10 Mikroprozessoren und Mikrocomputer

10.1 Vorbemerkungen

Wie bereits aus dem vorhergehenden Abschnitt bekannt ist, besteht die Hardware einer DV-Anlage (eines Computersystems) aus der **Zentraleinheit** (dem eigentlichen Computer) und der **Peripherie** (Abb. 10.1). Zur Peripherie gehören hauptsächlich alle Arten von Ein- und Ausgabegeräten und die Massenspeicher. Die Datenverarbeitung findet in der Zentraleinheit, dem Computer, statt. Die Zentraleinheit besteht aus dem **Leitwerk** – es steuert den gesamten Ablauf der Datenverarbeitung –, dem **Rechenwerk** – hier werden die Daten verarbeitet –, dem **Speicher** – in ihm sind die Programme und Daten abgelegt – und zusätzlich einer **Ein-/Ausgabesteuerung** – über sie läuft der Datenverkehr zwischen Peripherie und der Zentraleinheit (Abb. 10.2). Das Rechenwerk, in dem neben arithmetischen auch logische und Verschiebeoperationen ablaufen können, wird als **ALU** (**A**rithmic und **L**ogic **U**nit) bezeichnet. Rechenwerk und Leitwerk zusammen bilden den **Zentralprozessor,** auch **CPU** (**C**entral **P**rocessing **U**nit) genannt.

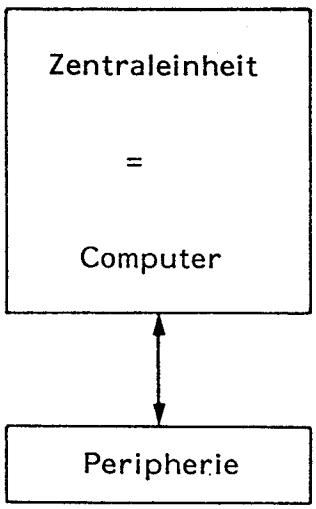


Abb. 10.1 – Bestandteile einer DV-Anlage

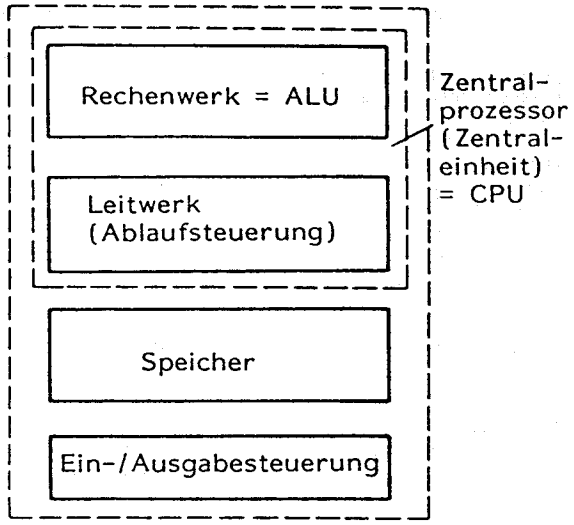


Abb. 10.2 – Bestandteile der Zentraleinheit

10.2 Mikroprozessor, Mikrocomputer und Mikrocontroller

10.2.1 Mikroprozessor

Ein Mikroprozessor ist ein in einem IC integrierter Zentralprozessor. Er erfüllt also die Funktion eines Leitwerks und eines Rechenwerks. Die ersten Mikroprozessoren kamen 1971 auf den Markt. Sie wurden sehr schnell weiterentwickelt. Mit ihrer Leistungsfähigkeit steigen auch die Einsatzmöglichkeiten.

10.2.2 Mikrocomputer

Ein Mikrocomputer ist dadurch gekennzeichnet, daß er aus wenigen hochintegrierten IC besteht und als Zentralprozessor (CPU) einen Mikroprozessor verwendet. Im Aufbau und in der Arbeitsweise unterscheidet er sich grundsätzlich nicht von einem „normalen" Computer, wie ein Vergleich zwischen Abb. 10.2 und 10.3 zeigt: In beiden Fällen besteht der Computer aus der CPU, dem Speicher und der Ein-/Ausgabe. Bei einem Mikrocomputer ist ein Teil des Arbeitsspeichers ein Festwertspeicher (ROM, PROM oder EPROM), der andere ein Schreib-/Lesespeicher (RAM).

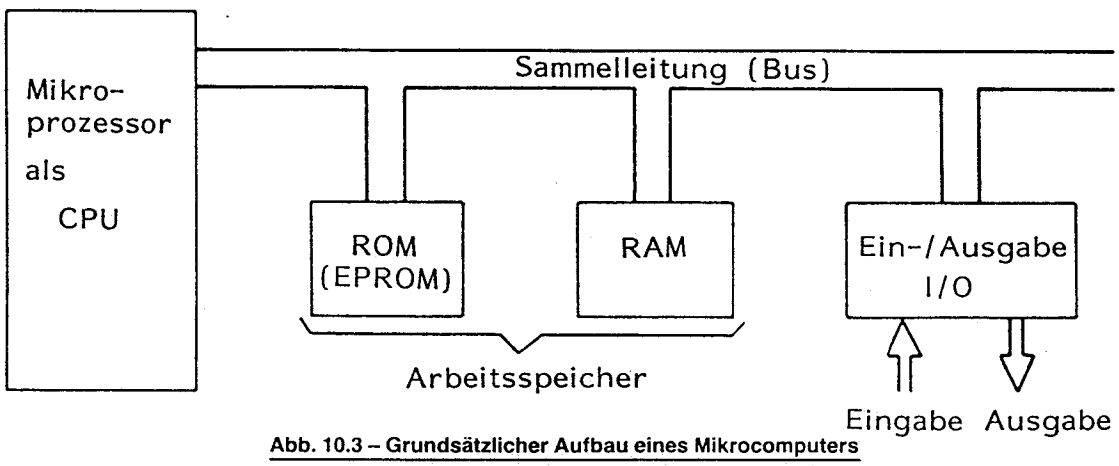


Abb. 10.3 – Grundsätzlicher Aufbau eines Mikrocomputers

Die Anwendungen der Mikrocomputer lassen sich in zwei große Bereiche einteilen:

a) Ersatz für festverdrahtete Hardware
In diesem Anwendungsbereich ersetzen Mikrocomputer Steuerungen, die vorher aus Zählern, Registern und Verknüpfungsschaltungen aufgebaut waren. Die Mikrocomputer bewirken hier hauptsächlich folgende Vorteile: Erstens enthalten die Steuerungen nur noch sehr viel weniger Bauelemente, wodurch sie billiger und zuverlässiger werden, und zweitens ist so eine Steuerung viel flexibler. Eine Mikroprozessorsteuerung kann einfach durch Programmänderung unter Beibehaltung der Hardware einer sich ändernden Aufgabenstellung angepaßt werden. Mikroprozessorsteuerungen findet man in allen Bereichen, so z. B. in Waschmaschinen, preisrechnenden Waagen, Fernsehspielgeräten, elektronischen Einspritzanlagen für Kraftfahrzeuge, Prüf- und Meßgeräten, Datenendgeräten und vor allem auch in DV-Anlagen für Untersteuerungen wie z. B. die Ein-/Ausgabesteuerung.

b) Frei programmierbare Rechner
Mikroprozessoren sind so weit entwickelt, daß sich mit ihnen sehr kleine, preiswerte, aber trotzdem sehr leistungsfähige Computer aufbauen lassen.

Soll der Mikrocomputer als Steuercomputer eingesetzt werden, werden die Anwenderprogramme im Festwertspeicher untergebracht. Bei Großserien mit unveränderbaren Abläufen werden ROM verwendet. Bei kleinen Serien oder wenn ein Programm von Zeit zu Zeit modernisiert werden soll, sind EPROM besser geeignet.

Bei Verwendung als frei programmierbarer Rechner wurde früher das Betriebssystem (z. B. MS-DOS) im Festwertspeicher deponiert. Bei einigen Home-Computern ist das auch heute noch der Fall. Beim PC, dem Personalcomputer (ursprünglich: **persönlicher Computer**), geht man allerdings meistens einen anderen Weg. Nach dem Einschalten des Geräts wird das Betriebssystem von der Festplatte oder von Disketten in den RAM (Arbeitsspeicher) geladen. Im ROM-Bereich befinden sich nur die absolut erforderlichen Startprogramme, wie z. B. das Basis-Ein-Ausgabeprogramm des Computers. Das hat den Vorteil, daß der Benutzer bei Bedarf ein moderneres Betriebssystem einspielen kann. Auch die wechselnden Anwenderprogramme und die Daten werden im RAM gespeichert.

Ein weiteres Kennzeichen eines Mikrocomputers sind die Sammelleitungen, über die alle Funktionseinheiten untereinander verbunden sind. Über Sammelleitungen versorgt der Mikroprozessor die einzelnen Einheiten mit Adressen und Steuersignalen, und es laufen über sie in beiden Richtungen die Daten.

Die Ein-/Ausgabe, häufig als I/O (Input/Output) bezeichnet, besteht aus Schaltungen, die den Mikrocomputer befähigen, mit peripheren Geräten wie Tastaturen, Anzeigen, Massenspeichern, Stellgliedern usw. Daten auszutauschen.

Eine wichtige Bedeutung der Mikrocomputer liegt darin, daß mit ihnen so preiswerte und kleine DV-Anlagen auf den Markt gekommen sind, daß jetzt viele Arbeitsplätze mit Computern ausgestattet werden können. Anfangs, als die Computer aus Relais, Röhren, Transistoren und auch noch als sie aus IC mit geringerem Integrationsgrad (SSI = small scale integration) hergestellt wurden, waren sie so groß und teuer (mehrere Millionen DM), daß sie nur von Großbetrieben und Behörden für umfangreiche Aufgaben eingesetzt wurden. Als Mitte der sechziger Jahre der Integrationsgrad erhöht werden konnte (MSI = medium scale integration), entstanden die sogenannten Minicomputer. Sie waren sehr viel kleiner und billiger (einige 10000,– bis

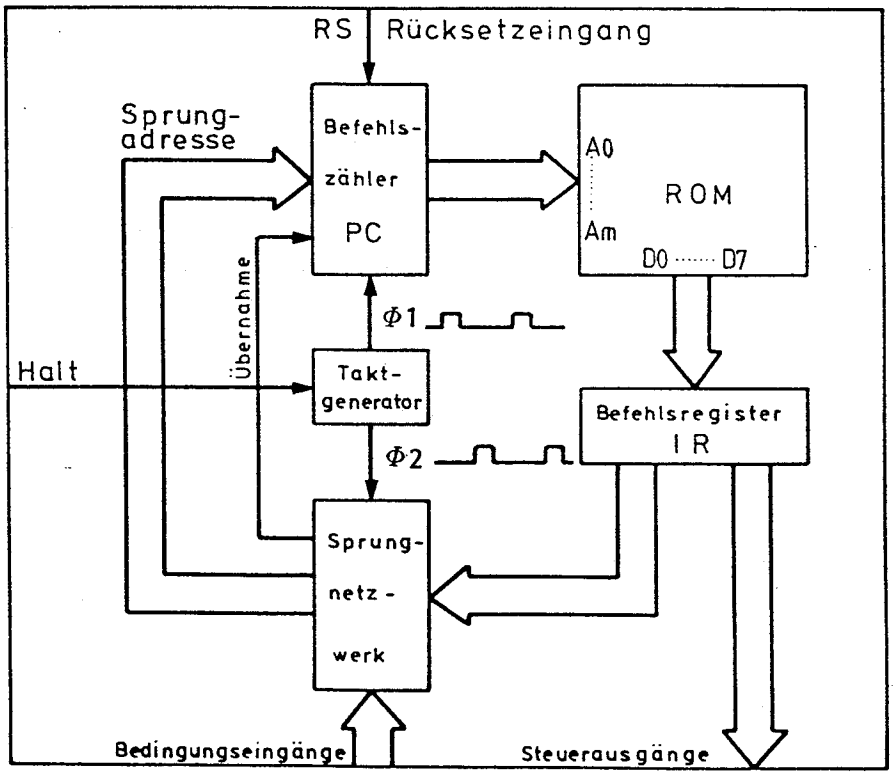


Abb. 10.4 – Mikrocontroller

wenige 100000,– DM), so daß die Zahl der Anwendungen, für die der Computer wirtschaftlich eingesetzt werden konnte, schnell anstieg. Nachdem der Schritt zur Großschaltkreisintegration (LSI = large scale integration und VLSI = very large scale integration) geschafft war, wurden die Mikrocomputer möglich. Sie haben z. T. schon die Leistungsfähigkeit von Minicomputern erreicht bei sehr viel geringeren Kosten. LSI- und VLSI-IC wie Mikroprozessoren, ROM und RAM mit großer Kapazität werden natürlich auch für Großrechenanlagen eingesetzt, wodurch diese entweder bei gleicher Leistung kleiner und billiger oder noch schneller und leistungsfähiger gemacht werden.

10.2.3 Mikrocontroller

Ein Mikrocontroller (Mikroprogrammsteuerwerk) enthält wie ein Mikroprozessor keine neuartigen Schaltungen. Auch seine Bedeutung liegt darin, daß eine umfangreiche Schaltung in einem IC integriert und damit billiger wurde. In ihm ist ein kompletter Computer auf einem Chip integriert (Ein-Chip-Computer). Somit beinhaltet er die CPU, den Speicher und einige Ein-/Ausgabebausteine. Die Hersteller umgeben einen Kern – meistens CPU, Speicher und parallele E/A-Bausteine – je nach Anwendungsfall mit A/D- und/oder D/A-Wandlern, Timern, seriellen Schnittstellen usw.

Diese Mikrocontroller sind überwiegend für Steuerungsaufgaben gedacht. Daher enthalten Sie nur relativ wenig Speicherkapazität. Häufig angeboten werden Bausteine mit 4-KByte-ROM und 256-Byte-RAM. Im ROM ist das Programm für die Steuerungsabläufe enthalten. Teilweise werden auch Mikrocontroller mit integriertem EPROM angeboten. Der Anwender kann seine Steuerungsprogramme selbst eingeben und später auch wieder ändern. Eine andere Art sind die Prototypversionen, die keinen Festwertspeicher enthalten.. Dafür besitzen sie eine „Rucksackfassung" auf dem Gehäuse, in die ein Standard-EPROM eingesetzt werden kann. Sie sind zur Programmentwicklung gedacht; erst wenn die Steuerung einwandfrei funktioniert, werden sie durch Mikrocontroller ersetzt, in die das getestete Programm implementiert wurde. Der Ablauf der Steuerung kann über Bedingungseingänge beeinflußt werden. Entsprechend dem Programmablauf werden die an den Steuerausgängen angeschlossenen Schaltungen gesteuert. Abb. 10.4 zeigt den möglichen inneren Aufbau eines Mikrocontrollers.

Der **Befehlszähler** (**program counter PC**) wird vom Taktgenerator fortlaufend hochgezählt. Der Zählerstand dient zur Ansteuerung der Adreßeingänge A0 bis Am des ROM. Im ROM sind in direkter Folge die Befehle abgelegt. Durch die Zähleransteuerung wird ein Befehl nach dem anderen ausgelesen und im **Befehlsregister** (**instruction register IR**) zwischengespeichert. Die im Befehl enthaltenen Steuersignale werden an die Steuerausgänge weitergeleitet.

