausteinen mit dem Bus verbunden werden kann.

Der Zweiweg-Daten-Bus-Treiber wird durch Signale von der Gatter-Matrix gesteuert. Seine Ausgänge können für direkten Speicherzugriff in den hochohmigen Zustand geschaltet werden.

Zustandsspeicher

Zu Beginn jedes Befehlszyklus gibt der SAB 8080A „Zustands"-Information an den Daten-Bus aus.

Der SAB 8228/38 speichert diese Information im Zustandsspeicher, sobald an dem $\overline{\text{STSTB}}$ (STATUS STROBE)-Eingang L-Pegel liegt. Der Ausgang des Zustandsspeichers ist mit der Gatter-Matrix verbunden.

Gatter-Matrix

Die Gatter-Matrix erzeugt Steuersignale $\overline{\text{MEM R}}$ („memory read" – Speicher lesen), $\overline{\text{MEM W}}$ (memory write – Spreicher schreiben), $\overline{\text{I/O R}}$ (input/output read – Eingang/Ausgang lesen), $\overline{\text{I/O W}}$ (input/output write – Eingang/Ausgang schreiben) und $\overline{\text{INTA}}$ (interrupt acknowledge – Unterbrechungs-Quittung) durch Verknüpfen der Ausgänge des Zustandsspeichers.

Die „Lese"-Steuersignale ($\overline{\text{MEM R}}$, $\overline{\text{I/O R}}$ und $\overline{\text{INTA}}$) stammen aus der logischen Verknüpfung des (oder der) zugehörigen Zustands-Bits und dem DBIN-(data bus input) Eingang, der direkt mit dem DBIN-Ausgang des SAB 8080A verbunden ist.

Die „Schreib-Steuersignale" ($\overline{\text{MEM W}}$, $\overline{\text{I/O W}}$) entstehen durch logische Verknüpfung von Zustands-Bits und des $\overline{\text{WR}}$-Signals des SAB 8080A.

Alle Steuersignale sind „LOW"-aktiv und können direkt mit den RAM, ROM und E/A-Bausteinen verbunden werden.

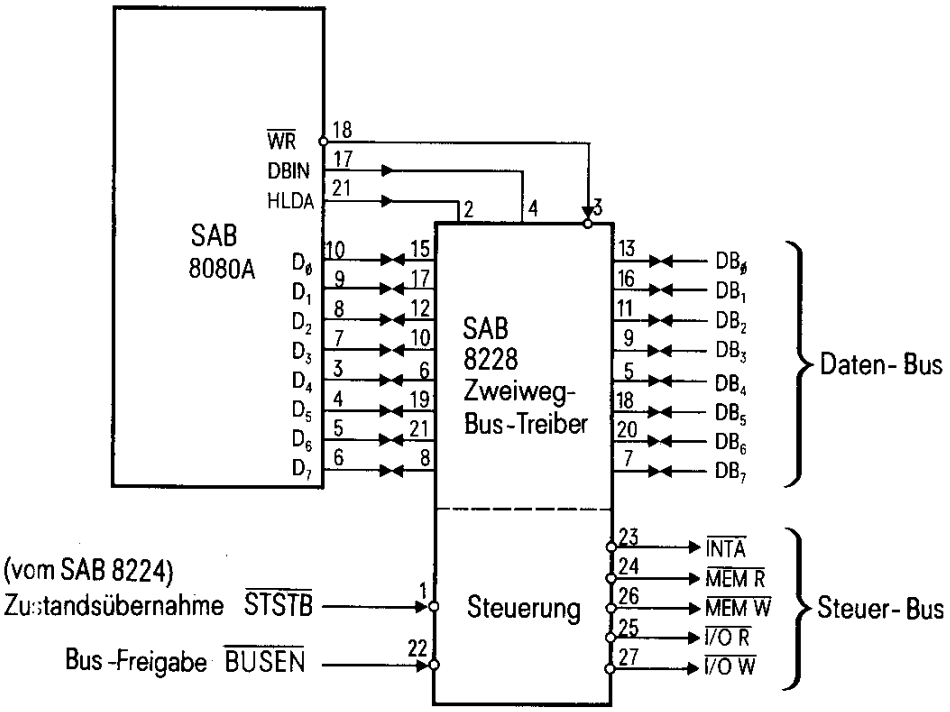
Das $\overline{\text{INTA}}$-Steuersignal zeigt an, daß die Programmunterbrechung vom Mikroprozessor akzeptiert wurde und wird dazu benutzt, der Schaltung, von der die Unterbrechungsanforderung kam, die Anschaltung an den Bus zu gestatten. In kleineren Systemen, in denen nur ein Basis-Vektor benötigt wird, kann dieser Ausgang (Pin 23)

einfach über einen Widerstand (1 kΩ) mit +12 V Versorgungsspannung verbunden werden. Der SAB 8228/38 gibt auf den Bus dann zur geeigneten Zeit automatisch einen RST 7-Befehl. Dadurch ist die automatische Erzeugung eines Unterbrechungs-Vektors ohne zusätzliche Bausteine möglich.

Bei der Verwendung von „CALL" als Unterbrechungsbefehl, erzeugt der SAB 8228/38 einen $\overline{\text{INTA}}$-Impuls für jedes der 3 Befehls-Bytes.

Der $\overline{\text{BUSEN}}$-(bus enable – Bus-Freigabe-)Eingang in der Gatter-Matrix ist ein asynchroner Eingang, der die Daten-Bus-Ausgänge und Steuersignal-Ausgänge in ihren potentialfreien Zustand versetzt, wenn $\overline{\text{BUSEN}}$ auf „HIGH" liegt. Wenn $\overline{\text{BUSEN}}$ „LOW" ist, arbeiten Daten-Puffer und Steuersignale normal.

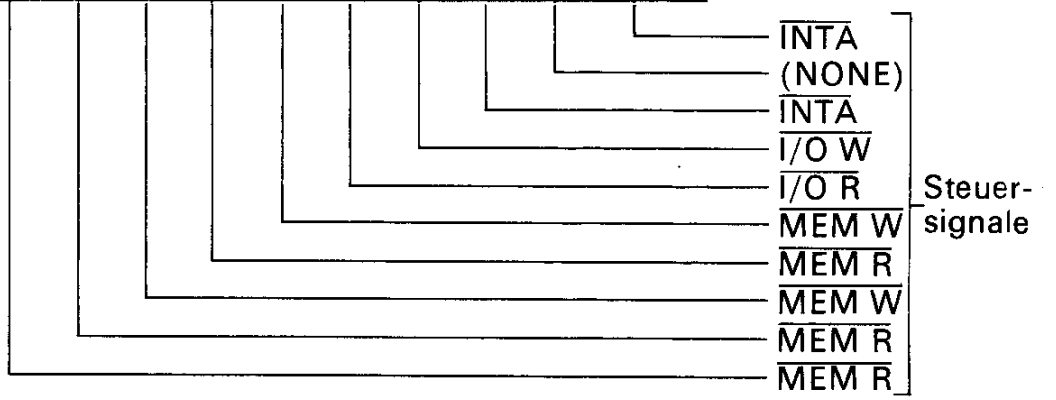
Anschluß an den SAB 8080A



Erzeugung der Steuersignale aus der Zustandsinformation

Art des Arbeitsablaufes

Daten Bus-Bit	Zustands-information	① Befehlsabruf	② Speicher Lesen	③ Speicher Schreiben	④ Stack Lesen	⑤ Stack Schreiben	⑥ Eingabe Lesen	⑦ Ausgabe Schreiben	⑧ Unterbrechungs-Quittung	⑨ Halt-Quittung	⑩ Unterbrechungs-Quittung während des Halt-Zustandes
$D_\emptyset$	INTA	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	1
D_1	$\overline{WO}$	1	1	Ø	1	Ø	1	Ø	1	1	1
D_2	STACK	Ø	Ø	Ø	1	1	Ø	Ø	Ø	Ø	Ø
D_3	HLTA	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	1
D_4	OUT	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø
D_5	M_1	1	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	1
D_6	INP	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
D_7	MEMR	1	1	Ø	1	Ø	Ø	Ø	Ø	1	Ø



Grenzdaten *)

Betriebstemperatur	0 bis + 70 °C
Lagertemperatur	−65 bis +150 °C
Versorgungsspannung	−0,5 bis + 7 V
Eingangsspannung	−1,5 bis + 7 V
Ausgangsstrom	100 mA

*) Die Überschreitung der angegebenen Grenzdaten kann zu Dauerschäden am Baustein führen.