Bei dem in Abb. 10.4 dargestellten Mikrocontroller gibt es mehrere Möglichkeiten, von außen den Programmablauf zu beeinflussen. Über den Eingang RS kann der Befehlszähler auf Null gesetzt werden, so daß die Steuerung von vorn anfängt. Die gesteuerte Schaltung kann über den Halt-Eingang den Taktgenerator sperren, damit den Programmablauf unterbrechen und ihn zum geeigneten Zeitpunkt wieder freigeben. Über die Bedingungseingänge sind Programmverzweigungen in Abhängigkeit vom Zustand der äußeren Schaltung möglich. Dafür enthält das im ROM abgelegte Programm bedingte Sprungbefehle. Das Sprungnetzwerk prüft jedesmal, ob es sich bei dem Bitmuster im Befehlsregister um einen bedingten Sprungbefehl handelt und ob die Sprungbedingung erfüllt ist, d. h. der entsprechende Bedingungseingang angesteuert ist. Ist beides der Fall, so wird die im Bitmuster enthaltene Sprungadresse zum Befehlszähler durchgeschaltet und durch einen Übernahmebefehl in ihn eingeschrieben. Der Programmablauf wird dann unter dieser Adresse im ROM fortgesetzt. Bei nicht erfüllter Bedingung läuft das Programm an der alten, im Befehlszähler stehenden Adresse weiter.

Der Taktgenerator in Abb. 10.4 erzeugt zwei gegeneinander phasenverschobene Takte $\Phi 1$ und $\Phi 2$. Mit $\Phi 1$ wird der Befehlszähler weitergeschaltet. $\Phi 2$ aktiviert jeweils zwischen zwei Impulsen von $\Phi 1$ das Sprungnetzwerk zur Prüfung der Sprungbedingungen.

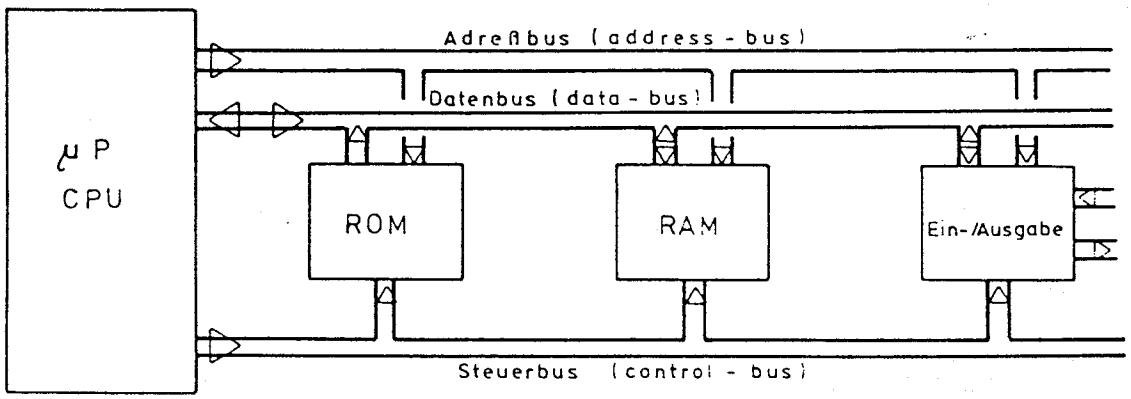


Abb. 10.5 – Bus-Systeme im Mikrocomputer

10.3 Sammelleitungssysteme eines Mikrocomputers

Wie schon im vorigen Abschnitt erwähnt, sind die Sammelleitungssysteme (Bus-Systeme) ein typisches Kennzeichen der Mikrocomputer – übrigens auch schon der Minicomputer. Ein Bus ist ein Leitungsbündel, an das alle Funktionseinheiten des Mikrocomputers parallel angeschaltet sind. Der Datenverkehr läuft immer zwischen dem Mikroprozessor (der CPU) und einer von ihm ausgewählten Funktionseinheit (ROM, RAM oder I/O). Wie Abb. 10.5 zeigt, gibt es in einem Mikrocomputer drei Bus-Systeme.

10.3.1 Datenbus (data-bus)

Über den Datenbus wird der gesamte Datenverkehr in einem Mikrocomputer abgewickelt. Er ist bidirektional, d. h., Daten laufen über ihn sowohl zum Mikroprozessor (z. B. bei Eingabe oder Lesen aus dem Speicher) als auch vom Mikroprozessor weg (z. B. bei Ausgabe oder Schreiben in den Speicher).

Da alle Funktionseinheiten parallel an den Datenbus angeschaltet sind, müssen ihre Anschlüsse schaltbar sein, denn es darf gleichzeitig immer nur eine Einheit Daten auf den Datenbus senden, und diese Daten sind immer für einen ganz bestimmten Empfänger gedacht. Die mit dem Datenbus verbundenen Schaltungen haben daher Ausgänge mit offenem Kollektor (OC) oder meist TRI-STATE-Ausgänge (TS).

Eine wichtige Kenngröße für einen Mikroprozessor ist die Breite des Datenbusses, d. h. die Zahl der Leitungen im Datenbus und damit die Zahl der parallel übertragenen Bit. Bei vielen Prozessoren stimmt sie überein mit der Zahl der parallel verarbeiteten Bit im Prozessor selbst. Gängige Mikroprozessoren arbeiten z. Z. mit einer Datenwortlänge von 8, 16 und 32 Bit. Die 16-Bit-Prozessoren waren lange Zeit der Standard bei den Personalcomputern, werden aber immer mehr von den 32-Bit-Typen abgelöst. 8-Bit-Prozessoren werden für einfache Steuerungsaufgaben verwendet. Seit einiger Zeit sind auch schon 64-Bit-Mikroprozessoren im Angebot.

10.3.2 Adreßbus (address-bus)

Der Adreßbus wird nur in einer Richtung (unidirektional) betrieben. Auf ihm sendet der Mikroprozessor die Adressen zu den Funktionseinheiten, um eine bestimmte Speicherzelle oder ein bestimmtes Ein-/Ausgangstor zu kennzeichnen. Die Breite des Adreßbusses bestimmt den Speicherbereich, den der Mikroprozessor direkt adressieren kann. Die Adreßbusbreite liegt bei den derzeit gängigen Mikroprozessoren (µP) zwischen 16 und 32 Bit. Das entspricht einem Adreßbereich von 2^{16} = 64 KByte bis 2^{32} = 4 GByte.

Typ Hersteller	Interner Datenbus	Externer Datenbus	Adreßbus	adressierbarer Speicherbereich
Z 80 ZILOG	8 Bit	8 Bit	16 Bit	64 KByte
8085 INTEL	8 Bit	8 Bit	16 Bit	64 KByte
8088 INTEL	16 Bit	8 Bit	20 Bit	1 MByte
8086 INTEL	16 Bit	16 Bit	20 Bit	1 MByte
80286 INTEL	16 Bit	16 Bit	24 Bit	16 MByte
80386SX INTEL	32 Bit	16 Bit	24 Bit	16 MByte
80386DX INTEL	32 Bit	32 Bit	32 Bit	4 GByte
i486SX (80486SX) INTEL	32 Bit	32 Bit	32 Bit	4 GByte
i486DX (80486DX) INTEL	32 Bit	32 Bit	32 Bit	4 GByte
68008 MOTOROLA	32 Bit	8 Bit	20 Bit	1 MByte
68000 MOTOROLA	32 Bit	16 Bit	24 Bit	16 MByte
68020 MOTOROLA	32 Bit	32 Bit	32 Bit	4 GByte
68030 MOTOROLA	32 Bit	32 Bit	32 Bit	4 GByte
68040 MOTOROLA	32 Bit	32 Bit	32 Bit	4 GByte

Tabelle 10.1 – Busbreiten von Mikroprozessoren

10.3.3 Steuerbus (control-bus)

Über den Steuerbus bestimmt der Mikroprozessor, mit wem er gerade kommunizieren will. Dafür enthält der Steuerbus Leitungen mit Namen wie MEM-R/W (Speicher Lesen/Schreiben), RAM-R/W (RAM Lesen/Schreiben), I/O-R/W (Ein-/Ausgabe Lesen/Schreiben), zum Teil auch für Lesen (R = read) und Schreiben (W = write) getrennte Leitungen. Diese Leitungen führen vom Mikroprozessor auf die Eingänge CE, CS, R/W, WE (write enable = Schreibfreigabe) o. ä. der Speicher- und I/O-Bausteine. Wenn der Speicher mehrere IC enthält, müssen die Steuerleitungen z. T. mit Adreßleitungen verknüpft werden, damit nur das IC, in dem die adressierte Speicherzelle liegt, aktiviert wird. Die Mikroprozessoren liefern häufig nicht direkt die Signale für den Steuerbus. Es ist dann ein Steuernetzwerk zwischen den Mikroprozessor und den Steuerbus zu schalten, das die Steuerbussignale erzeugt.

Die bisher aufgeführten Steuerbusleitungen übertragen Signale vom Mikroprozessor zu den übrigen Funktionseinheiten. Daneben gibt es Steuerleitun-

gen in umgekehrter Richtung, die z. B. für die Speichersynchronisation, für den direkten Speicherzugriff, für die Unterbrechungssteuerung und für die Einzelschrittsteuerung erforderlich sind (s. Abschn. 10.5).

10.4 Interne Organisation eines Mikroprozessors

Es gibt eine ganze Reihe von Firmen, die Mikroprozessoren herstellen. Viele von ihnen haben mehrere Typen im Programm. Diese verschiedenen Mikroprozessoren unterscheiden sich nicht nur in der Breite des Datenbusses und im adressierbaren Speicherbereich, auch ihre interne Organisation ist zum Teil sehr unterschiedlich. Durch die Erfindung des Planartransistors (1958) wurde die Entwicklung der Mikroprozessoren eingeleitet. Es wurde dadurch ein Weg ermöglicht, auf einem Chip mehrere Bauteile unterzubringen (zu integrieren).

1971 gelang es der Firma INTEL, die erste vollintegrierte 4-Bit-Zentraleinheit mit der Bezeichnung 4004 auf den Markt zu bringen. Ein Jahr später folgte schon der 8008. Dabei handelte es sich um eine 8-Bit-CPU. Im Jahre 1974 brachte INTEL eine verbesserte Version vom 8008 heraus, den 8080. Dieser Mikroprozessor hatte schon einen umfangreichen Befehlssatz und wurde von anderen Firmen in Lizenz nachgebaut oder (wie von der Firma ZILOG) weiterentwickelt (Z80).

Auch INTEL entwickelte den 8080 weiter und integrierte in den Mikroprozessor 8085 ehemals externe Bausteine, die zum Betreiben des 8080 notwendig waren. In den 70er Jahren hatte sich der 8085 einen festen Platz in der Steuerungstechnik erworben. Man verwendete diesen Prozessor oder ähnliche (z. B. weit verbreitet den Z80) auch in den ersten Personalcomputern. Dort, anders als bei den Steuerungen, merkte man jedoch sehr schnell, daß die 8-Bit-Prozessoren eine relativ geringe Arbeitsgeschwindigkeit besitzen. Daraufhin wurde in den Entwicklungslabors der großen Hersteller mit der Konstruktion von leistungsfähigeren Bausteinen begonnen. Man legte dabei das Augenmerk auf

- die Vergrößerung der parallelen Verarbeitungsbreite,
- die Erhöhung der Verarbeitungsgeschwindigkeit,
- die Erweiterung des Befehlsvorrats,
- die Vergrößerung des adressierbaren Speicherbereichs und
- die Fähigkeit, mehrere Operationen parallel durchführen zu können.

Seit 1988 hat man allerdings eine Phase erreicht, in der man eventuell an die Grenzen des mechanisch und physikalisch Machbaren gestoßen ist. Eine weitere Beschleunigung der sequentiell arbeitenden Rechner (Befehle werden nacheinander bearbeitet) war kaum noch möglich. So wurden die Konstrukteure gezwungen, sich früher oder später mit neuen Rechnerarchitekturen zu beschäftigen. Sie orientieren sich seitdem in zwei Richtungen. Einige erkunden vollkommen neue Wege, andere versuchen durch weitere Anpassung an heute mögliche technische Neuerungen optimale Ergebnisse zu erzielen.

In der **CISC-Technik** (**C**omplex **I**nstruction **S**et **C**omputer) wird mit komplexen Befehlssätzen und einem umfangreichen Befehlsvorrat (bis zu mehreren tausend verschiedenen Befehlen) gearbeitet. Musterbeispiele sind dort die 680X-Prozessoren von MOTOROLA und die 80X86-Prozessoren von INTEL. Sie besitzen für ihren Befehlsvorrat ein implementiertes (eingebautes) Befehls-ROM, in dem jeder einzelne Befehl in Form eines Mikroprogramms gespeichert ist. Diese Befehle müssen von der Ablaufsteuerung (Leitwerk) teilweise in über 300 kleinen Schritten (Maschinenzyklen) abgearbeitet werden. Seit einiger Zeit weiß man, daß solche komplexen Befehle von den meisten Programmierern kaum genutzt werden. Es zeigte sich, daß beim Ablauf von Hochspracheprogrammen ein großer Teil der Zeit mit dem Laden von internen Prozessorregistern aus dem Arbeitsspeicher und dem Zurückladen in den Arbeitsspeicher zugebracht wird. Daher werden von vielen typischen Anwendungsprogrammen ca. 80% des Ablaufs mit nur 20% der angebotenen Befehle erledigt. Diese überraschenden Erkenntnisse führten dazu, daß auf eine andere Prozessorarchitektur mit einem stark reduzierten Befehlssatz zurückgegriffen wurde.

Dieses neue Konzept war schon Anfang der 80er Jahre auf Basis von 30 relativ einfachen Befehlen mit der **RISC-Architektur** [ursprünglich: **r**educed **i**nstructions **s**et **c**omputer; heute: **r**eusable (mehrfach ladbar) **i**nstructions **s**et **c**omputer] entwickelt worden. Im Gegensatz zum CISC-Prozessor mit seinen durchschnittlich 16 internen Datenregistereinheiten besitzt ein RISC-Prozessor bis zu 100 interne Datenregister. Durch diese Struktur wird die Zentraleinheit erheblich weniger mit zeitintensiven Lade- bzw. Speicherbefehlen belastet, weil viele aktuelle Daten im Prozessor „aufbewahrt" werden können. Aufgrund des eingeschränkten Befehlssatzes muß ein RISC-Rechner seine einfachen Befehle zur Erledigung einer Aufgabe eventuell häufiger abarbeiten als ein CISC-Rechner. Durch die hohe Registerzahl läuft das Programm insgesamt aber bedeutend schneller ab. Nach heutiger Einschätzung wird die RISC-Architektur aus diesem Grund die CISC-Rechner vom Markt verdrängen.

Bei diesen unterschiedlichen internen Organisationen ist es heute nicht mehr möglich, einen allgemein gültigen inneren Aufbau eines Mikroprozessors darzustellen. In diesem Abschnitt wird deshalb von einem gedachten 8-Bit-Mikroprozessor ausgegangen,

der die wichtigsten Leistungsmerkmale der 8-Bit-Typen beinhaltet. Dort kann man noch von einer grundsätzlich einheitlichen Architektur ausgehen. Soweit erforderlich, wird aber auf besonders wichtige oder interessante Unterschiede der 16- und 32-Bit-Familien eingegangen. Die dabei verwendeten Bezeichnungen und Abkürzungen sind, soweit sie nicht bei allen Mikroprozessoren einheitlich sind, nach Möglichkeit in Übereinstimmung mit weit verbreiteten Typen wie Intel 8080/85 und ZILOG Z80 gewählt.

Das Blockschaltbild in Abb. 10.6 enthält nur die Verbindungen, über die Daten und Adressen transportiert werden. Die Steuerleitungen zwischen jeder dargestellten Einheit und der Ablaufsteuerung wurden weitgehend weggelassen, weil sonst die Darstellung unübersichtlich wird. Die Aufgabe der einzelnen Blöcke in Abb. 10.6 ergibt sich großenteils aus dem Grundprinzip, nach dem die 8-Bit-Computer die Programme abarbeiten (Von-Neumann-Zyklus):

a) Befehl aus dem Speicher holen
b) Befehl interpretieren
c) falls erforderlich, Daten (Operanden) aus dem Speicher holen
d) Befehl ausführen
e) nächste Befehlsadresse ermitteln

10.4.1 Akkumulator

Der Akkumulator ist ein Datenregister. Er kann vom Speicher, der Ein-/Ausgabe und von anderen CPU-Registern geladen werden und seinen Inhalt auch dorthin abspeichern. Für die meisten Operationen des Rechenwerks (der ALU) enthält der Akkumulator vor der Verarbeitung einen Operanden und nach der Verarbeitung das Ergebnis. Es gibt Mikroprozessoren mit mehreren Akkumulatoren (z. B. Motorola 6800) und solche, die keinen Akkumulator haben, dafür aber mehrere allgemeine Register, die auch als Akkumulator verwendet werden können. Bei ihnen muß im Befehl angegeben sein, welcher Akkumulator bzw. welches Register angesprochen werden soll.