Statische Kenndaten und Betriebsbedingungen

$T_u = 0$ ° bis 70 °C; $V_{cc} = 5\ V \pm 5\%$ (wenn nicht anders angegeben)

Symbol	Bezeichnung	Grenzwerte			Einheit	Prüfbedingung
		min.	typ.*)	max.		
V_C	Eingangs-Klemmspannung, alle Eingänge		−0,75	−1	V	$V_{CC} = 4{,}75\ V$ $I_C = -5\ mA$
I_F	L-Eingangsstrom $\overline{STSTB}$			−500		
	D_2, D_6			−750		$V_{CC} = 5{,}25\ V$
	$D_\emptyset, D_1, D_4, D_5, D_7$	–		−250		$V_F = 0{,}45\ V$
	Alle anderen Eingänge			−250	µA	
I_R	H-Eingangsstrom $\overline{STSTB}$		–	100		$V_{CC} = 5{,}25\ V$ $V_R = 5{,}25\ V$
	$DB_\emptyset - DB_7$			20		
	Alle anderen Eingänge			100		–
V_{TH}	Eingangs-Schwellenspannung	0,8		2	V	$V_{CC} = 5\ V$
I_{CC}	Stromaufnahme		140	190	mA	$V_{CC} = 5{,}25\ V$
V_{OL}	L-Ausgangsspannung $D_\emptyset - D_7$	–	–	0,45	V	$V_{CC} = 4{,}75\ V$ $I_{OL} = 2\ mA$
	Alle anderen Ausgänge			0,45		$I_{OL} = 10\ mA$
V_{OH}	H-Ausgangsspannung $D_\emptyset - D_7$	3,6	3,8	–		$V_{CC} = 4{,}75\ V$ $I_{OH} = -10\ \mu A$
	Alle anderen Ausgänge	2,4				$I_{OH} = -1\ mA$
I_{OS}	Kurzschlußstrom, alle Ausgänge	15		90	mA	$V_{CC} = 5\ V$
$I_{O(off)}$	Ausgangsstrom im hochohmigen Zustand aller Steuer-Ausgänge	–	–	100	µA	$V_{CC} = 5{,}25\ V$ $V_O = 5{,}25\ V$
				−100		$V_O = 0{,}45\ V$
I_{INT}	INTA-Strom			5	mA	(siehe Bild Seite 190)

*) Typische Werte gelten für $T_U = 25$ °C und Nenn-Versorgungsspannung.

Schaltzeiten

$T_U = 0$ bis $+70\,°C$; $V_{CC} = 5\,V \pm 5\%$

Symbol	Bezeichnung	Grenzwerte		Einheit	Prüfbedingung
		min.	max.		
t_{PW}	Impulsdauer der Zustandübernahme	22		ns	–
t_{SS}	Vorbereitungszeit, Zustandseingaben $D_\emptyset - D_7$	8	–		
t_{SH}	Haltezeit, Zustandseingaben $D_\emptyset - D_7$	5			
t_{DC}	Verzögerung von $\overline{STSTB}$ bis zu jedem Steuersignal	20	60		$C_L = 100$ pF
t_{RR}	Verzögerung von DBIN bis zu den Steuerausgängen	–	30		
t_{RE}	Verzögerung von DBIN bis Freigeben oder Sperren des 8080A-Bus		45		$C_L = 25$ pF
t_{RD}	Verzögerung vom System-Bus zum 8080A-Bus während des Lesens		30		
t_{WR}	Verzögerung von $\overline{WR}$ bis zu den Steuerausgängen	5	45		$C_L = 100$ pF
t_{WE}	Verzögerung des System-Bus $DB_\emptyset - DB_7$ nach $\overline{STSTB}$	–	30		
t_{WD}	Verzögerung vom 8080A-Bus $D_\emptyset - D_7$ zum System-Bus $DB_\emptyset - DB_7$ während des Schreibens	5	40		
t_E	Verzögerung von $\overline{\text{System-Bus-Freigabe}}$ zum System-Bus $DB_\emptyset - DB_7$	–	30		
t_{HD}	Verzögerung zwischen HLDA und $\overline{INTA}$, $\overline{IOR}$, $\overline{MEMR}$		25		
t_{DS}	Vorbereitungszeit, System-Bus Eingabe zu HLDA	10	–		
t_{DH}	Haltezeit zwischen System-Bus-Eingabe und HLDA	20			

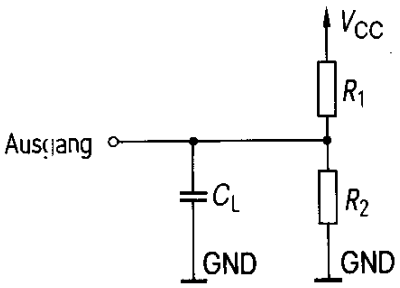
Kapazität[1])

Prüfbedingungen: $V_{vor.}=2{,}5$ V; $V_{CC}=5$ V; $T_U=25$ °C; $f=1$ MHz

Symbol	Bezeichnung	Grenzwerte			Einheit
		min.	typ.[2])	max.	
C_{IN}	Eingangskapazität	–	8	12	pF
C_{OUT}	Ausgangskapazität Steuersignale		7	15	
I/O	E/A-Kapazität (D oder DB)		8		

[1]) Dieser Parameter wird nur stichprobenmäßig kontrolliert und nicht zu 100% geprüft.
[2]) Typische Werte bei $T_U=25$ °C und der Nenn-Versorgungsspannung.

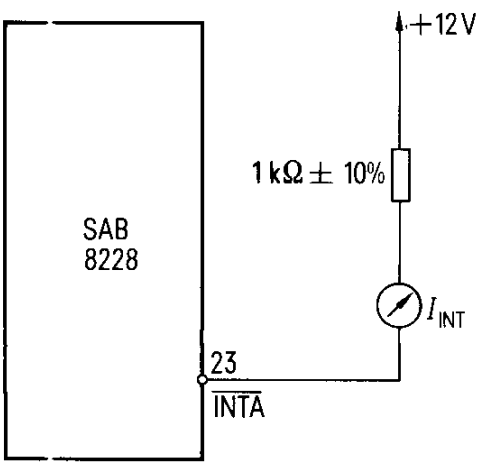
Meßschaltung



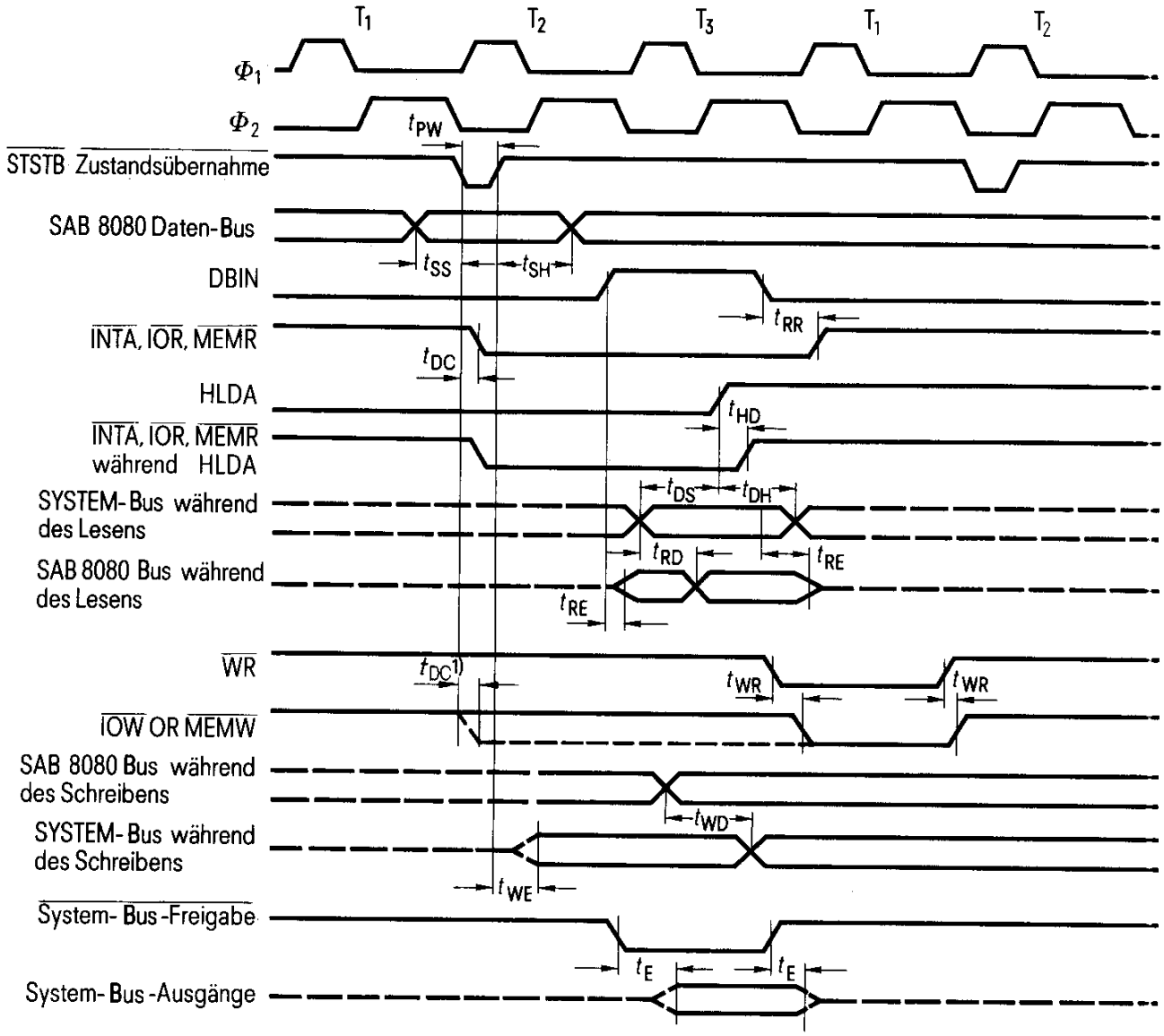
Für $D_\emptyset - D_7$: $R_1=4$ kΩ; $R_2=\infty$; $C_L=25$ pF;
Für die übrigen Ausgänge gilt:

$R_1=500\,\Omega$; $R_2=1$ kΩ; $C_L=100$ pF;

INTA-Meßschaltung (für RST7)



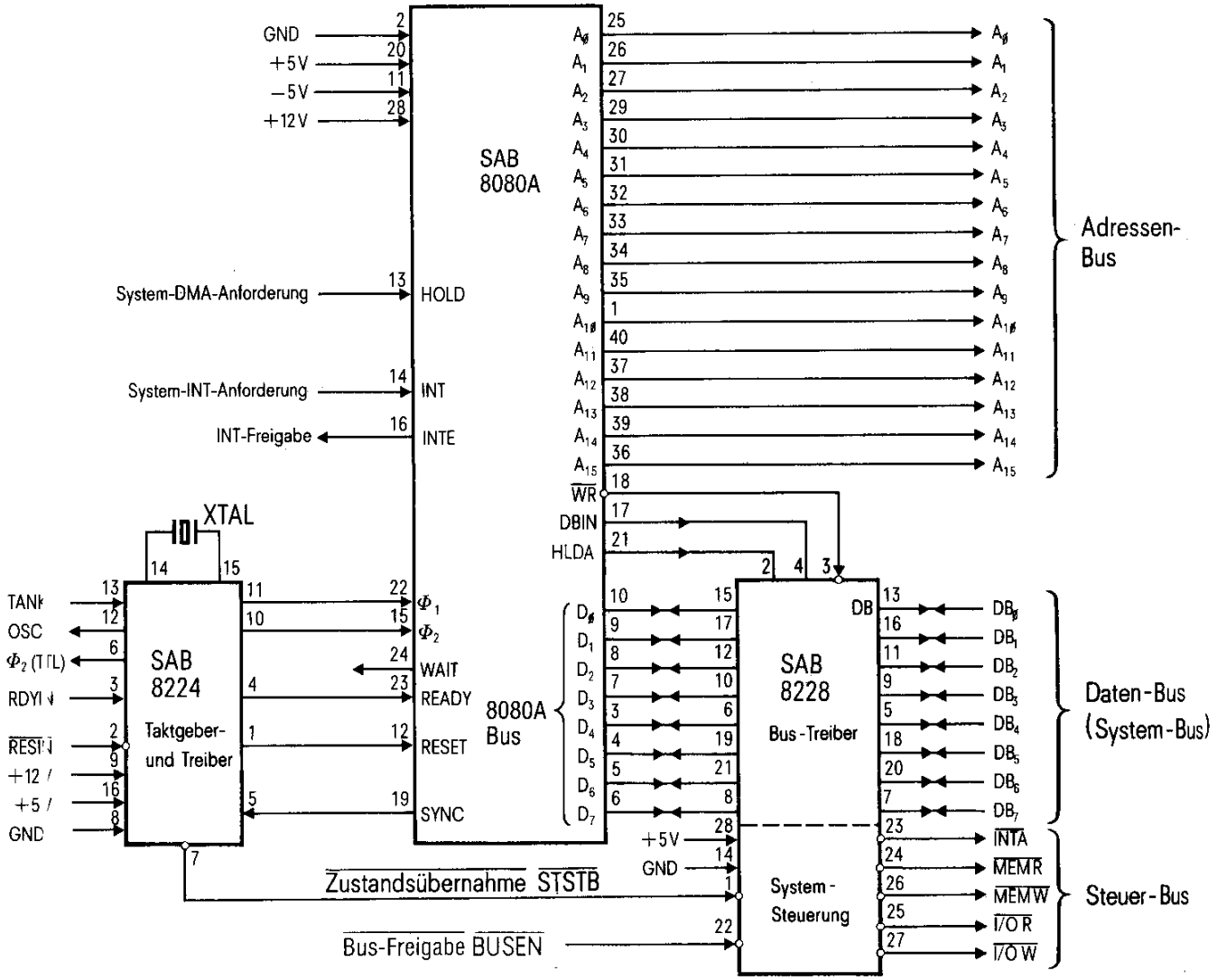
Impulsdiagramm



Spannungsmeßpunkte: $D_{ø}$–D_7 (falls Ausgänge) Logische „Ø" = 0,8 V,
Logische „1" = 3,0 V.
Alle übrigen Signale gemessen bei 1,5 V.

[1]) Vorgezogenes Steuersignal nur für SAB 8238

Standardschnittstelle für den SAB 8080A



6. MIL-Baustein-Ausführungen

Für Anwendungen mit verschärften Umweltbedingungen stehen eine Reihe von Bausteinen mit erweiterten Spezifikationen zur Verfügung. Als ein wesentlicher Punkt sei der erweiterte Temperaturbereich von −55 °C bis 100 °C und mehr hervorgehoben. Alle diese Bausteine werden nach gewissen Standardnormen qualifiziert und geprüft (z.B. MIL-STD-833 usw.).