10.4.2 ALU

Die ALU ist das Rechenwerk im Mikroprozessor. Sie hat zwei 8-Bit-Eingänge (oben und unten im Blockschaltbild) für die beiden Operanden und nach links einen 8-Bit-Ausgang für das Ergebnis, das über den internen Datenbus wieder in den Akkumulator gelangt. In den meisten Fällen enthält die ALU ein Schaltnetz, mit dem die beiden Operanden negiert, addiert, subtrahiert und verglichen sowie über UND-,

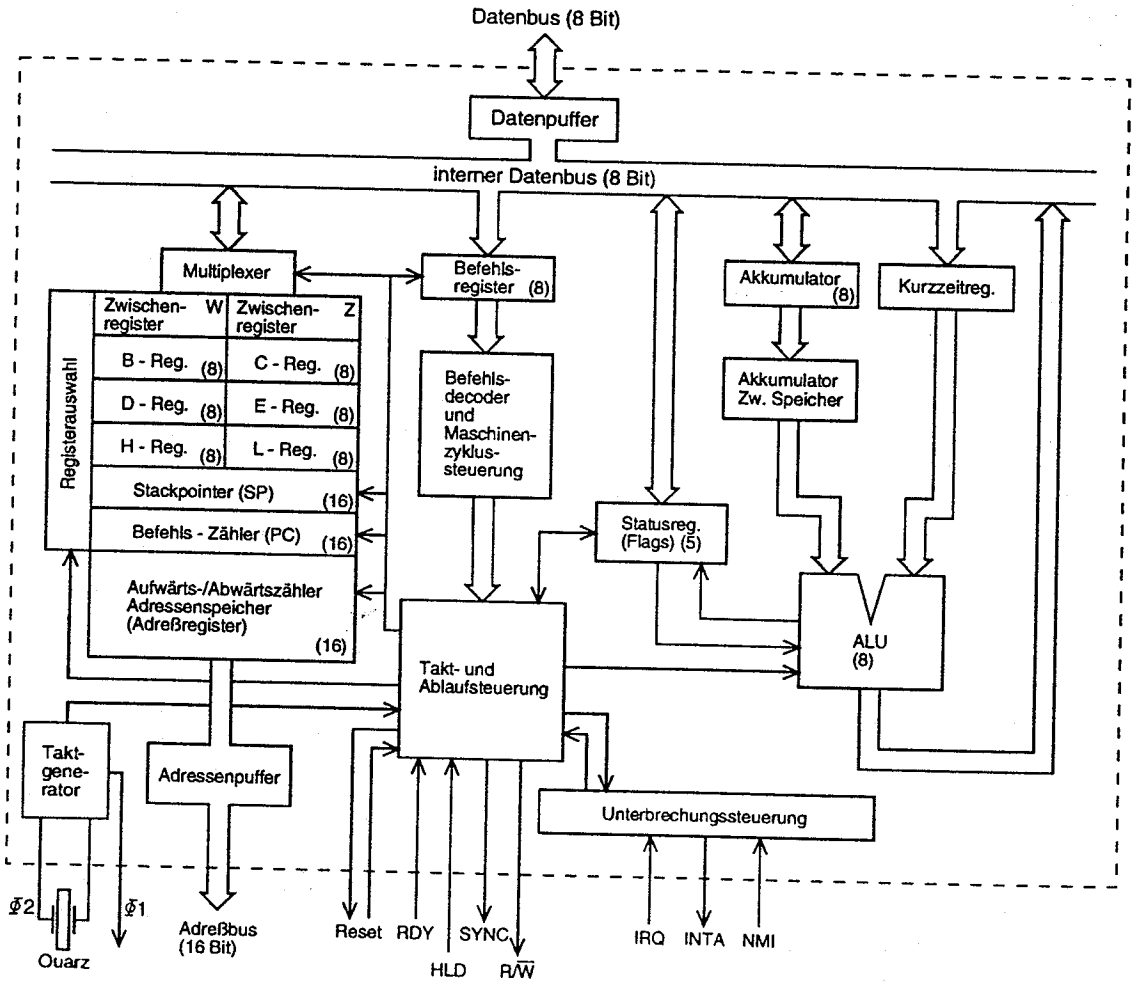


Abb. 10.6 – Blockschaltbild eines Mikroprozessors

ODER- und Exklusiv-ODER (XOR)-Glieder verknüpft werden können.

Mit der beschriebenen Ausstattung der ALU, wie sie bei 8-Bit-Mikroprozessoren üblich ist, müssen „kompliziertere" Operationen wie Multiplikation und Division vom Programmierer umständlich auf Addition und Subtraktion zurückgeführt werden. Die größeren 16-Bit-Prozessoren können durch eine umfangreichere ALU und Ablaufsteuerung auch mit einem Befehl multiplizieren und dividieren.

10.4.3 Statusregister

An die ALU angeschlossen ist auch das Statusregister (Zustandsregister). Seine Bit kennzeichnen das Ergebnis der Operation in der ALU und werden daher auch als Flags (Fahnen) bezeichnet. Ist das Ergebnis Null, so wird das Zero-Flag (Z) auf 1 gesetzt. Entsprechend kennzeichnet C = 1 (Carry-Flag), daß ein Übertrag entstanden ist und N = 1 (Negative-Flag), daß das Ergebnis negativ ist. Während die genannten drei Ergebniseigenschaften (Null, Übertrag und Vorzeichen) bei allen Mikroprozessoren im Statusregister angezeigt werden, ist die Belegung der übrigen Bit des Statusregisters unterschiedlich.

Der Inhalt der Zustandsbit entscheidet über den weiteren Programmablauf bei Programmverzweigungen. Programmverzweigungen werden über bedingte Sprungbefehle verwirklicht. Beim Befehl „Springe, wenn Ergebnis = 0" fragt die Ablaufsteuerung den Zustand des Zero-Flags ab. Bei Z = 1 wird das Programm zu der im Sprungbefehl enthaltenen Adresse verzweigt, bei Z = 0 an der alten Adresse fortgesetzt. Entsprechend bedingte Sprungbefehle gibt es auch für die übrigen Flags. Wegen dieser Bedeutung der Zustandsbit muß der Programmierer genau wissen, welche Befehle Auswirkungen auf den Inhalt bestimmter Zustandsbit haben.

10.4.4 Befehlszähler (program counter PC)

Die Befehlsfolge, die das abzuarbeitende Programm darstellt, liegt im Speicher. Damit der Mikroprozessor einen Befehl nach dem anderen abarbeiten kann, muß er jeweils die Speicheradresse des nächsten Befehls kennen. Damit dies einfach möglich ist, werden die Befehle in aufeinanderfolgende Speicherzellen abgelegt. Die Adresse des nächsten Befehls kann daher einfach durch Mitzählen der abgearbeiteten Befehle ermittelt werden. Diese Aufgabe übernimmt der Befehlszähler. Er wird zu Beginn der Verarbeitung auf die Anfangsadresse des Programms gesetzt und dann bei jeder Befehlsausführung weitergezählt.

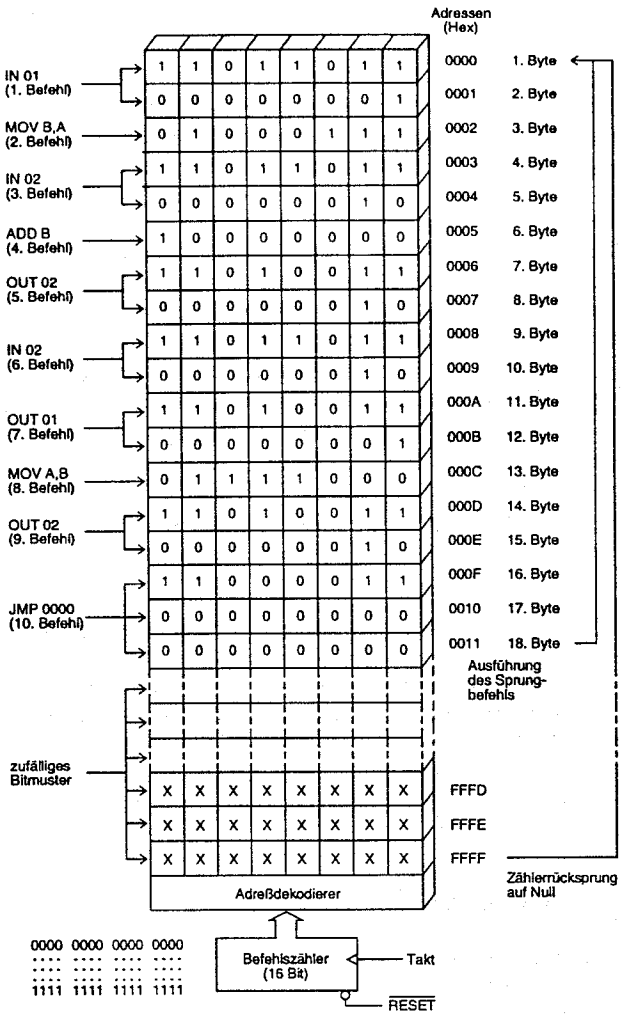


Abb. 10.7 – Befehle im Speicher

Dadurch, daß der Befehlszähler nach dem höchsten Wert FFFFH wieder mit 0000H beginnt, wird die Ablaufsteuerung zu diesem Zeitpunkt auch wieder mit dem ersten Befehl weiterarbeiten. Natürlich kann dieses Zurückspringen nur erfolgen, wenn kein Halt-Befehl im Programm enthalten ist. Hat man diesen Befehl vergessen, oder wurde er bewußt nicht verwendet, wird der Zähler immer „im Kreis" arbeiten. Man sagt auch, das Programm läuft in einer unendlichen Schleife. Normalerweise wird es dabei auf zufällige Bitmuster im nicht programmierten Teil des Speichers stoßen. Diese werden dann von der Zentraleinheit interpretiert und ausgeführt. Im ungünstigen Fall wird dabei das Programm zerstört, der Rechner „stürzt ab".

Die Mikroprozessoren haben einen Befehlssatz, der unterschiedlich lange Befehle enthält. Bei den 8-Bit-Mikroprozessoren beträgt die Befehlslänge 1 bis 3 Byte (beim Prozessor Z80 sogar bis zu 4 Byte). Da bei diesen Prozessoren die Befehle (und auch die Daten) byteweise im Speicher abgelegt sind, ist der Befehlszähler eigentlich ein Bytezähler. Das erste Byte eines Befehls beinhaltet immer den Opera-

tionscode. Aus dem Operationscode erkennt die Befehlsdekodierung und Ablaufsteuerung unter anderem, ob noch weitere Bytes zu dem Befehl gehören und liest sie gegebenenfalls aus dem Speicher. Bei jedem Lesen eines Befehlsbytes wird der Inhalt des Befehlszählers um 1 erhöht.

Bei den 8-Bit-Mikroprozessoren hat der Befehlszähler meist eine Länge von 16 Bit = 2 Byte (entsprechend dem 16-Bit-Adreßbus). Durch einen Sprungbefehl kann der Mikroprozessor dazu veranlaßt werden, die gerade bearbeitete Befehlsfolge zu verlassen und zu einem an anderer Stelle im Speicher liegenden Programmteil überzugehen. 3 Byte lange Sprungbefehle enthalten im zweiten und dritten Byte die Beginnadresse des neuen Programmteils. Bei Sprungbefehlen wird daher der Befehlszähler nicht weitergezählt, sondern mit dem zweiten und dritten Befehlsbyte neu geladen.

10.4.5 Befehlsregister (instruction register IR)

Das Befehlsregister nimmt jeweils das erste Byte eines Befehls auf, also den Operationscode, und speichert ihn so lange, bis der Befehl abgearbeitet ist. Das Befehlsregister gibt den Operationscode unverändert an die Befehlsdekodierung und Ablaufsteuerung weiter.

10.4.6 Daten-Adreß-Register

Ein Daten-Adreß-Register (hier das HL-Register) kann für zwei verschiedene Zwecke verwendet werden. Zum einen ist es ein allgemeines Datenregister, in dem Daten zwischengespeichert werden können. Das Zwischenspeichern von Daten in einem CPU-Register geht schneller als das vorübergehende Ablegen der Daten im Speicher. In den meisten Fällen besteht auch die Möglichkeit, durch Befehle den Inhalt der Datenregister um 1 zu erhöhen (zu inkrementieren) oder um 1 zu verringern (zu dekrementieren).

Als Adreßregister wird das Daten-Adreß-Register bei Befehlen mit der in Mikroprozessoren weit verbreiteten impliziten Adressierung verwendet. Implizite Adressierung heißt, die Adresse ist bereits im Operationscode mit enthalten. Befehle dieser Adressierungsart, die auch als registerindirekte Adressierung bezeichnet wird, haben daher nur eine Länge von einem Byte. Der Operationscode enthält neben der auszuführenden Tätigkeit noch die Information, daß der Operand unter der im Daten-Adreß-Register liegenden Adresse im Speicher zu finden ist. Der 1-Byte-Befehl MOV A, M z. B. bewirkt, daß der Inhalt des Daten-Adreß-Registers HL auf den Datenbus gegeben und der Inhalt der so adressierten Speicherzelle in den Akkumulator geladen wird. Die implizite Adressierung mit Hilfe des Daten-Adreß-Registers benötigt nur 1-Byte-Befehle, dafür sind aber zusätzliche Befehle erforderlich, um das Daten-Adreß-Register erst einmal mit der gewünschten Adresse zu laden.

10.4.7 Indexregister

Ein Indexregister ist ein Hilfsregister. Zum einen kann es wie das Daten-Adreß-Register als allgemeines Register verwendet werden. Zum anderen wird es benötigt für die indizierte Adressierung. Bei der indizierten Adressierung wird zum Inhalt der im 2. und 3. Byte des Befehls angegebenen Adresse erst der Inhalt des Indexregisters addiert und dann mit dieser Summe der Speicher adressiert. Diese Adressierungsart bietet große Vorteile, wenn eine ganze Liste von Daten verarbeitet werden soll. Im Befehl wird dann die Anfangsadresse der Liste angegeben. Das Indexregister wird mit der Anzahl der Daten in der Liste geladen. Mit dem ersten Befehl mit indizierter Adressierung wird somit der letzte Platz in der Liste angesprochen. Nach jeder Verarbeitung wird der Inhalt des Indexregisters um 1 vermindert, so daß nacheinander vom letzten bis zum ersten Listenplatz alle Daten verarbeitet werden. Sobald der Inhalt des Indexregisters Null wird, wird das Zero-Flag auf 1 gesetzt. Daran erkennt der Mikroprozessor, daß die Liste vollständig abgearbeitet ist.

10.4.8 Stapelzeiger (stack pointer SP)

Ein Stapelspeicher (stack), häufig auch Kellerspeicher genannt, ist ein Speicher, bei dem das zuletzt Eingeschriebene als erstes wieder ausgelesen wird, so wie man normalerweise von einem Stapel das als erstes wieder herunternimmt, was man zuletzt obendrauf gelegt hat. Aus dieser Eigenschaft entstand auch die Bezeichnung LIFO (last in first out). Es gibt CPU, in denen der Stapelspeicher aus einer Reihe Register gebildet wird, die so zusammengeschaltet sind, daß sich die beschriebene Eigenschaft ergibt. Die meisten Mikroprozessoren enthalten aber nur den Stapelzeiger (Rückwärtszähler) als Steuerung für den Kellerspeicher, der selbst an beliebiger Stelle

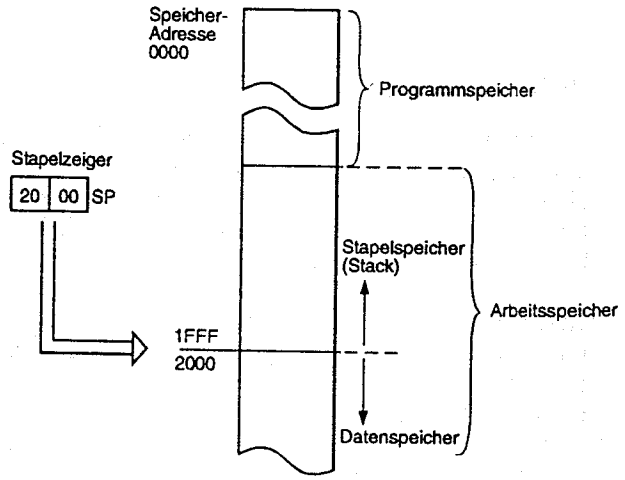
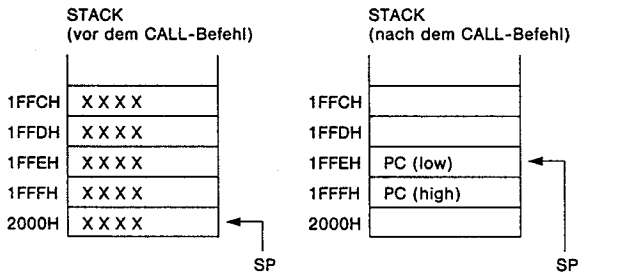


Abb. 10.8 – Stack

im Speicher liegt. Dafür muß der Stapelzeiger eine Länge von 2 Byte haben. Im Ruhezustand (leerer Stapelspeicher) ist der Stapelzeiger mit der Anfangsadresse des Kellerspeichers geladen. Nach jedem Schreiben in den Kellerspeicher wird der Stapelzeigerinhalt um 1 erhöht und zeigt damit auf die nächste freie Zelle des Stapelspeichers. Vor jedem Lesen wird der Stapelzeigerinhalt um 1 vermindert und adressiert daher die Speicherzelle, in die als letztes geschrieben wurde. So wird durch den Stapelzeiger ein nahezu beliebig großer Kellerspeicher verwirklicht, wobei der Mikroprozessor selbst nur ein 16-Bit-Register enthält. Bei vielen Mikroprozessoren wächst der Kellerspeicher nach unten (mit zunehmender Füllung des Stapelspeichers zeigt der Stapelzeiger auf immer niedrigere Adressen), weil beim Schreiben der Stapelzeigerinhalt vermindert und beim Lesen wieder erhöht wird.

Der Stapelspeicher wird z. B. dazu verwendet, um bei Unterprogrammansprüngen die Rücksprungadresse zu speichern. Die CALL-Befehle (springe zum Unterprogramm) speichern den Inhalt des Befehlszählers (die Adresse des nächsten Befehls im Hauptprogramm) im Kellerspeicher ab und laden danach den Befehlszähler mit der im CALL-Befehl festgelegten Adresse.

Beispiel:



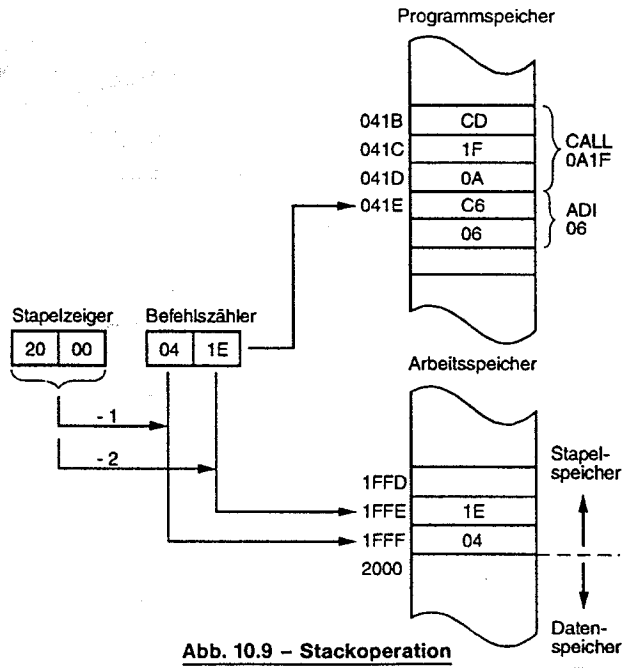


Abb. 10.9 – Stackoperation

Der Unterprogrammaufruf durch den CALL-Befehl wird immer (unbedingt) ausgeführt. Es werden keine Flags abgefragt. Erfolgt ein Unterprogrammaufruf über den CALL-Befehl, so muß die Rücksprungadresse vorher im Stack abgespeichert werden, erst dann kann der Sprung ins Unterprogramm erfolgen.

Die Abb. 10.9 stellt den Ablauf für den Befehl CALL 0A1F dar.

Das Unterprogramm muß immer mit dem Befehl RET (return from subroutine = kehre vom Unterprogramm zurück) enden. Durch den RET-Befehl wird der Befehlszähler mit dem zuletzt eingeschriebenen Stapelspeicherinhalt überschrieben, also mit dem Befehlszählerstand vor dem Einsprung ins Unterprogramm. Dadurch wird das Hauptprogramm wieder an der richtigen Stelle fortgesetzt.

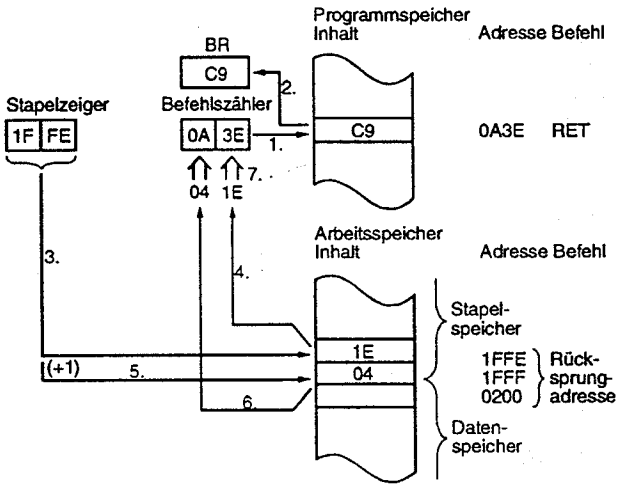


Abb. 10.10 – Returnoperation

Durch den Stapelspeicher wird es möglich, mehrere Unterprogramme ineinander zu verschachteln und trotzdem immer wieder die richtige Rücksprungadresse zu finden. Meist wird bei Unterprogrammsprüngen nicht nur der Befehlszählerstand, sondern auch der Inhalt der übrigen Register im Kellerspeicher gerettet. Das muß aber vom Programmierer durch besondere Befehle realisiert werden.

10.4.9 Taktgenerator, Befehlsdekodierung, Ablaufsteuerung

Alle Vorgänge im Inneren des Mikroprozessors werden von der Ablaufsteuerung bestimmt. Für ihre Arbeit muß der Ablaufsteuerung eine Taktfrequenz zugeführt werden. Es gibt fast nur noch Mikroprozessoren, bei denen sich der Taktgenerator mit auf demselben Chip befindet, und wenige andere, bei denen für den Taktgenerator ein weiteres IC erforderlich ist. In jedem Fall wird aber die Taktfrequenz durch externe Bauelemente eingestellt, und zwar je nach gewünschter Genauigkeit durch RC-Glieder oder

Quarze. Viele Mikroprozessoren verwenden einen Zwei-Phasen-Takt, wie er in Abb. 10.11 dargestellt ist. Er hat den Vorteil, daß innerhalb einer Taktperiode *T* vier Taktflanken auftreten und damit mehrere Vorgänge angestoßen werden können.

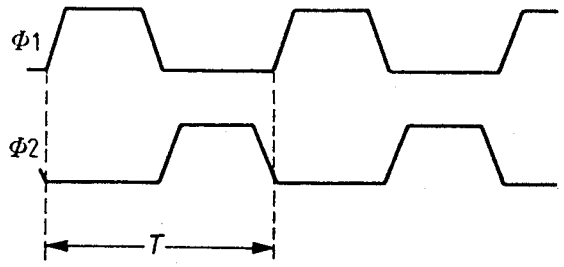


Abb. 10.11 – Zwei-Phasen-Takt

Die Ablaufsteuerung ist im Prinzip aufgebaut wie ein Mikrocontroller (s. Abb. 10.4). Ein Mikroprogramm, das in einem ROM abgespeichert ist, wird im Rhythmus der Taktfrequenz und in Abhängigkeit vom Zustand an den Bedingungseingängen durchlaufen. Die Bedingungseingänge sind mit den Ausgängen des Befehlsdekodierers, mit dem Statusregister und mit den externen Steuereingängen beschaltet. Die Steuerausgänge (in Abb. 10.9 nur teilweise abgebildet) führen zu den einzelnen Teilen des Mikroprozessors und aktivieren sie zum richtigen Zeitpunkt entsprechend dem Operationscode des gerade zu bearbeitenden Befehls. Die Funktion der externen Steuerein- und -ausgänge wird im Abschnitt 10.5 behandelt.

10.4.10 Interne Busse und Bustreiber

Der Mikroprozessor arbeitet auch intern mit Bussystemen, meist mit einem internen Datenbus und einem internen Adreßbus (s. Abb. 10.6). Die Ablaufsteuerung sorgt dafür, daß immer nur ein Funktionsteil auf einem Bus sendet und nur ein anderes die Daten aufnimmt. Zwischen internem und äußerem Adreßbus liegt ein Bustreiber, auch Buspuffer genannt. Er hat die Aufgabe, die Signale des internen Adreßbusses so weit zu verstärken, daß sie die extern angeschlossenen RAM-, ROM- und I/O-Bausteine steuern können. Die Ablaufsteuerung sorgt dafür, daß der Adreßtreiber die Adressen nur zu den gewünschten Zeitpunkten auf den Adreßbus schaltet. Interner und externer Datenbus sind durch den Datenbuspuffer voneinander getrennt. Der Datenbuspuffer ist ein bidirektionaler Busleitungstreiber. Die Ablaufsteuerung muß für ihn nicht nur den Zeitpunkt, sondern auch die Richtung der Durchschaltung (Eingabe oder Ausgabe) festlegen.

Ein Mikroprozessor-IC benötigt sehr viele Anschlüsse. So sind für einen 8-Bit-Mikroprozessor mit 16-Bit-Adreßbus (64-KByte-Speicher) erforderlich: 8 + 16 = 24 Anschlüsse für Daten- und Adreßbus, 2 bis 3 Anschlüsse für den Takt und 2 bis 4 Anschlüsse für die Stromversorgung. Bei einem 40poligen Gehäuse bleiben dann noch gerade 10 Anschlüsse für zusätzliche Steuerein- und -ausgänge. Reicht dies nicht aus, so wird ein Teil der Anschlüsse durch Multiplexbetrieb doppelt ausgenutzt. Beim 8080 wird z. B. während der ersten Taktperiode eines jeden Befehlszyklus ein Statuswort an den Datenbus gelegt. Aus diesem Statuswort werden extern die Signale für den Steuerbus erzeugt. Beim 8085 liegt der Datenbus auf den gleichen Anschlüssen wie das niederwertige Adreßbyte. Ein externer Demultiplexer trennt die zeitlich verschachtelten Signale auf den Adreß- und den Datenbus. Von den 16-Bit-Mikroprozessoren kommen einige, wie die Intel 8086 und 8088, durch verstärkte Doppelausnutzung der Anschlüsse auch mit 40poligen Gehäusen aus. Andere, wie der Motorola 68000, der zur Vereinfachung der äußeren Beschaltung auf die Mehrfachausnutzung der Anschlüsse verzichtet, benötigen ein 64poliges Gehäuse, der 68040 sogar ein 180poliges.

10.5 Arbeitsweise und Betriebsarten eines Mikroprozessors

10.5.1 Steuerung der Befehlsabarbeitung

Wie bereits im Abschnitt 10.4 dargestellt, sind für die Abarbeitung eines Befehls – für einen Befehlszyklus – folgende grundsätzliche Schritte erforderlich: Befehl aus dem Speicher holen (Befehlsabruf), Befehl interpretieren, eventuell Daten aus dem Speicher holen, Befehl ausführen und nächste Befehlsadresse ermitteln. Die Befehle wurden vor der Abarbeitung in unmittelbar aufeinanderfolgende Speicherzeilen geschrieben. Dabei muß die Reihenfolge, in der sie vom System abgearbeitet werden sollen, unbedingt eingehalten werden.

Eine 8-Bit-CU erhält nach dem Einschalten einen Auto-RESET und beginnt daher an der Adresse 0 mit der Bearbeitung der Befehle. Das bedeutet, daß in der Speicherzeile 0 immer ein ausführbarer Befehl stehen muß! Nach dem automatischen RESET schaltet die CU den Befehlszähler an den Adreßbus. Dann beginnt der sogenannte „Von-Neumann-Zyklus".

Jeder 8-Bit-Prozessor arbeitet einen Befehl in 3 Phasen (Zyklen) ab.

1. Die Holphase:
Der Befehl wird aus der durch den Befehlszähler adressierten Speicherzeile (hier 0000) des Speichers **geholt** (Leseoperation) und im Prozessor im Befehlsregister (Instruction Register IR) abgespeichert.

2. Die Dekodierphase:
Der Befehl wird von der Steuereinheit dekodiert. Er wird dafür mit den im ROM befindlichen Befehlen verglichen.

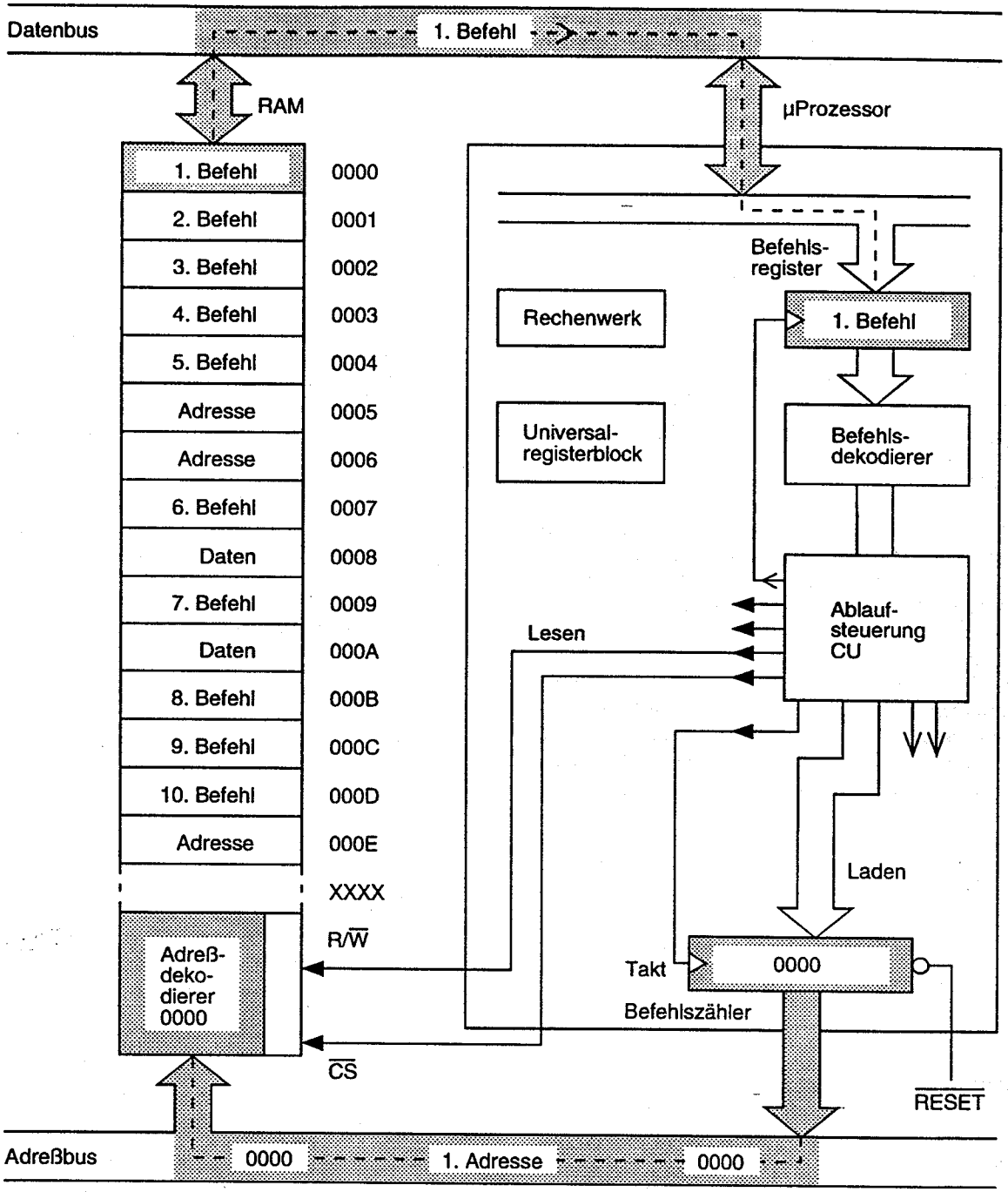


Abb. 10.12 – Von-Neumann-Zyklus

3. Die Ausführungsphase
Der Befehl wird von der Ablaufsteuerung ausgeführt. Dafür erzeugt sie die für den Ablauf der befohlenen Operation erforderlichen Steuersignale (nach ROM-Anweisungen). Während der Abarbeitung wird der Befehlszähler um einen Schritt weitergestellt. Steht ein Befehl auf mehreren Speicherzeilen, werden die zusammengehörenden Datenwörter (beim 8-Bit-System die Bytes) nacheinander in den Prozessor geholt. Dabei wird der Befehlszähler jeweils weitergeschaltet (inkrementiert). **Zum Ende der Befehlsabarbeitung zeigt er immer auf die folgende Adresse!**

Der Operationscode (Opcode) bestimmt die auszuführende Operation. Nachdem die CU den Opcode dekodiert hat, entscheidet sie, ob noch ein weiteres Wort aus dem Speicher geholt werden muß.