Es sind folgende Bausteine verfügbar:

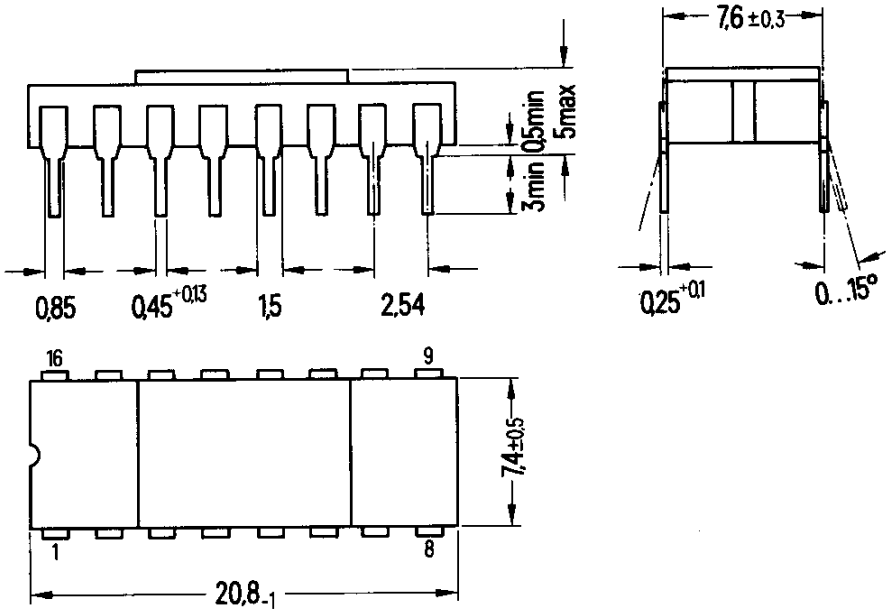
Mikrocomputer-Bausteine	EPROM-/ROM-Bausteine	RAM-Bausteine	Peripherie-Bausteine
M 8080A	M 2708	M 2102A-4	M 8212
M 8224	M 2716	M 2111A	M 8214
M 8228	M 2316E	M 2114	M 8216
		M 2142	M 8226
		M 5101-4	M 8251
		M 5101L-4	M 8255A
			M 8257
			M 8259

Dieses Angebot an Bausteinen in MIL-Ausführung wird laufend ergänzt.

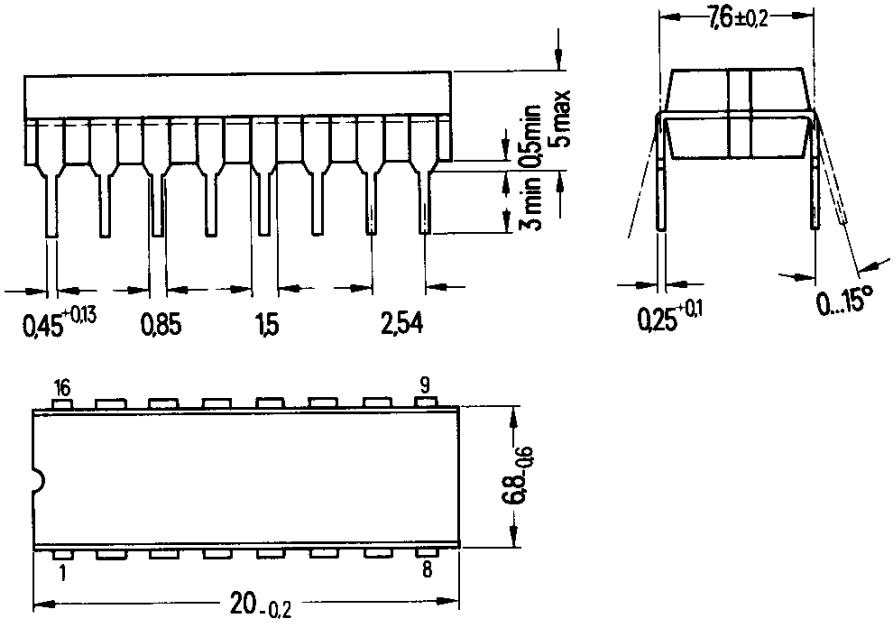
Weitere Auskünfte erteilt Ihnen gerne unsere zuständige Geschäftsstelle. Ein Verzeichnis finden Sie am Ende dieses Datenbuches.

7. Gehäuseabmessungen

16pol. Keramik-DIP-Gehäuse Typ C

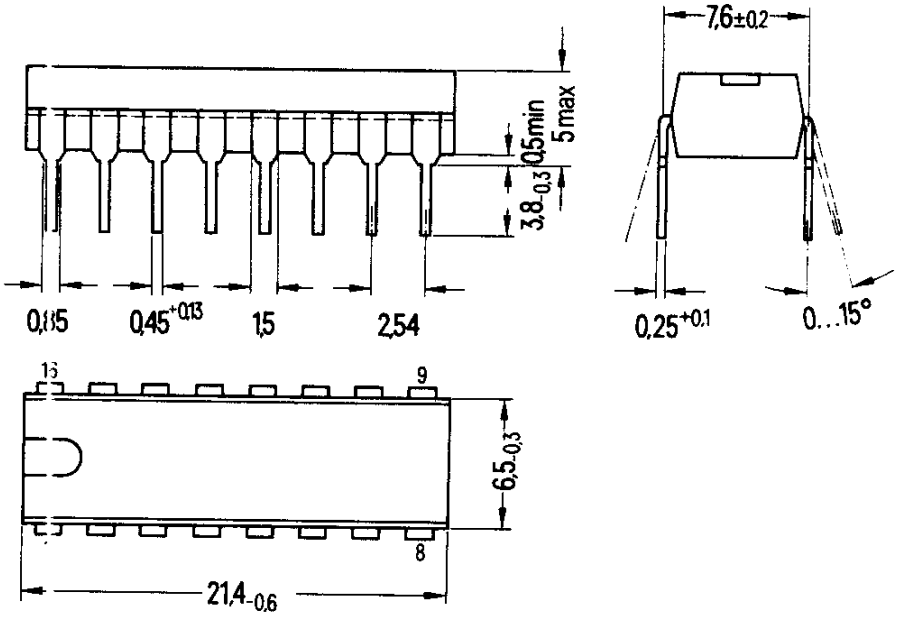


16pol. Keramik-DIP-Gehäuse Typ D



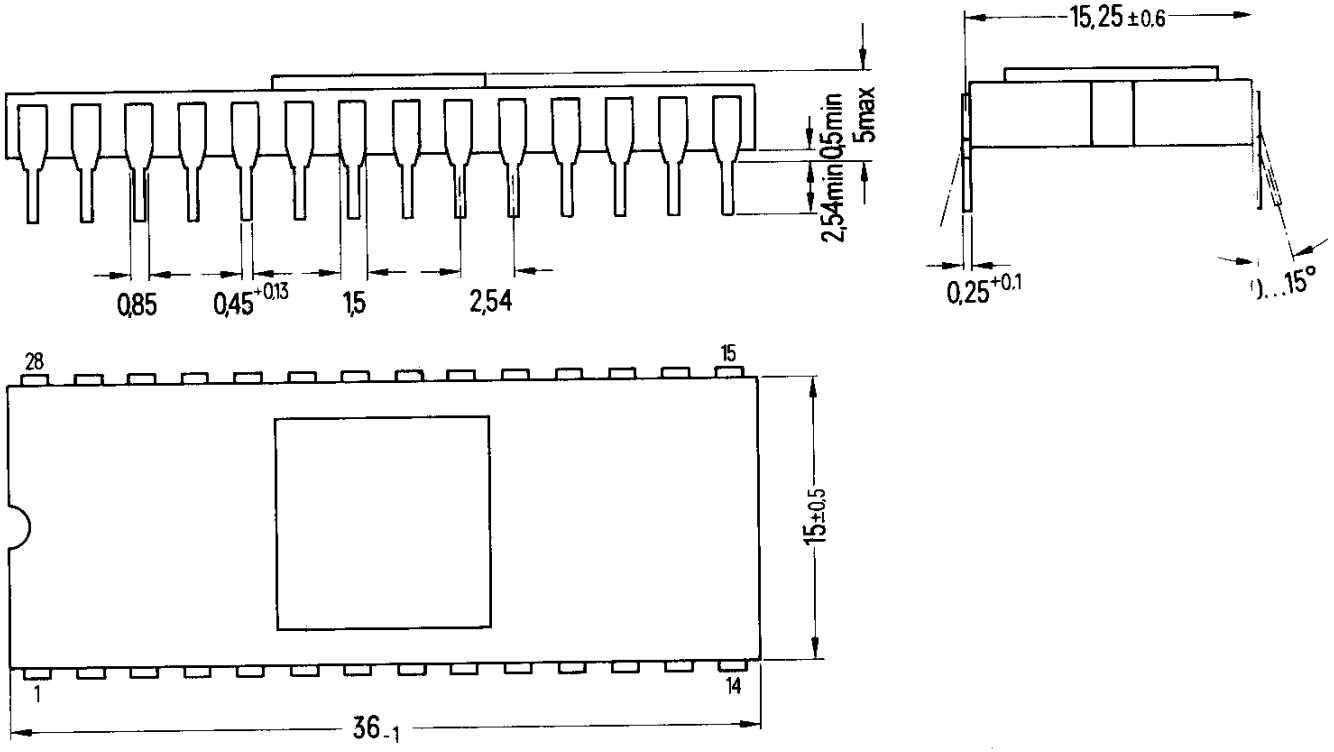
Gehäuseabmessungen

16pol. Kunststoff-DIP-Gehäuse Typ P

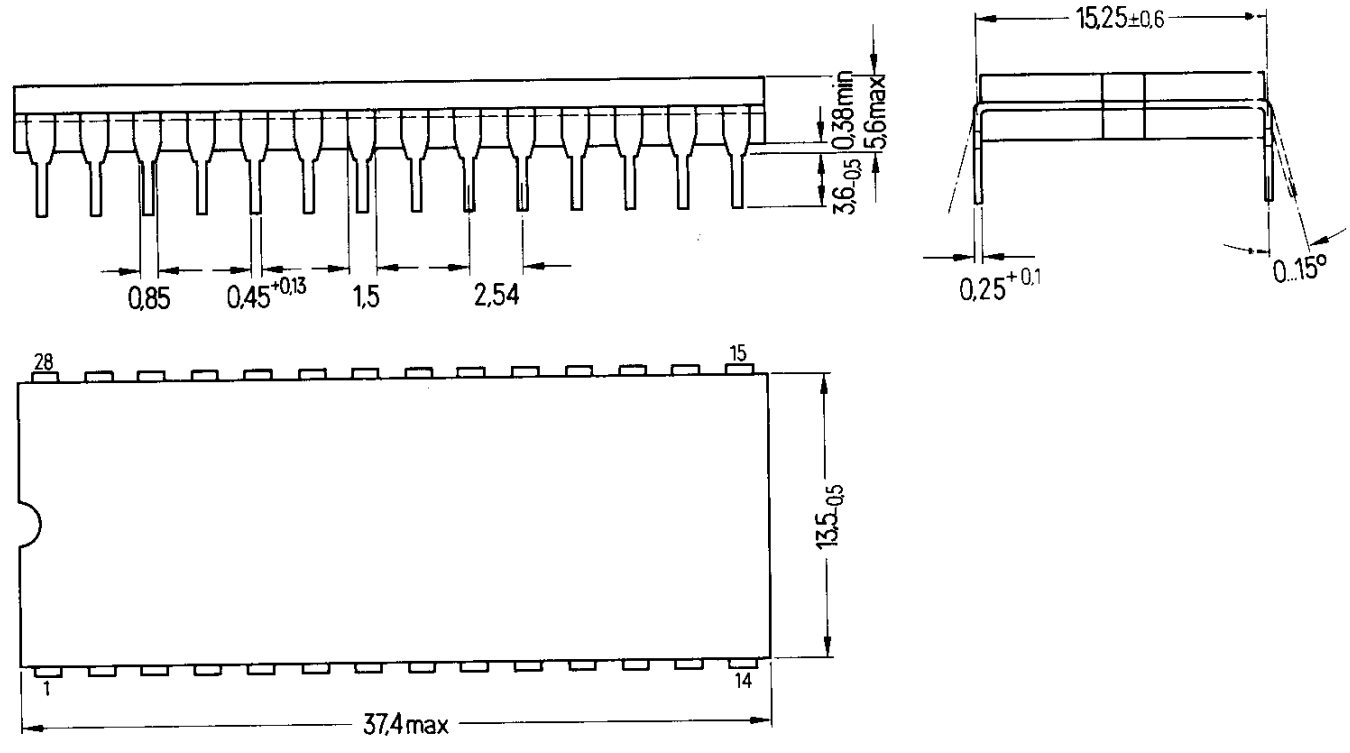


Gehäuseabmessungen

28pol. Keramik-DIP-Gehäuse Typ C

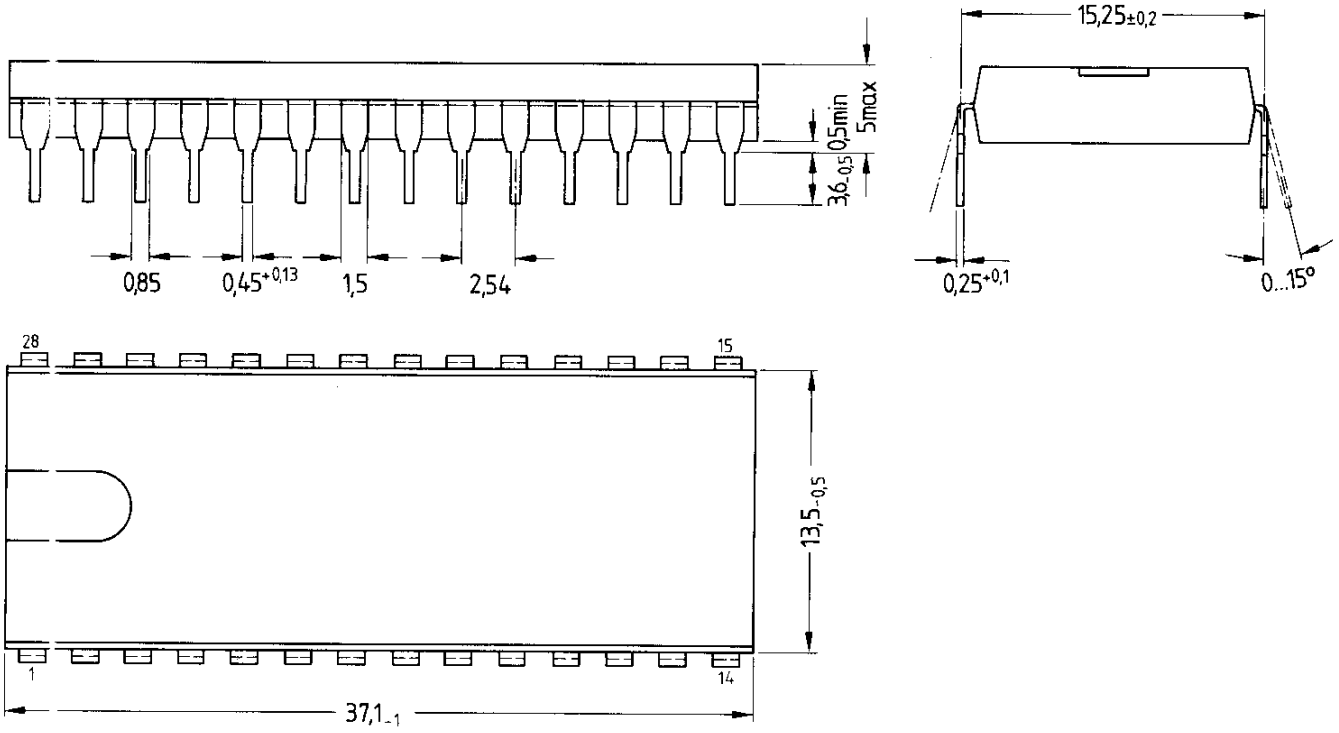


28pol. Keramik-DIP-Gehäuse Typ D

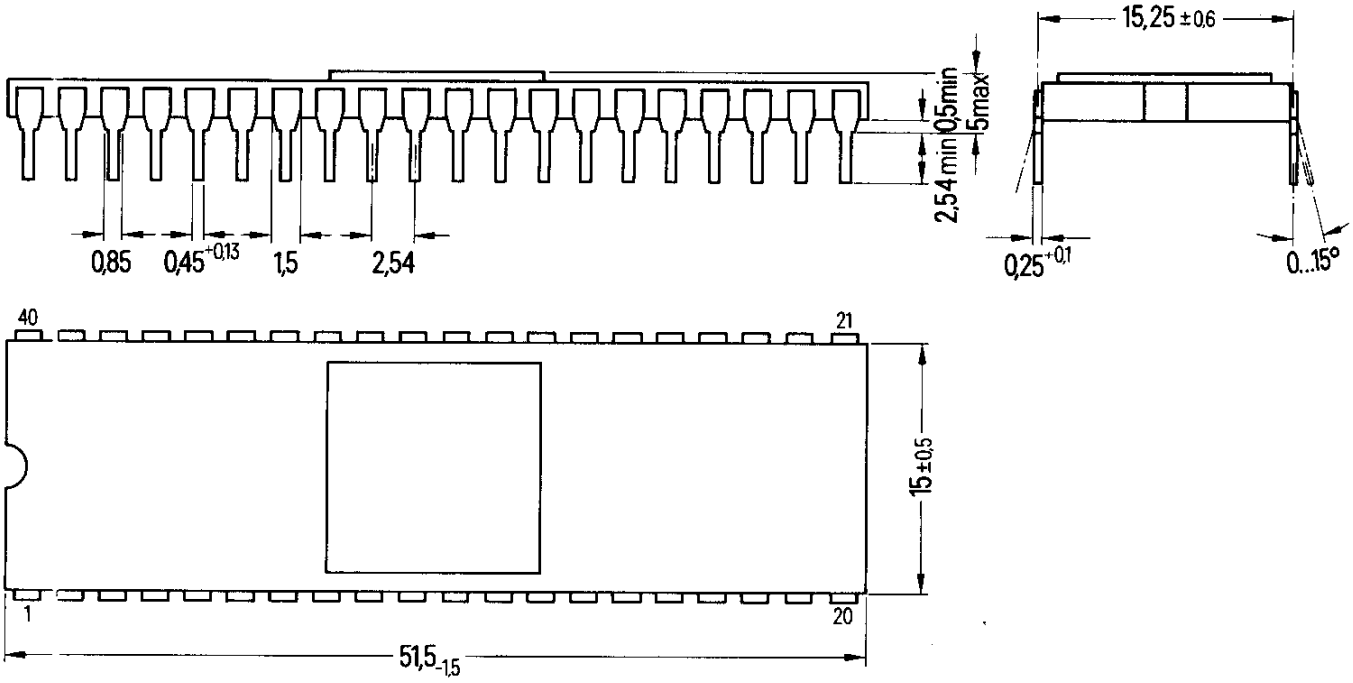


Gehäuseabmessungen

28pol. Kunststoff-DIP-Gehäuse Typ P

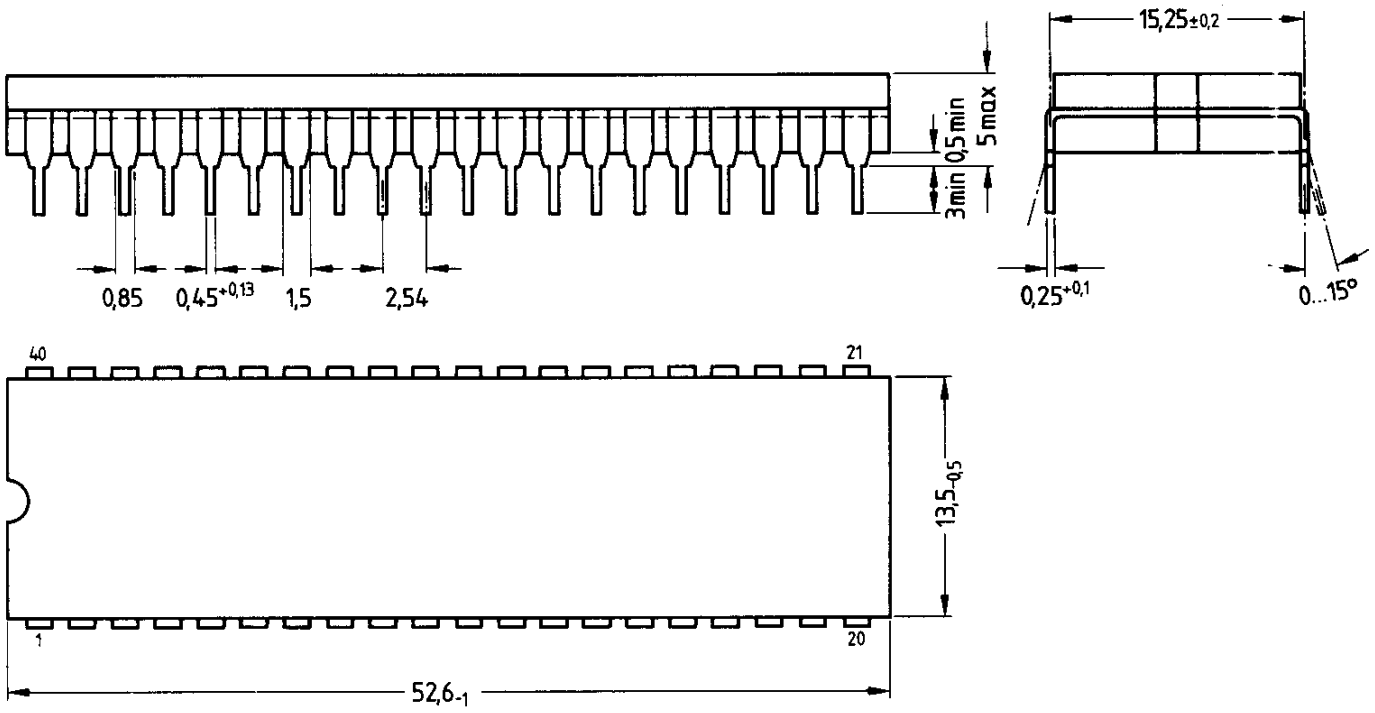


40pol. Keramik-DIP-Gehäuse Typ C

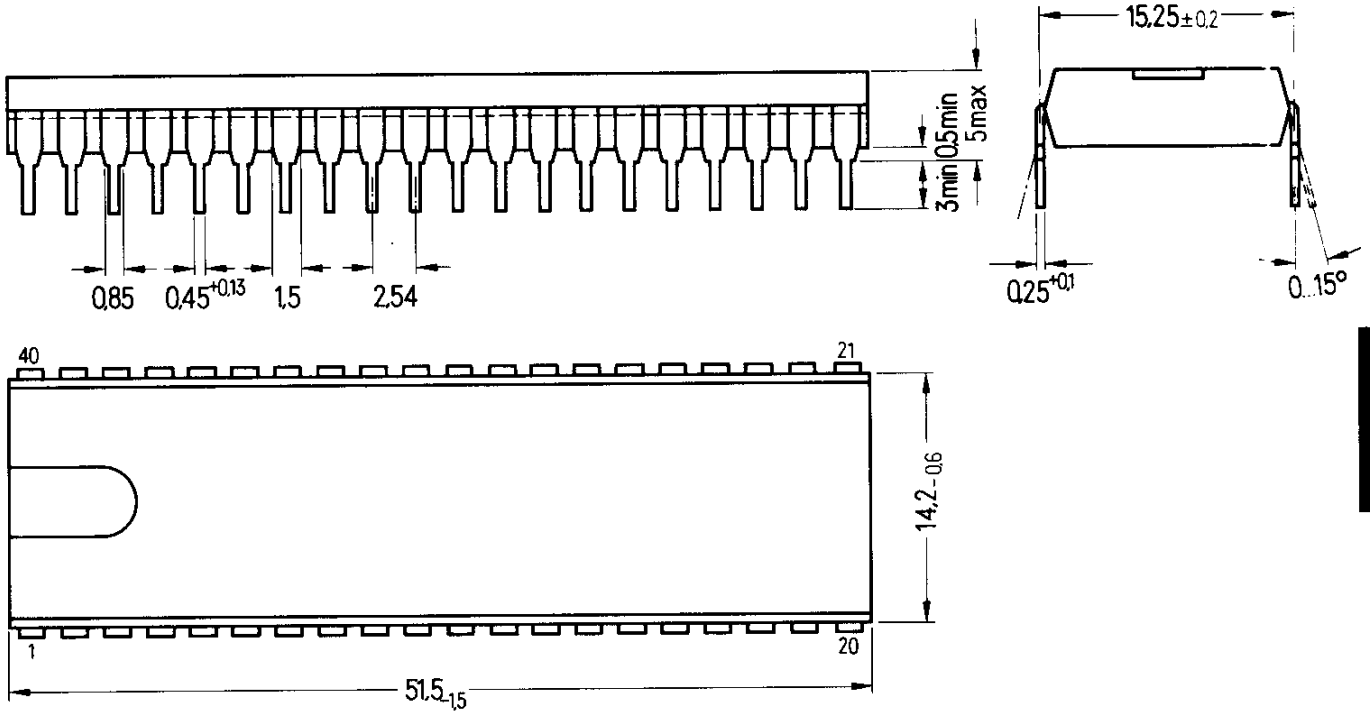


Gehäuseabmessungen

40pol. Keramik-DIP-Gehäuse Typ D



40pol. Kunststoff-DIP-Gehäuse Typ P



Unsere Geschäftsstellen

Bundesrepublik Deutschland und Berlin (West)

Siemens AG
Salzufer 6–8
Postfach 110560
1000 Berlin 11
☎ (030) 3939-1, Tx 1810-278
FAX (030) 3939-2630

Siemens AG
Contrescarpe 72
Postfach 107827
2800 Bremen 1
☎ (0421) 364-1, Tx 245451
FAX (0421) 364-687

Siemens AG
Lahnweg 10
Postfach 1115
4000 Düsseldorf 1
☎ (0211) 3030-1, Tx 8581301
FAX (0211) 3030-506

Siemens AG
Gutleutstraße 31
Postfach 4183
6000 Frankfurt 1
☎ (0611) 262-1, Tx 414131
FAX (0611) 262-2253

Siemens AG
Lindenplatz 2
Postfach 105609
2000 Hamburg 1
☎ (040) 282-1, Tx 2162721
FAX (040) 282-2210

Siemens AG
Am Maschpark 1
Postfach 5329
3000 Hannover 1
☎ (0511) 199-1, Tx 922333
FAX (0511) 199-2799

Siemens AG
N 7, 18 (Siemenshaus)
Postfach 2024
6800 Mannheim 1
☎ (0621) 296-1, Tx 462261
FAX (0621) 296-222

Siemens AG