Durch das Weiterzählen des Befehlszählers ist eine automatische Befehlsverarbeitung möglich. Die Befehle werden dabei sequentiell (nacheinander) abgearbeitet. Von dieser Arbeitsweise kann durch Sprungbefehle abgewichen werden. Ein Sprungbefehl verursacht ein Umladen des Befehlszählers. Hierbei wird eine neue Adresse in den Zähler einge-

geben, von der aus er dann sequentiell weiterarbeitet. Die sequentielle Arbeitsweise ist auch die „große" Schwäche der herkömmlichen µP. Sie erfordert natürlich einen größeren Zeitaufwand als eine Parallelverarbeitung von Daten. Die Hersteller versuchen, diesen Nachteil durch höhere Verarbeitungsgeschwindigkeit oder Veränderung der Prozessorarchitektur auszugleichen. Der Nachteil läßt sich aber nicht voll beseitigen. Für Hochgeschwindigkeits-Computer werden deshalb immer häufiger Parallel-Systeme verwendet, die nicht nach dem Von-Neumann-Zyklus arbeiten.

Ist ein herkömmlicher Computer gestartet worden, wiederholt sich der Abarbeitungszyklus immer wieder. Er läßt sich nur durch einen Einzelschritt- bzw. Halt-Befehl oder das Abschalten der Versorgungsspannung stoppen.

Bei den 8-Bit-Mikroprozessoren sieht dieser Ablauf im einzelnen etwas anders aus, weil sie die Befehle nicht auf einmal, sondern byteweise aus dem Speicher holen und die Befehle unterschiedlich lang sind. Die Steuerung dieser Abläufe ist bei einzelnen Mikroprozessoren wieder z.T. sehr unterschiedlich. Da es hier nur darum geht, die grundsätzliche Arbeitsweise eines Mikroprozessors zu verstehen, wird im folgenden eine relativ einfache zeitliche Steuerung beschrieben, so wie sie in ähnlicher Form z.B. in den Mikroprozessoren Motorola 6800 und Mostek 6502 eingesetzt ist.

Ein Befehlszyklus besteht aus mehreren Maschinenzyklen. Die Zahl der Maschinenzyklen je Befehlszyklus hängt vom Befehl ab und liegt zwischen 2 und 7. Ein Maschinenzyklus hat die Dauer einer Taktperiode *T* (s. Abb. 10.11). Bei der Befehlsabarbeitung treten drei verschiedene Grundtypen von Maschinenzyklen auf:

a) Maschinenzyklus Lesen

Der Maschinenzyklus Lesen muß immer dann ablaufen, wenn ein Byte vom RAM, vom ROM oder von der I/O über den Datenbus in den Mikroprozessor eingelesen werden soll. Abb. 10.13 zeigt für die Operation neben den Takten $\Phi 1$ und $\Phi 2$ den Steuerausgang R/$\overline{W}$ (Lesen/Schreiben) sowie den Adreßbus A0 ... A15 und den Datenbus D0 ... D7. Der Lesevorgang ist durch R/W = H gekennzeichnet. Wenn $\Phi 1$ im Zustand H ist, wird R/$\overline{W}$ nach H gesteuert und der Adreßbus, der vorher im hochohmigen Zustand Z ist, aktiviert. Dabei wird die Adresse vom Mikroprozessor auf den Adreßbus ausgegeben. Da dabei von den 16 Adreßleitungen einige mit dem Pegel H und andere mit L angesteuert werden, sind im Diagramm beide Pegel parallel eingezeichnet. Da es einige Zeit dauert, bis die Leitungskapazitäten des Adreßbusses geladen sind und damit die Adresse auf dem Bus stabil ist, darf der Datenbus erst später freigegeben werden. In der hier angenommenen Steuerung wird der Datenbus aktiviert, wenn $\Phi 2$ den Zustand H angenommen hat. In dieser Zeit sendet der Speicher oder die I/O das Datenbyte, das dann von der CPU aufgenommen wird.

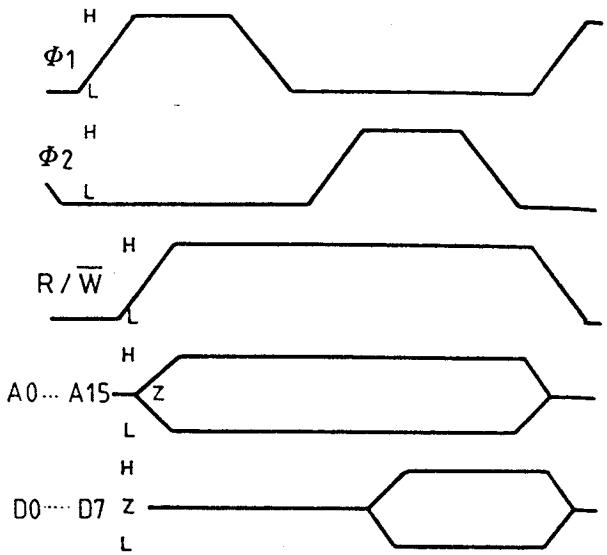


Abb. 10.13 – Maschinenzyklus Lesen

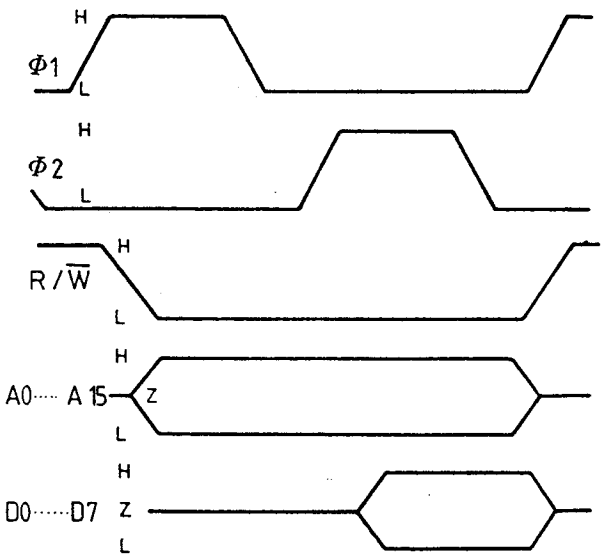


Abb. 10.14 – Maschinenzyklus Schreiben

b) Maschinenzyklus Schreiben

Soll ein Byte vom Mikroprozessor ausgegeben und in den RAM oder die I/O eingeschrieben werden, so muß dafür der Maschinenzyklus Schreiben gestartet werden. Abb. 10.14 unterscheidet sich von Abb. 10.13 nur dadurch, daß der Steuerausgang R/$\overline{W}$ den das Schreiben kennzeichnenden Zustand L annimmt. Weitere Unterschiede zwischen den beiden Maschinenzyklen bestehen z.T. bei weiteren Steuerausgängen und natürlich beim inneren Ablauf im Mikroprozessor. Wie die Abbildungen 10.13 und 10.14 zeigen, wird mit $\Phi 1$ der Adreßbus und mit $\Phi 2$ der Datenbus aktiviert. Bei Mikroprozessoren mit dieser Form der zeitlichen Steuerung der Befehlsausführungen wird

daher $\Phi 1$ als Adreßtakt und $\Phi 2$ als Datentakt bezeichnet.

c) Maschinenzyklus für interne Operationen
Für das Abarbeiten der Befehle sind neben Lese- und Schreiboperationen teilweise auch im Mikroprozessor intern ablaufende Vorgänge zu steuern, für die ein eigener Maschinenzyklus vorgesehen ist. Während dieser Zyklen passiert nichts auf den Bussystemen.

Der zeitliche Ablauf der Befehlsausführung besteht für jeden Befehl immer aus einer unterschiedlichen Folge der drei Maschinenzyklen, wie im folgenden an zwei Beispielen gezeigt werden soll:

ADC-6A83: Dieser Befehl soll bewirken, daß der Inhalt der Speicherzelle 6A83 unter Berücksichtigung eines eventuell vorhandenen Übertrags zum Inhalt des Akkumulators addiert und das Ergebnis wieder in den Akkumulator gebracht wird. Im Speicher sieht der Befehl z.B. folgendermaßen aus: 6D 83 6A. 6D sei der Operationscode des Befehls. Die beiden folgenden Bytes kennzeichnen die Adresse. Bei den meisten Mikroprozessoren werden 2 Byte lange Adressen so im Speicher abgelegt, daß das niederwertige Adreßbyte ADL in der Speicherzelle mit der niedrigeren Adresse liegt und das höherwertige Adreßbyte ADH dann in der darauffolgenden. Für die Ausführung dieses Befehls sind vier Maschinenzyklen Lesen erforderlich (siehe Abb. 10.15).

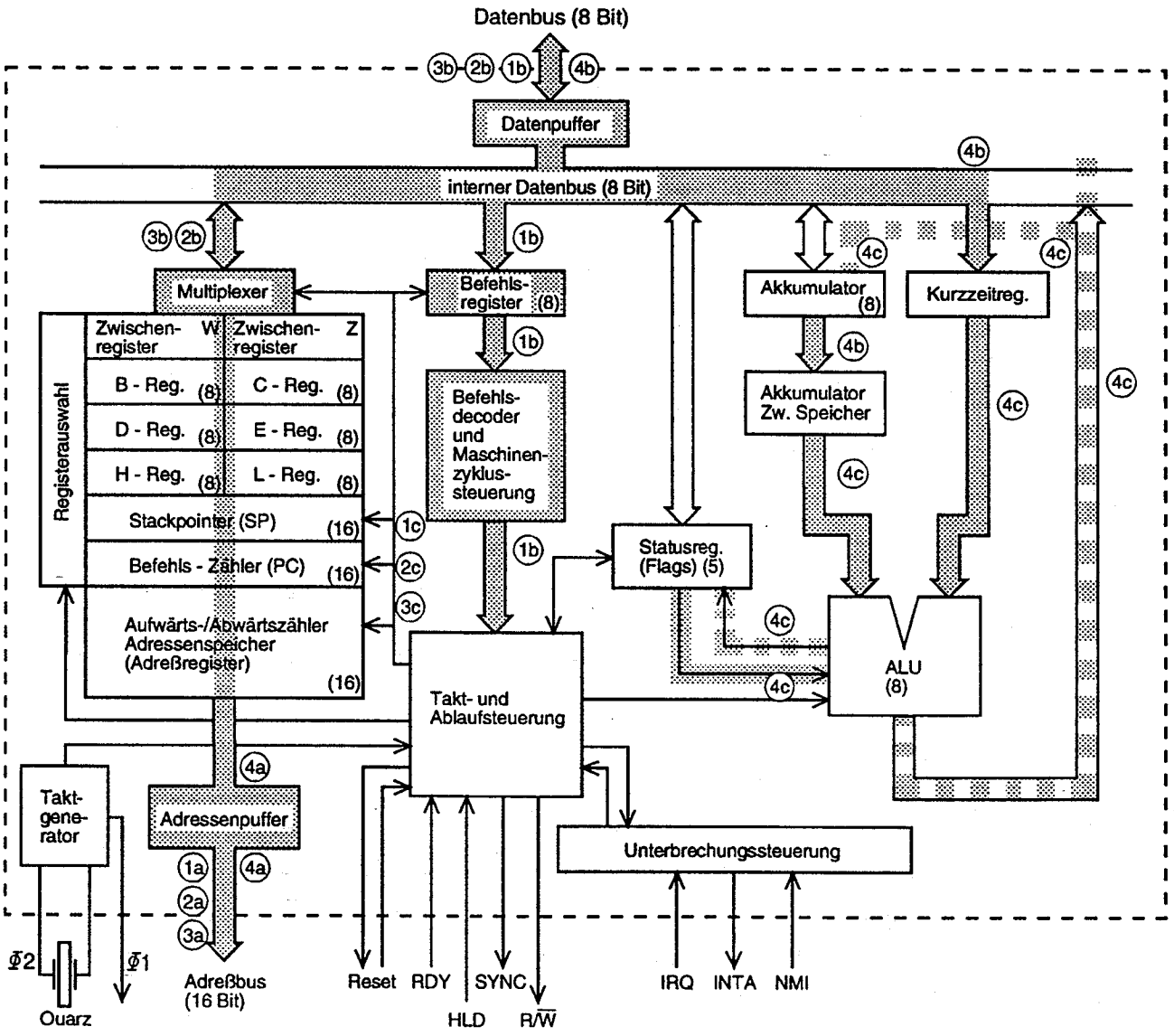


Abb. 10.15 – Ausführung des Befehls ADC-6A83

1. Maschinenzyklus Lesen: Operationscode holen. Der Inhalt des Befehlszählers wird auf den Adreßbus gegeben (1a). Über den Datenbus gelangt das erste Befehlsbyte in das Befehlsregister und von dort aus weiter in die Befehlsdekodierung und Ablaufsteuerung (1b). Am Ende des Zyklus wird der Befehlszählerinhalt um 1 erhöht (1c).
Die Ablaufsteuerung erkennt am Operationscode, welche weiteren Maschinenzyklen erforderlich sind, um den Befehl abzuarbeiten.

2. Maschinenzyklus Lesen: ADL holen.
Der Inhalt des Befehlszählers wird auf den Adreßbus gegeben (2a), und über den Datenbus kommt das niederwertige Adreßbyte, das im Adreßregister zwischengespeichert wird, in die CPU (2b). Mit 2c wird wieder der Befehlszähler um 1 erhöht.

3. Maschinenzyklus Lesen: ADH holen.
Es laufen die gleichen Vorgänge wie unter 2. ab, sie sind mit 3a bis 3c in Abb. 10.15 gekennzeichnet.