Richard-Strauss-Straße 76
Postfach 202109
8000 München 2
☎ (089) 9221-1, Tx 529421-25
FAX (089) 9221-4499

Siemens AG
Von-der-Tann-Straße 30
Postfach 4844
8500 Nürnberg 1
☎ (0911) 654-1, Tx 622251
FAX (0911) 654-3436,
34614, 3716

Siemens AG
Geschwister-Scholl-Straße 24
Postfach 120
7000 Stuttgart 1
☎ (0711) 2076-1, Tx 723941
FAX (0711) 2076-706

Siemens Bauteile Service
Lieferzentrum Fürth
Postfach 146
8510 Fürth-Bislohe
☎ (0911) 3001-1, Tx 623818

Europa

Belgien
Siemens S.A.
chaussée de Charleroi 116
B-1060 Bruxelles
☎ (02) 5373100, Tx 21347

Bulgarien
RUEN,
Büro für Firmenvertretungen und
Handelsvermittlungen bei der
Vereinigung „Interpred"
San Stefano 14/16
BG-1504 Sofia 4
☎ 457082, Tx 22763

Dänemark
Siemens A/S
Borupvang 3
DK-2750 Ballerup
☎ (02) 656565, Tx 35313

Finnland
Siemens Osakeyhtiö
Mikonkatu 8
Fach 8
SF-00101 Helsinki 10
☎ (90), 1626-1, Tx 124465

Frankreich
Siemens S.A.
39–47, boulevard Ornano
F-93200 Saint-Denis
(B.P. 109, F-93203 Saint Denis
CEDEX 1)
(für Personalpost: B.P. 122,
F-93204 Saint-Denis CEDEX 1)
☎ (16-1) 8206120, Tx 620853

Griechenland
Siemens Hellas E.A.E.
Voulis 7
P.O.B. 601
Athen 125
☎ (01) 3293-1, Tx 216291

Großbritannien
Siemens Limited
Siemens House
Windmill Road
Sunburry-on-Thames
Middlesex TW 16 7HS
☎ (09327) 85691, Tx 8951091

Irland
Siemens Limited
8, Raglan Road
Dublin 4
☎ (01) 684727, Tx 5341

Island
Smith & Norland H/F
Nóatún 4
P.O.B. 519
Reykjavik
☎ 28322, Tx 2055

Italien
Siemens Elettra S.p.A.
Via Fabio Filzi, K 25/A
Casella Postale 4183
I-20124 Milano
☎ (02) 6248, Tx 330261

Jugoslawien
Generalexport
Masarikova 5/XIV
Poštanski fah 223
YU-11001 Beograd
☎ (011) 684866, Tx 11287

Luxemburg
Siemens Société Anonyme
17, rue Glesener
B.P. 1701
Luxembourg
☎ 49711-1, Tx 3430

Niederlande
Siemens Nederland N.V.
Wilhelmina van Pruisenweg 26
NL-2595 AN Den Haag
(Postbus 16068,
NL-2500 BB Den Haag)
☎ (070) 782782, Tx 31373

Norwegen
Siemens A/S
Østre Aker vei 90
Postboks 10, Veitvet
N-Oslo 5
☎ (02) 153090, Tx 18477

Österreich
Siemens Aktiengesellschaft
Österreich
Apostelgasse 12
Postfach 326
A-1031 Wien
☎ (0222) 7293-0, Tx 131866

Polen
PHZ Transactor S.A.
ul. Stawki 2
P.O.B. 276
PL-00-950 Warszawa
☎ 398910, Tx 815554

Portugal
Siemens S.A.R.L.
Avenida Almirante Reis, 65
Apartado 1380
P-1100 Lisboa-1
☎ (019) 538805, Tx 12563

Rumänien
Siemens birou
de consultaţii tehnice
Strada Edgar Quinet Nr. 1
R-70106 Bucureşti 1
☎ 151825, Tx 11473

Schweden
Siemens Aktiebolag
Norra Stationsgatan 69
Box 23141
S-10435 Stockholm 23
☎ (08) 241700, Tx 11672

Schweiz
Siemens-Albis AG
Freilagerstraße 28
Postfach
CH-8047 Zürich
☎ (01) 2473111, Tx 52131

Spanien
Siemens S.A.
Orense, 2
Apartado 155
Madrid 20
☎ (91) 4552500, Tx 27769

Tschechoslowakei
EFEKTIM,
Technisches Beratungsbüro
Siemens AG
Anglická ulice 22, 3. Stock
P.O.B. 1087
CS-12000 Praha 2
☎ 258417, Tx 122389

Türkei
Etmas Elektrik Tesisatve
Mühendislik A.S.
Meclisi Mebusan Caddesi,
35 Findikli
P.K. 213 Findikli
Istanbul
☎ 452090, Tx 24233

Ungarn
Intercooperation AG,
Siemens Kooperationsbüro
Böszörményi út 9–11
P.O.B. 1525
H-1126 Budapest
☎ (01) 154970, Tx 224133