4. Maschinenzyklus Lesen: Operand holen und Befehl ausführen.

Um den Operand zu holen, wird jetzt der Inhalt des Adreßregisters auf den Adreßbus gegeben (4a). Das auf dem Datenbus ankommende Byte gelangt in das Zwischenregister der ALU (4b). Gleichzeitig liegt an der ALU der momentane Akkumulatorinhalt und vom Statusregister her der Inhalt des Übertragsbit (Carry-Flag). Im Schritt 4c werden alle drei Werte addiert, das Ergebnis wird über den internen Datenbus in den Akkumulator geschrieben, und die Zustandsbits im Statusregister werden dem Ergebnis entsprechend gesetzt.

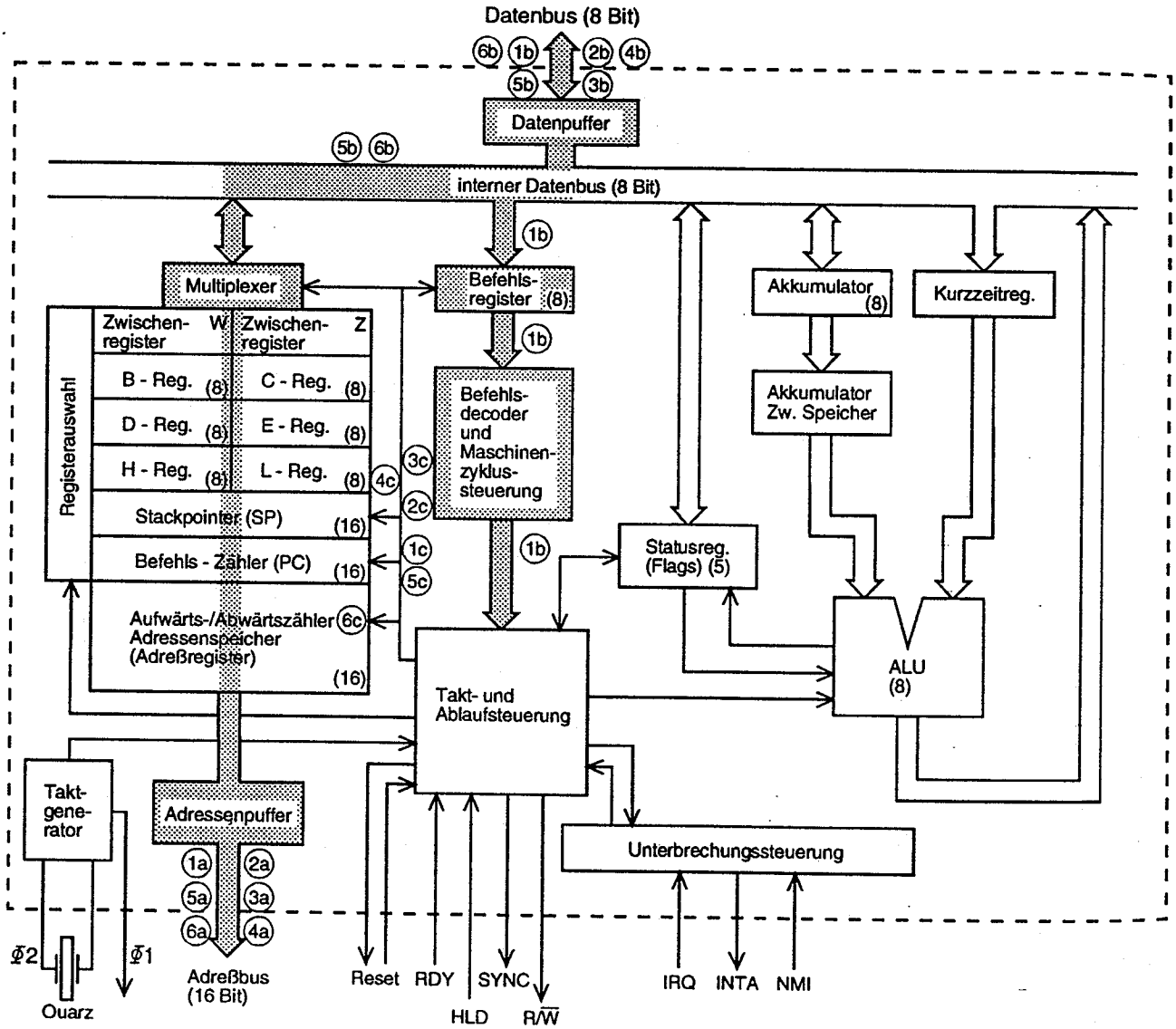


Abb. 10.16 – Ausführung des Befehls CALL-1234

CALL-1234: Mit diesem Befehl wird ein an Adresse 1234 beginnendes Unterprogramm angesprungen. Bei der Befehlsausführung wird automatisch der momentane Inhalt des Befehlszählers (die Rücksprungadresse) und der Inhalt des Statusregisters auf den Stapelspeicher gerettet und anschließend die im Befehl angegebene Beginnadresse des Unterprogramms in den Befehlszähler übernommen. Im Speicher liegt der Befehl in der Form OP-Code, ADL, ADH: 20 34 12. Die Ausführung dieses Befehls benötigt sechs Maschinenzyklen, und zwar drei Lesezyklen und drei Schreibzyklen (Abb. 10.16).

1. Maschinenzyklus Lesen: Operationscode holen (1a, 1b, 1c).

2. Maschinenzyklus Schreiben: PCH auf den Stapelspeicher retten.
Mit dem Stapelzeiger wird der Speicher adressiert (2a). PCH wird über den Datenbus in die adressierte Speicherzelle geschrieben (2b). Der Inhalt des Stapelzeigers wird um 1 vermindert (2c).

3. Maschinenzyklus Schreiben: PCL auf den Stapelspeicher retten (3a, 3b, 3c).

4. Maschinenzyklus Schreiben: Statusregister S auf den Stapelspeicher retten.
Mit 4a adressiert der Stapelzeiger den Speicher. S wird über den Datenbus in die ausgewählte Speicherzelle gebracht (4b) und der Stapelzeiger wieder um 1 vermindert (4c).

5. Maschinenzyklus Lesen: niederwertiges Byte der Sprungadresse (34) holen.
Mit dem Befehlszähler wird der Speicher adressiert (5a). Der ausgelesene Inhalt wird im Adreß-Hilfsregister zwischengespeichert (5b) und der Befehlszählerinhalt um 1 erhöht (5c).

6. Maschinenzyklus Lesen: höherwertiges Byte der Sprungadresse (12) holen.
6a und 6b verlaufen wie 5a und 5b. Mit 6c wird der Inhalt des Adreßregisters in den Befehlszähler übernommen. Damit ist dieser Befehl ausgeführt, und der Mikroprozessor beginnt die Bearbeitung des nächsten Befehls, dessen Adresse er gerade in den Befehlszähler übernommen hat.

10.5.2 Speichersynchronisation (Steuereingang RDY)

Bei der Darstellung der Maschinenzyklen Schreiben und Lesen (Abb. 10.13 und 10.14) wurden genügend schnelle Speicher vorausgesetzt. Sie mußten in der Lage sein, noch innerhalb der Taktperiode auf die Adreß- und R/$\overline{W}$-Ansteuerung zu reagieren. Die meisten der heute verwendeten Halbleiterspeicher haben diese Arbeitsgeschwindigkeit. Dynamische Speicher benötigen zwischendurch aber immer wieder Zeit für den Refreshvorgang und sind in dieser Zeit nicht ansprechbar. Damit ein Mikroprozessor auch mit langsamen Speichern sowie mit dynamischen Speichern zusammenarbeiten kann, besitzt er den Steuereingang RDY (ready = bereit). Langsame Speicher legen, nachdem der Speicher adressiert wurde, den Eingang RDY so lange auf L, bis sie mit dem Schreiben oder Lesen fertig sind.

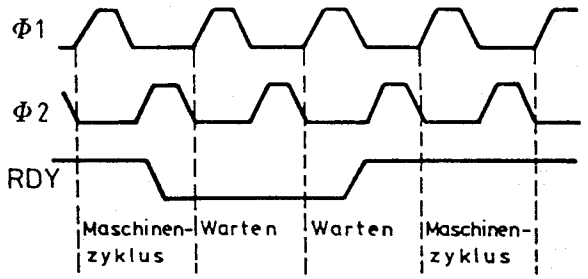


Abb. 10.17 – Steuereingang RDY

Ein L am Eingang RDY bewirkt, daß der Mikroprozessor für eine oder mehrere Taktperioden in den Wartezustand geht. In dieser Zeit läuft zwar der Takt weiter, es wird aber kein Maschinenzyklus durchgeführt. Wird RDY wieder H, so ist die nächste Taktperiode wieder ein Maschinenzyklus. Während des Wartezustands bleibt der Adreßbus aktiviert (und der Datenbus abgeschaltet). Es gibt Mikroprozessoren, die die Warteanforderung über einen Steuerausgang quittieren.

10.5.3 Ein-/Ausgabe

Unter der Ein-/Ausgabe versteht man den Datenverkehr zwischen dem Mikroprozessor und den peripheren Geräten. Dieser Datenverkehr läuft über zwei Teilabschnitte, nämlich einmal zwischen Mikroprozessor und E/A-Baustein und zum anderen zwischen E/A-Baustein und peripherem Gerät. Hier soll nur der erste Teil zwischen CPU und E/A-Baustein betrachtet werden. Da ein Mikrocomputer mehrere Ein-/Ausgabekanäle (I/O-Ports) besitzen kann, müssen die einzelnen Ein- und Ausgänge im Prinzip wie Speicherzellen adressiert werden. Es gibt zwei Methoden, die Ein-/Ausgabekanäle zu adressieren, die isolierte Ein-/Ausgabe und die Speicher-Ein-/Ausgabe (Memory mapped I/O).

Für die **isolierte E/A** muß der Mikroprozessor besondere Ein- und Ausgabebefehle besitzen, z. B. die Befehle IN und OUT. Es handelt sich dabei meist um 2-Byte-Befehle. Das erste Byte enthält den Operationscode, das zweite Byte die E/A-Adresse. Sie wird auf den Leitungen A0...A7 des Adreßbusses ausgegeben. Man kann damit 256 verschiedene E/A-Kanäle adressieren. Der Mikroprozessor muß dabei aber den externen Bausteinen (RAM, ROM, E/A) mitteilen, ob es sich bei der ausgegebenen Adresse um eine Speicheradresse oder eine E/A-Adresse handelt. Er benötigt dafür wenigstens einen

weiteren Steuerausgang. Der Ausgang R/$\overline{W}$ in Abb. 10.6, der zwischen Lesen und Schreiben unterscheidet, muß aufgeteilt werden in die Ausgänge MEM-R/W (Speicher Lesen oder Schreiben) und I/O-R/W (E/A Lesen oder Schreiben). Wird mit diesen Leitungen nicht nur zwischen Schreiben und Lesen unterschieden, sondern soll mit ihnen auch eine genaue zeitliche Steuerung dieser Vorgänge erfolgen, so sind für Lesen und Schreiben getrennte Leitungen erforderlich: MEM-R, MEM-W, I/O-R, I/O-W. Diese Leitungen bilden den wichtigsten Teil des Steuerbusses. Der Befehl IN bewirkt, daß von dem im zweiten Byte adressierten E/A-Kanal ein Byte in den Akkumulator übernommen wird. Mit dem OUT-Befehl wird der Akkumulatorinhalt an den im zweiten Byte adressierten E/A-Kanal ausgegeben. Die isolierte E/A ist z. B. bei den Mikroprozessoren 8080, 8085 und Z80 möglich.

Bei der **Speicher-E/A** werden die E/A-Kanäle einfach wie Speicherbausteine behandelt. Unter bestimmten Speicheradressen findet der Mikroprozessor statt Speicherzellen E/A-Kanäle vor. Es sind dafür keine besonderen Steuerausgänge erforderlich. Alle Befehle, die einen Speicherzugriff beinhalten, können somit auch für die Ein-/Ausgabe verwendet werden.

Ein Vergleich beider E/A-Arten kommt zu folgendem Ergebnis: Die isolierte E/A engt nicht den adressierbaren Speicherbereich ein. Sie benötigt neben besonderen E/A-Befehlen mindestens einen zusätzlichen Steuerausgang. Die Speicher-E/A verringert die Speicherkapazität. In den meisten Fällen ist das aber von untergeordneter Bedeutung. Wenn man z. B. von 65536 Adressen 1024 für die E/A verwendet, bleiben immer noch 64512 Speicheradressen übrig. Für die Speicher-E/A spricht, daß alle Befehle, die Speicherplätze adressieren, auch für die E/A verwendet werden können. Dadurch wird die Ein-/Ausgabe sehr flexibel.

10.5.4 Direkter Speicherzugriff (Steuereingang HLD)

Die im vorigen Abschnitt beschriebene Ein-/Ausgabe wird auch als programmierte E/A bezeichnet, weil sie vom Mikroprozessor bzw. vom gerade abgearbeiteten Programm gesteuert wird. Sie ermöglicht nur Datenverkehr zwischen dem E/A-Kanal und dem Mikroprozessor. Soll z. B. der Speicher von außen geladen werden, so muß Byte für Byte erst in den Mikroprozessor und dann von dort in den Speicher transportiert werden. Dies ist nicht nur umständlich, sondern für größere Datenmengen auch relativ zeitaufwendig. Abhilfe bietet der direkte Speicherzugriff (**d**irect **m**emory **a**ccess, DMA). Direkter Speicherzugriff ist nur unter zwei Voraussetzungen möglich:

1. Das periphere Gerät, von dem die Daten direkt in den Speicher geschrieben werden sollen, muß in der Lage sein, komplett die Steuerung dafür zu übernehmen, z. B. also nacheinander die Adressen auf den Adreßbus legen können.

2. Der Mikroprozessor muß für die Zeit des direkten Speicherzugriffs „abgeschaltet" werden können. Adreß- und Datenbus dürfen nicht von zwei Geräten gleichzeitig angesteuert werden. Viele Mikroprozessoren haben für diesen Zweck den Steuereingang HLD (Hold = Halt). Er wirkt ähnlich wie der RDY-Eingang. Bei wirksamer Ansteuerung des Hold-Eingangs stellt der Mikroprozessor seine Aktivitäten trotz weiterlaufenden Taktes ein. Daten- und Adreßbus werden dabei in den hochohmigen Zustand gesteuert, so daß das periphere Gerät unbeeinflußt vom Mikroprozessor die Kontrolle über Daten- und Adreßbus übernehmen kann. Die meisten Mikroprozessoren mit HLD-Eingang haben auch einen dazugehörigen Ausgang, über den sie dem peripheren Gerät den Haltezustand quittieren.

10.5.5 Programmunterbrechungen (Interrupts, Steueranschlüsse IRQ, NMI und INTA)

Externe Geräte stehen häufig vor dem Problem, den Mikrocomputer auf sich aufmerksam machen zu müssen. Dabei kann es sich z. B. um eine Störungsmeldung handeln oder um die Mitteilung, daß ein langsamer Ausgabevorgang beendet ist und neue Daten ausgegeben werden können, oder um die Ankündigung einer Eingabe. Für dieses Problem gibt es im Prinzip drei Lösungen:

a) Der Mikroprozessor wird über den Bereit-Eingang RDY in den Wartezustand gesteuert, solange das externe Gerät nichts einzugeben hat. Es ist offensichtlich, daß diese Lösung nur in Frage kommt, wenn man sicher ist, daß nach kürzester Zeit (µs) eine Eingabe erfolgt, da es nicht sinnvoll ist, einen Mikroprozessor überwiegend mit Warten zu beschäftigen.

b) In das abzuarbeitende Programm werden Befehlsfolgen eingebaut, durch die alle Eingänge in gleichen Abständen abgefragt werden, ob ein Eingabewunsch vorliegt (polling mode). Dieses Verfahren ist vor allem dann günstig, wenn mit regelmäßigen Eingabewünschen von der Peripherie gerechnet werden kann und die Abfrageabstände darauf abgestimmt sind. Die Leistung des Mikroprozessors wird dann kaum unnötig vermindert.

c) Der Mikrocomputer verfügt über einen oder mehrere Steuereingänge, über die externe Geräte auf sich aufmerksam machen können. Bei einer Meldung unterbricht der Mikroprozessor das laufende Programm und kümmert sich um das externe Gerät. Wenn er damit fertig ist, arbeitet er das Hauptprogramm weiter ab. Dieses

Unterbrechungs-E/A genannte Verfahren ist vor allem bei unregelmäßigen Anforderungen von den externen Geräten am günstigsten, weil der Mikroprozessor sich nur dann den E/A-Geräten zuwendet, wenn es erforderlich ist.

Für die Aufnahme der Unterbrechungsanforderung benötigt der Mikroprozessor mindestens einen Eingang, der meist mit IRQ (interrupt request = Unterbrechungsanforderung) oder INT (interrupt) bezeichnet wird. Wenn dieser Eingang wirksam angesteuert wird, laufen im Mikroprozessor folgende Vorgänge ab, die dem Unterprogrammansprung aufgrund des CALL-Befehls sehr ähnlich sind: Nachdem der gerade bearbeitete Befehl beendet ist, wird das Hauptprogramm unterbrochen. Der Inhalt des Statusregisters und des Befehlszählers wird auf den Kellerspeicher gerettet. Der Befehlszähler wird mit der Beginnadresse des Unterprogramms, das die Unterbrechung bearbeitet, dem sog. Unterbrechungsvektor, geladen, und das Unterprogramm kann abgearbeitet werden. Das Unterprogramm muß mit dem Befehl RET enden, wodurch Befehlszähler und Statusregister wieder mit ihrem alten Inhalt geladen werden und das Hauptprogramm fortgesetzt werden kann.

Es gibt verschiedene Möglichkeiten, wie der Mikroprozessor zum Interruptvektor – der Beginnadresse des Unterbrechungsunterprogramms – kommt. Im einfachsten Fall ist sie fest im Mikroprozessor vorgegeben und wird von der Ablaufsteuerung automatisch bei einer Unterbrechungsanforderung in den Befehlszähler geladen. Eine andere Möglichkeit besteht darin, daß die Ablaufsteuerung wiederum den Befehlszähler mit einer fest vorgegebenen Adresse lädt, unter dieser Adresse aber nicht das Unterprogramm beginnt; sondern hier legt der Programmierer die Beginnadresse des Unterprogramms ab, das dadurch an jeder beliebigen Stelle im Speicher liegen kann. Eine weitere Möglichkeit, dem Befehlszähler den Interruptvektor mitzuteilen, besteht darin, daß der Mikroprozessor über einen Steuerausgang INTA (interrupt acknowledge = Unterbrechungsquittung) dem peripheren Gerät die Unterbrechungsanforderung quittiert und das externe Gerät oder ein besonderer Steuerbaustein daraufhin den Interruptvektor auf den Datenbus legt.

Damit wichtige, z. B. zeitkritische Programmteile nicht durch einen Interrupt unterbrochen werden, kann der IRQ- oder INT-Eingang meist per Befehl unwirksam oder wirksam gesteuert werden. Dazugehörige Befehle heißen z. B. EI (enable interrupt = ermöglicht Unterbrechungen) und DI (disenable interrupts = mache Unterbrechungen unmöglich) oder CLI (clear interrupt) und SEI (set interrupt). Einige Mikroprozessoren haben einen weiteren Interrupteingang mit der Bezeichnung NMI (non maskable interrupt = nicht abschaltbare Unterbrechung). Über diesen Eingang eintreffende Unterbrechungsanforderungen werden in jedem Fall sofort bearbeitet.

Neben den bisher genannten Möglichkeiten bieten einige Mikroprozessoren einen mehrfachen Interrupt. Für mehrere verschiedene Geräte stehen dann Interrupteingänge zur Verfügung, entweder direkt am Mikroprozessor oder über einen Steuerbaustein. Jeder Interrupt wird mit seinem eigenen Unterprogramm beantwortet. Die einzelnen Interrupteingänge können mit unterschiedlicher Priorität ausgestattet sein. Bei gleichzeitigem Auftreten mehrerer Unterbrechungsanforderungen wird die mit der höchsten Priorität zuerst bearbeitet. Ein Interrupt höherer Dringlichkeit unterbricht auch einen gerade laufenden Interrupt niederer Priorität, aber nicht umgekehrt. Umfangreiche Interruptmöglichkeiten erhöhen sehr die Flexibilität und Leistungsfähigkeit eines Mikroprozessors.

10.5.6 Einzelschrittsteuerung (Steuerausgang SYNC)

Für die Fehlersuche bei neuerstellten Programmen ist es häufig sehr hilfreich, wenn man das Programm in Einzelschrittsteuerung Befehl für Befehl abarbeiten und sich nach jedem Befehl die Inhalte aller Register ansehen kann. Viele Mikroprozessoren haben dafür den Ausgang SYNC (Synchronisation). An diesem Ausgang liefert der Mikroprozessor jedesmal einen Impuls, wenn er einen Operationscode aus dem Speicher holt, also jedesmal, wenn ein neuer Befehl beginnt. Mit diesem Impuls kann der Mikroprozessor über den Eingang RDY zu Beginn eines jeden Befehls in den Wartezustand gesteuert werden. Es kann mit dem SYNC-Signal auch ein Interrupt ausgelöst und damit ein Unterprogramm angesteuert werden, das sofort das Auslesen der Register ermöglicht. Es muß dann aber mit Hilfe der Adreßdekodierung dafür gesorgt werden, daß das Unterprogramm selbst nicht in Einzelschrittsteuerung durchlaufen wird.

10.5.7 Rücksetzen (RESET, Steuereingang RES)

Der Rücksetzeingang dient dazu, den Mikroprozessor aus einem nach Anlegen der Speisespannung oder sonst entstandenen undefinierten Zustand wieder in einen definierten Anfangszustand zu bringen. Bei einigen Mikroprozessoren wird durch eine wirksame Ansteuerung des RESET-Eingangs der Befehlszähler auf Null gesetzt. Der Vorgang ist unabhängig von der Position des Befehlszählers. Das bedeutet, daß man einen Programmablauf mit einem RESET abbrechen und wieder bei Null beginnen kann. Jeder µProzessor besitzt einen RESET-Eingang. Dieser wird normalerweise mit einer Taste (RESET-Taste) und einer RC-Kombination beschaltete. Mit der Taste läßt sich der Befehlszähler (z. B. bei einer Störung) jederzeit zurückstellen. Das

Programm beginnt wieder an der Adresse 0. Beim Einschalten der Versorgungsspannung erfolgt ein sogenannter Auto-RESET. Der entladene Kondensator legt durch den Aufladevorgang kurzzeitig 0-Potential an den Eingang. Der Zähler steht nach dem Einschalten immer auf 0! Dieses wiederum bedeutet, **daß ab Adresse 0 im Speicher immer ein Programm beginnen muß!** Andere Mikroprozessoren steuern nach RESET eine bestimmte andere Speicherzelle an, aus der sie sich die vom Programmierer dort abgelegte Beginnadresse holen.

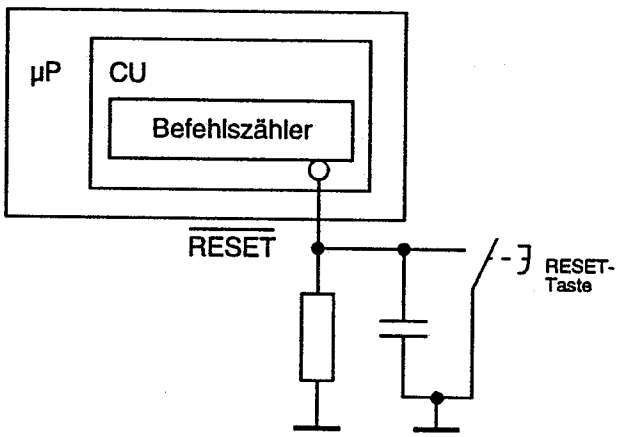


Abb. 10.18 – Ansteuerung des Befehlszählers

10.6 Befehls- und Adressierungsarten

Auch in diesem Abschnitt wird wieder von dem 8-Bit-Mikroprozessor mit 16-Bit-Adreßbus ausgegangen. Wie bereits aus den vorigen Abschnitten bekannt ist, enthält bei diesen Prozessoren das erste Befehlsbyte den Operationscode[1]. Bei einer Länge des Operationscodes von 8 Bit sind theoretisch 256 verschiedene Befehle möglich. Die meisten Mikroprozessoren haben einen Befehlsvorrat mit 40 bis 80 verschiedenen Befehlen. Bezieht man die unterschiedlichen Register und Adressierungsarten ein, so liegt die Zahl der ausgenutzten Operationscodekombinationen zwischen 150 und 250. Bevor auf die verschiedenen Befehle näher eingegangen wird, werden erst einmal die unterschiedlichen Adressierungsverfahren beschrieben.

10.6.1 Adressierungsverfahren

10.6.1.1 Allgemeines

Die meisten Befehle müssen zwei bis drei Adressen beinhalten. Z. B. muß in allen Befehlen, die einen Datentransport bewirken, die Quellenadresse (Wo kommt das Byte her?) und die Zieladresse (Wo soll das Byte hin?) angegeben sein. Ein Additionsbefehl muß drei Adressen enthalten: die Quellenadressen beider Summanden und die Zieladresse für das Ergebnis. Lägen alle drei Adressen im Speicher, so müßte der Befehl bei Speicheradressen von 16 Bit eine Länge von 7 Byte haben, nämlich 1 Byte für den Operationscode und 3 mal 2 Byte für die Adressen. Da das sehr unhandlich und speicherintensiv ist, liegt bei allen Befehlen höchstens eine Adresse im Speicher. Die anderen Quellen oder Ziele sind immer Register des Mikroprozessors. Die Quellen- und Zielregister werden nicht besonders adressiert, ihre Angabe ist immer im Operationscode enthalten. Ist eine Adresse bereits im Operationscode enthalten, so spricht man von impliziter Adressierung (implizit = mit inbegriffen). Register sind bei Mikroprozessoren immer implizit adressiert. Bei nur einer Speicheradresse haben die Befehle daher eine Länge von maximal 3 Byte.

Die im folgenden dargestellten Adressierungsverfahren beziehen sich alle auf die Adressierung des Operanden, der im Speicher liegt oder in den Speicher gebracht werden soll. Ein einzelner Mikroprozessor erlaubt nicht alle angegebenen Adressierungsverfahren. Jeder Typ hat ein anderes Adressierungsrepertoire.

10.6.1.2 Unmittelbare (immediate) Adressierung

Bei der unmittelbaren Adressierung folgt dem Operationscode (OP-Code) als zweites Byte nicht die Datenadresse, sondern gleich die Daten selbst. Bei den meisten Mikroprozessoren kann dem OP-Code nur ein Datenbyte folgen. Diese Adressierungsart eignet sich besonders für das Laden der Register mit konstanten Anfangswerten und zum Rechnen und Vergleichen mit Konstanten. Ein Befehl mit unmittelbarer Adressierung ist z. B:

LDA # 0F

Das Zeichen # symbolisiert die unmittelbare Adressierung. Der Befehl bewirkt, daß der Akkumulator unmittelbar mit dem Byte 0F, also der Bitfolge 00001111 geladen wird. LDA steht für: Lade den Akkumulator.

10.6.1.3 Direkte (absolute) Adressierung

Befehle mit direkter Adressierung enthalten nach dem OP-Code im zweiten und dritten Byte die absolute Adresse, unter der der Operand im Speicher zu finden ist. Es ist eine sehr einfache und universelle Adressierungsart mit einer Befehlslänge von drei Byte. Ein Befehl mit dieser Adressierung ist z. B.:

STX-1A2B

1) Eine Ausnahme bildet der Mikroprozessor Z80, bei dem der Operationscode bei einigen Befehlstypen zwei oder drei Byte lang ist.

Der Befehl bewirkt, daß der Inhalt des X-Registers in der Speicherzelle mit der Adresse 1A2B abgespeichert wird (STX = store X-Register = speichere X-Register).

10.6.1.4 Direkte Seite-Null- (Zero-Page)-Adressierung

Man kann sich den gesamten Speicherbereich mit den 65536 Adressen so vorstellen, daß er aus 256 Seiten mit je 256 Speicherzellen besteht. Das hochwertige Adreßbyte (Adreßbit A8...A15) adressiert dann eine der 256 Seiten und das niederwertige Adreßbyte (Adreßbit A0...A7) eine der 256 Speicherzellen auf dieser Seite. Wenn man im OP-Code selbst schon eine bestimmte Seite kennzeichnet – meist die Seite Null –, dann braucht die Adresse nur noch aus dem niederwertigen Adreßbyte zu bestehen, und die Befehlslänge verkürzt sich auf zwei Byte. Mit dieser Adressierungsart können allerdings nur die ersten 256 Speicherzellen angesprochen werden. Ein Beispiel für diese Adressierung ist:

ADCZ-E0

Das Z kennzeichnet die Zero-Page-Adressierung. Mit diesem Befehl wird der Inhalt der Speicherzelle 00E0 zum Inhalt des Akkumulators addiert. Wegen der um ein Byte kürzeren Befehle können mit dieser Adressierungsart der Programmspeicherbedarf und die Verarbeitungszeit etwas verringert werden.

10.6.1.5 Indizierte (indexed) Adressierung

Bei der indizierten Adressierung enthält der OP-Code die Anweisung, zu der im zweiten und dritten Byte angegebenen Anfangsadresse erst den Inhalt eines Hilfsregisters, z. B. des Indexregisters X, zu addieren und dann den Speicher zu adressieren. Wie bereits im Abschnitt 10.4.7 beschrieben, vereinfacht diese Adressierungsart das Programmieren erheblich, wenn ganze Listen abzuarbeiten sind. Ein Beispiel für diese Befehlsart ist:

LDA-1234,X

Dieser Befehl veranlaßt, daß der Akkumulator mit dem Inhalt der Speicherzelle, deren Adresse sich aus der Summe aus 1234 und dem Inhalt des X-Registers ergibt, geladen wird.

10.6.1.6 Relative Adressierung

Die relative Adressierung gibt im Adreßteil des Befehls den Abstand des gewünschten Speicherplatzes vom momentanen Befehlszählerstand an. Die relative Adresse steht im zweiten Byte. Das Byte wird als siebenstellige Dualzahl mit Vorzeichen interpretiert (A7 = 0: positive Zahl, A7 = 1: negative Zahl). Damit kann die adressierte Speicherzelle in folgendem Bereich liegen: (momentaner Befehlszählerstand – 128) bis (momentaner Befehlszählerstand + 127). Die relative Adressierung wird bei Sprungbefehlen verwendet und bietet hier den Vorteil, daß die Sprungadressen nicht verändert zu werden brauchen, wenn das Programm in einem anderen Speicherbereich abgelegt wird. Beispiel:

BCS-3A

BCS steht für Branch on Carry Set = Verzweige, wenn das Übertragsbit im Statusregister im Zustand 1 ist. Wenn das Carry-Flag 1 ist, springt das Programm zur Adresse: momentaner Befehlszählerstand + 3A.

10.6.1.7 Indirekte Adressierung

Bei Befehlen mit indirekter Adressierung wird im zweiten und dritten Byte eine Speicherzelle adressiert, in der sich nicht die gewünschten Daten, sondern erst die Adresse befindet, unter der die gewünschten Daten im Speicher abgelegt sind. Diese Adressierungsart hat u. a. den Vorteil, daß beim Programmieren die endgültigen Adressen noch nicht bekannt sein müssen. Die indirekte Adressierung kann mit der indizierten kombiniert werden. Beispiel für indirekte Adressierung:

JMP-(FFA0)

Die Klammern sind eine Möglichkeit, die indirekte Adressierung zu kennzeichnen. Der Befehl bewirkt, daß das Programm zu der Adresse springt, die in den Speicherzellen mit den Adressen FFA0 und FFA1 abgelegt ist. Da die Adresse 2 Byte lang ist, wird sie in der im Befehl angegebenen Adresse und der folgenden abgelegt.