Union der Sozialistischen Sowjetrepubliken
Ständige Vertretung der
Siemens AG in Moskau
Internationales Postamt
Postfach 77
SU-Moskau G 34
☎ 2027711, Tx 7413

Afrika

Ägypten
Siemens Resident Engineers
33, Dokki Street
P.O.B. 775
Dokki/Cairo
Arab Republik Egypt
☎ 982671, Tx 321

Äthiopien
Siemens Ethiopia Ltd.
P.O.B. 5505
Addis Ababa
☎ 15 1599, Tx 21052

Algerien
Siemens Algèrie S.A.R.L.
3, Viaduc Youghourta
B.P. 224, Alger-Gare
Alger
☎ 615966/67, Tx 52817

Libyen
Siemens Resident Engineers
Socialist People's Libyan Arab
Jamahiriya
P.O.B. 46
Tripoli
☎ 41534, Tx 20029

Marokko
SETEL
Société Electrotechnique
et de Télécommunications S.A.
Immeuble Siemens
km 1, Route de Rabat
Casablanca-Ain Sebâa
☎ 351025, Tx 25914

Nigeria
Siemens Nigeria Ltd.
Siemens House
Industrial estate 3 f,
Block A
P.O.B. 304, Apapa
Oshodi (Lagos)
☎ 842502, Tx 21357

Sudan
National Electrical
& Commercial Company (NECC)
P.O.B. 1202
Khartoum
Republic of Sudan
☎ 80818, Tx 642

Südafrika
Siemens Limited
Siemens House,
Corner Wolmarans and
Biccard Streets, Braamfontein 2001
P.O.B. 4583
Johannesburg 2000
☎ (011) 7159111, Tx 58-7721

Tunesien
Sitelec S.A.,
Immeuble Saâdi - Tour C
Route de l'Ariana
Tunis-El Menzah TN
☎ 231526, Tx 12326

Zaire
Siemens Zaire S.A.R.L.
B.P. 9897
6, rue Limité
Kinshasa 1
☎ 22608, Tx 21377

Amerika

Argentinien
Siemens Sociedad Anónima
Avenida Pte. Julio A. Roca 516
Casilla Correo Central 1232
RA-1067 Buenos Aires
☎ 300411, Tx 121812

Bolivien
Sociedad Comercial é Industrial
Hansa Limitada
CalleMercadoesquinaYanacocha
Cajón Postal 1402
La Paz
☎ 355317, Tx 5261

Brasilien
Siemens S.A.
Sede Central
Avenida Mutinga, 3650
Pirituba
BR-05110 São Paulo-SP
(Caixa Postal 1375,
BR-01000 São Paulo)
☎ (011) 2610211
Tx 11-23633, 11-23641

Chile
Gildemeister S.A.C.,
Area Siemens
Casilla 99-D
Santiago de Chile
☎ 82523,
Tx TRA SGO 392, TDE 40588

Ecuador
Siemens S.A.
Avenida América y
Hernández Girón s/n.,
Casilla de Correos 3580
Quito
☎ 454000, Tx 22190

Kanada
Siemens Electric Limited
7300 Trans-Canada Highway
Pointe Claire, Québec H9R 1C7
(P.O.B. 7300, Pointe Claire,
Québec H9R 4R6)
☎ (514) 6957300, Tx 5-822778

Kolumbien
Siemens S.A.
Carrera 65, No. 11–83
Apartado Aéreo 80150
Bogotá 6
☎ 2628811, Tx 44750

Mexico
Siemens S.A.
Poniente 116, No. 590
Col. Ind. Vallejo
Apartado Postal 15064
México 15, D.F.
☎ 5670722, Tx 1772700

Uruguay
Conatel S.A.
Ejido 1690
Casilla de Correo 1371
Montevideo
☎ 917331, Tx 934

Venezuela
Siemens S.A.
Apartado 3616
Caracas 101
☎ (02) 2392133, Tx 25131

Vereinigte Staaten von Amerika
Siemens Corporation
186 Wood Avenue South
Iselin, New Jersey 08830
☎ (201) 494-1000
Tx WU 844491
TWX WU 7109980588

Asien

Afghanistan

Afghan Electrical Engineering
and Equipment Limited
Alaudin, Karte 3
P.O.B. 7
Kabul 1
☎ 40446, Tx 35

Bangladesch

Siemens Bangladesh Ltd.
74, Diskusha Commercial Area
P.O.B. 33
Dacca 2
☎ 244381, Tx 5524

Hongkong

Jebsen & Co., Ltd.
Siemens Division
Prince's Building, 24th floor
P.O.B. 97
Hong Kong
☎ 5225111, Tx 73221

Indien

Siemens India Ltd.
Head Office
134-A, Dr. Annie Besant Road, Worli
P.O.B. 6597
Bombay 400018
☎ 379906, Tx 112373

Indonesien

Representative Siemens AG
Jl. Kebon Sirih 4
P.O.B. 2469
Jakarta Pusat
☎ 351051, Tx 46222

Irak

Siemens Iraq Consulting Office
P.O.B. 3120
Baghdad
☎ 98198, Tx 2393

Iran

Siemens Sherkate Sahami Khass
Ave. Ayatolla Taleghani 32
Siemenshaus
Teheran 15
☎ (021) 614-1, Tx 212351

Japan

Siemens K.K.
Sales and Administration
Gotanda Fujikura Building,
7th + 9th floor
11–20, Nishigotanda 2-chome,
Shinagawa-ku
P.O.B. 68, Osaki
Tokyo 141
☎ (03) 4902171, Tx 22808

Korea (Republik)

Siemens Electrical
Engineering Co., Ltd.
C.P.O.B. 3001
Seoul
☎ 7783431, Tx 23229

Kuwait

Abdul Aziz M. T. Alghanim Co.
& Partners
Abdulla Fahad Al-Mishan Building
Al-Sour Street
P.O.B. 3204
Kuwait, Arabia
☎ 423336, Tx 2131

Libanon

Ets. F. A. Kettaneh S.A.
(Kettaneh Frères)
Medawar
P.B. 110242
Beyrouth
☎ 251040, Tx 20614

Malaysia

Guthrie Engineering (Malaysia)
Sdn. Bhd.,
Electrical &
Communications Division
17, Jalan Semangat
P.O.B. 30
Petaling Jaya/Selangor
☎ 773344, Tx 37573

Pakistan

Siemens Pakistan Engineering
Co. Ltd.
Ilaco House, Abdullah Haroon Road
P.O.B. 7158
Karachi 3
☎ 516061, Tx 2820

Philippinen

Engineering Equipment, Inc.
Machinery Division,
Siemens Department
E. Rodriguez Avenue
Murphy, Quezon City
P.O.Box 7160, ADC-MIA 3120
☎ 773011,
Tx RCA 7222382, EEC 3695

Saudi-Arabien

Arabia Electric Ltd.
Head Office
P.O.B. 4621
Jeddah
☎ 59521, Tx 401864

Singapur

Siemens Components PTe. Ltd.
19B – 45B, Jalan Tenteram
Singapore 12
☎ 550811, Tx RS 21000

Syrien

Syrian Import
Export & Distribution
Co., S.A.S. SIEDCO
Port Saïd Street
P.O.B. 363
Damas
☎ 1343133, Tx 11267

Taiwan

Delta Engineering Ltd.
42, Hsu Chang Street, 8th floor
P.O.B. 58497
Taipei
☎ 3114731, Tx 21826

Thailand

B. Grimm & Co., R.O.P.
1643/4,Phetburi Road
(Extension)
G.P.O.B. 66
Bangkok 10
☎ 2524081, Tx 2614

Yemen (Arab. Republik)

Tihama Tractors
& Engineering Co. Ltd.
P.O.B. 49
Sanaa
Yemen Arab Republic
☎ 2462, Tx 2217

Australien

Australien

Siemens Industries Limited
544 Church Street, Richmond
Melbourne, Vic. 3121
☎ (03) 4297111, Tx 30425

SIEMENS

Bestell-Nr. B 1950
Printed in Germany
KG 05805.