10.6.1.8 Implizite (implied) Adressierung

Die implizite Adressierung kommt mit Befehlen aus, die nur 1 Byte lang sind, weil bei ihnen die Adresse im OP-Code enthalten ist. Im Abschnitt 10.6.1.1 wurde gezeigt, daß die Mikroprozessoren diese Adressierung zur Auswahl des gewünschten Registers verwenden. Viele Mikroprozessoren benutzen eine Variante der impliziten Adressierung, die registerindirekte Adressierung, in großem Umfang auch für die Speicheradressierung. Bei diesem Verfahren enthält das im OP-Code gekennzeichnete 16-Bit-Register die Speicheradresse des Operanden (s. Abschnitt 10.4.6). Beispiel:

MOV X,M

Durch diesen ein Byte langen Befehl wird das X-Register mit dem Inhalt der Speicherzelle geladen, deren Adresse im Daten-Adreß-Register HL angegeben ist.

Die hier beschriebenen einfachen Adressierungsverfahren können bei einzelnen Mikroprozessoren auch noch erweitert und miteinander kombiniert auftreten.

10.6.2 Befehlsarten

10.6.2.1 Allgemeines

Fast alle Befehle eines Mikroprozessors beziehen sich auf ein oder mehrere CPU-Register. Bevor man sich mit den einzelnen Befehlen eines Mikroprozessors beschäftigt, muß man sich eine Übersicht verschaffen, welche für den Programmierer wichtigen Register der Prozessor enthält. Für den Programmierer sind alle die Register wichtig, die er per Programm beeinflussen kann. Die Hersteller der Mikroprozessoren geben in den Datenblättern ein sog. Programmiermodell ihres Prozessors an, das im Gegensatz zum Blockschaltbild nur die für die Programmierung wichtigen Register enthält. Abb. 10.19 zeigt das Programmiermodell des Mikroprozessors aus Abb. 10.6. Aus ihm ist außerdem die Länge der einzelnen Register erkennbar. Bei der nun folgenden Besprechung der Befehlsarten sind die Befehle nach ihrer Funktion eingeteilt.

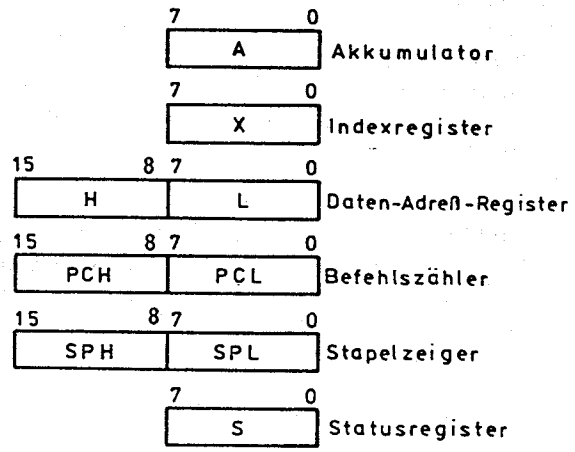


Abb. 10.19 – Programmiermodell

10.6.2.2 Transferbefehle

Transferbefehle transportieren Daten innerhalb des Mikrocomputers, ohne sie dabei zu verändern. Man kann sie weiter unterteilen in Befehle, die die Daten von einem Register in ein anderes bringen, in Befehle, die Daten von einem Register in den Speicher oder zur E/A transportieren und in Befehle, die umgekehrt die Daten vom Speicher oder der E/A in ein CPU-Register laden.

Beispiele für Register-Register-Transferbefehle:

TAX (Transfer Akkumulator zum Indexregister X)

MOV SP, HL (Lade den Stackpointer mit dem Inhalt des Daten-Adreß-Registers HL)

Alle Register-Register-Transferbefehle haben nur eine Länge von einem Byte.

Beispiele für Register-Speicher-Transferbefehle:

STX ... (Speicher Inhalt des Indexregisters X unter Adresse ... ab)

OUT ... (Gib Inhalt des Akkumulators auf E/A-Kanal Nr. ... aus)

PHA (push accu on stack = Akkumulatorinhalt wird in die durch den Stapelzeiger adressierte Speicherzelle gebracht)

Der Push-Befehl ist ein Byte lang. Die Länge der übrigen Befehle hängt von der jeweiligen Adressierungsart ab.

Beispiele für Speicher-Register-Transferbefehle:

LDA ... (Lade den Akkumulator mit dem Inhalt der adressierten Speicherzelle)

IN ... (Lade den Akkumulator mit dem Inhalt des E/A-Kanals Nr. ...)

PLA (pull accu from stack = Lade den Akkumulator mit dem Inhalt der Speicherzelle, die durch den Stapelzeiger adressiert ist)

10.6.2.3 Arithmetische Befehle

Bei den 8-Bit-Mikroprozessoren beschränken sich die arithmetischen Befehle meist auf Additions-, Subtraktions-, Inkrementier- und Dekrementierbefehle, die allerdings in vielen Varianten auftreten können. Beispiele:

ADC ... (Addiere den Inhalt der adressierten Speicherzelle unter Berücksichtigung des Übertrags zum Akkumulatorinhalt)

ADD X (Addiere Inhalt des X-Registers zum Akkumulatorinhalt ohne Übertrag)

SBB ... (Subtrahiere den Inhalt der adressierten Speicherzelle unter Berücksichtigung der Entlehnung – des borrows B – vom Akkumulatorinhalt)

INA (Inkrementiere den Akkumulator = erhöhe Akkumulatorinhalt um 1)

DEX (Dekrementiere X-Register = vermindere X-Register-Inhalt um 1)

10.6.2.4 Logische Befehle

Zu den logischen Befehlen gehören alle Befehle, die die Daten über die UND-, ODER-, NICHT-, XOR-Funktion oder über Vergleicher verknüpfen. Beispiele:

AND # 0F (UND-verknüpfe den Akkumulatorinhalt mit der Konstanten 0F; durch diesen Befehl wird das obere Halbbyte des Akkumulators ausgeblendet. Das Ergebnis steht wieder im Akkumulator).

XRA ... (Verknüpfe Akkumulatorinhalt und Inhalt der adressierten Speicherzelle über Exklusiv-ODER)

CPX ... (Compare – Vergleiche – den Inhalt des X-Registers mit dem Inhalt der adressierten Speicherzelle. Der Mikroprozessor rechnet X-Inhalt minus Speicherinhalt. Das Ergebnis – Null, positiv oder negativ – wird durch Setzen der entsprechenden Flags gekennzeichnet. Der Inhalt des X-Registers wird nicht verändert).

10.6.2.5 Registerbefehle

Hierunter fallen alle Befehle, die Registerinhalte durch Verschieben, Rotieren und Setzen oder Löschen einzelner Bit verändern. Beispiele:

ROL (Rotiere Akkumulatorinhalt linksherum über das Carry-Bit. Der Inhalt des Akkumulators wird um eine Position nach links verschoben. Der Inhalt des höchsten Bit A7 geht in das Übertragsbit und dessen Inhalt geht in das niedrigste Bit A0 des Akkumulators).

SHR (Schiebe Akkumulatorinhalt um eine Stelle nach rechts. Inhalt von A0 geht in das Übertragsbit, dessen Inhalt geht verloren. In A7 wird eine 0 eingeschoben).

SEC (Setze Carry-Flag; das Übertragsbit wird auf 1 gesetzt).

10.6.2.6 Sprungbefehle

Die Sprungbefehle werden unterteilt in die unbedingten Sprungbefehle, bei denen auf jeden Fall das Programm zu einer anderen Adresse springt, und in die bedingten Sprungbefehle, bei denen das Springen vom Erfülltsein einer Bedingung abhängig gemacht wird. Die Erfüllung oder Nichterfüllung der Sprungbedingung wird durch den Zustand der Statusbit gekennzeichnet. Bedingte Sprünge ermöglichen Programmverzweigungen. Während die unbedingten Sprungbefehle allgemein mit JMP (jump) bezeichnet werden, verwenden die einzelnen Mikroprozessoren für die bedingten Sprungbefehle unterschiedliche Bezeichnungen wie z. B. JC ... (jump conditioned) oder B ... (branch = verzweige). Beispiele:

JMP ... (Springe unbedingt zur angegebenen Adresse. Bei diesem Befehl sind die direkte, die indirekte und die registerindirekte Adressierung üblich).

JNZ ... (jump on not zero = Springe, wenn Ergebnis ≠ 0, zur angegebenen Adresse).

BMI ... (branch on minus = Springe (verzweige), wenn Ergebnis negativ, zur angegebenen Adresse).

10.6.2.7 Unterprogrammbefehle

Hierzu gehören die Befehle, durch die ein Unterprogramm angesprungen wird, und die Befehle, durch die man vom Unterprogramm wieder in das Hauptprogramm zurückkehrt. Einige Mikroprozessoren haben dafür je einen Befehl. Bei anderen kann das Anspringen von Unterprogrammen und das Zurückkehren von Bedingungen abhängig gemacht werden. In jedem Fall wird beim Aufruf eines Unterprogramms zumindest der Befehlszählerstand auf den Stapelspeicher gerettet. Übliche Bezeichnungen für die Befehle sind JSR (jump to subroutine) und CALL (call = aufrufen). Beispiele:

JSR ... (Springe zu dem an der angegebenen Adresse beginnenden Unterprogramm unter Rettung des Befehlszählerinhalts).

CNC ... (call on no carry = Rufe das an der angegebenen Adresse beginnende Unterprogramm auf, wenn das Carry-Flag = 0 ist).

RTS ... (return from subroutine = Kehre in das Hauptprogramm zurück. Die Rücksprungadresse ist vom Stapelspeicher zu holen).

RZ ... (return on zero = Kehre vom Unterprogramm zurück, wenn das Ergebnis = 0 ist).

10.6.2.8 Unterbrechungsbefehle

Diese Befehle stehen im Zusammenhang mit Unterbrechungsanforderungen. Hierzu gehören Befehle wie CLI, SEI und DI (s. Abschnitt 10.5.5), die einen Interrupt erlauben oder verbieten. Einige Prozessoren können einen Interrupt auch per Befehl anstoßen; der zugehörige Befehl heißt meist BRK (break = unterbreche) und wirkt ebenso wie ein von außen kommender Hardware-Interrupt. Bei manchen Prozessoren muß die Rückkehr von einer Interruptroutine anders behandelt werden als die Rückkehr von einem Unterprogramm. Der besondere Rückkehrbefehl heißt z. B. RTI (return from interrupt).

10.6.2.9 Sonstige Befehle

Zu dieser Gruppe gehören alle Befehle, die nicht einer anderen Gruppe zugeordnet werden können. Der wichtigste ist der NOP-Befehl (no operation), der nichts bewirkt. Er wird z. B. dazu verwendet, um programmierte Zeitschleifen auf die gewünschte Länge zu bringen oder um vorübergehend Löcher auszufüllen, die bei der Fehlersuche in Programmen entstehen, wenn ein Befehl aus dem Programm entfernt wird.

10.7 Erweiterung eines Mikroprozessors zum Mikrocomputer

Aus Abschnitt 10.2.2 ist bereits bekannt, daß ein Mikroprozessor mindestens um den Speicher und um die E/A erweitert werden muß, damit ein funktionsfähiger Mikrocomputer entsteht. Je nach verwendetem Mikroprozessor und je nach den gewünschten Leistungsmerkmalen kommen noch weitere Bausteine hinzu. Manche CPU benötigen einen externen Taktgenerator, fast alle mehr oder weniger aufwendige Schaltnetze zur Bildung des Steuerbusses. Für alle Mikroprozessoren, die einen Teil der Anschlüsse durch Multiplexbetrieb doppelt ausnutzen, ist ein entsprechender Demultiplexer erforderlich. Weiter können hinzukommen: Bausteine für mehrfachen Interrupt und für die Prioritätssteuerung der Interrupts, Bausteine für die Steuerung des direkten Speicherzugriffs und Hilfsbausteine wie z. B. Zeitgeber (Timer), die von der CPU für eine vorwählbare Zeit gestartet werden können und dann einen Interrupt auslösen. Von all diesen Möglichkeiten wird hier nur auf den Speicher und auf einen einfachen E/A-Baustein eingegangen.

10.7.1 Speicher eines Mikrocomputers

Mikrocomputer verwenden als Arbeitsspeicher Halbleiterspeicher, und zwar sowohl Festwertspeicher (ROM, PROM, EPROM) als auch Schreib-/Lesespeicher (RAM).

In den Festwertspeichern sind je nach Anwendung des Mikrocomputers Programme des Betriebssystems (Monitor, Assembler, Editor, Disassembler, Interpreter ...) oder Anwenderprogramme abgelegt. Ob ROM, PROM oder EPROM eingesetzt werden, ist überwiegend eine Frage der Stückzahlen. Bei hohen Stückzahlen (>1000) sind die bereits in der Fertigung maskenprogrammierten ROM am kostengünstigsten. Bei Kleinserien werden PROM eingesetzt, die der Anwender selbst programmieren kann. In der Entwicklung, für Einzelstücke, sind die EPROM am günstigsten, weil sie bei Programmierfehlern und Programmänderungen gelöscht und wiederverwendet werden können. Für die Anschaltung an den Mikroprozessor spielt es keine Rolle, ob ROM, PROM

oder EPROM verwendet werden. Es genügt also, z. B. nur ROM näher zu betrachten.

Festwertspeicher sind meist wortorganisiert, sie geben bei wirksamer Ansteuerung ein ganzes Byte aus. Die z. Z. üblichen Kapazitäten liegen bei 1 bis 256 KByte. Sie haben folgende Anschlüsse: 10 ... 18 Adreßeingänge je nach Kapazität, 8 Datenausgänge, 1–2 CS-Eingänge (CS = chip select = Bausteinauswahl) und die Anschlüsse für die Stromversorgung. Abb. 10.20 zeigt ein Beispiel für den Anschluß des ROM 2332 mit einer Kapazität von 4 KByte an die Bussysteme eines Mikrocomputers.

Wenn man einen Speicherbaustein an einen Mikroprozessor anschließt, muß bekannt sein, welcher Adreßbereich den Baustein ansprechen soll. Geht man von dem einfachen Fall aus, daß der Speicher nur aus Bausteinen mit einer Kapazität von 4 KByte aufgebaut wird, bietet es sich an, den gesamten Adreßbereich von 64 KByte aufzuteilen in 16 Teilbereiche mit je 4 KByte. Die vier Adreßleitungen mit der höchsten Wertigkeit, A12 bis A15, werden zur Auswahl des Teilbereichs verwendet. Die übrigen 12 Adreßleitungen A0...A11 bestimmen dann, welches der 4096 Byte eines Teilbereiches gemeint ist. Wie Abb. 10.20 zeigt, sind die Adreßleitungen A0...A11 direkt mit den gleichnamigen Eingängen des ROM verbunden. Die vier Adreßleitungen A12...A15 gehen auf die Eingänge A bis D des Dekoders SN 74 154. Dieses IC setzt eine vierstellige Dualzahl in den 1-aus-16-Code um und kennzeichnet damit die 16 Teilbereiche des Speichers, und zwar der Ausgang 0 den Adreßbereich von 0000 bis 0FFF, der Ausgang 1 den Adreßbereich 1000 bis 1FFF, ..., der Ausgang 10 den Adreßbereich von A000 bis AFFF und der Ausgang 15 den Adreßbereich F000 bis FFFF. Der Eingang CS1 des ROM ist in Abb. 10.20 verbunden mit dem Ausgang 12 des Dekoders. Der ROM wird daher nur angesprochen vom Adreßbereich C000 bis CFFF. Der zweite Eingang zur Bausteinauswahl CS2 ist mit der Leitung MEM/R des Steuerbusses verbunden. Über diese Leitung wird der Zeitpunkt festgelegt, zu dem der ROM das Datenbyte auf den Datenbus gibt. Die Datenausgänge des ROM Q0... Q7 sind auf den Datenbus geführt.

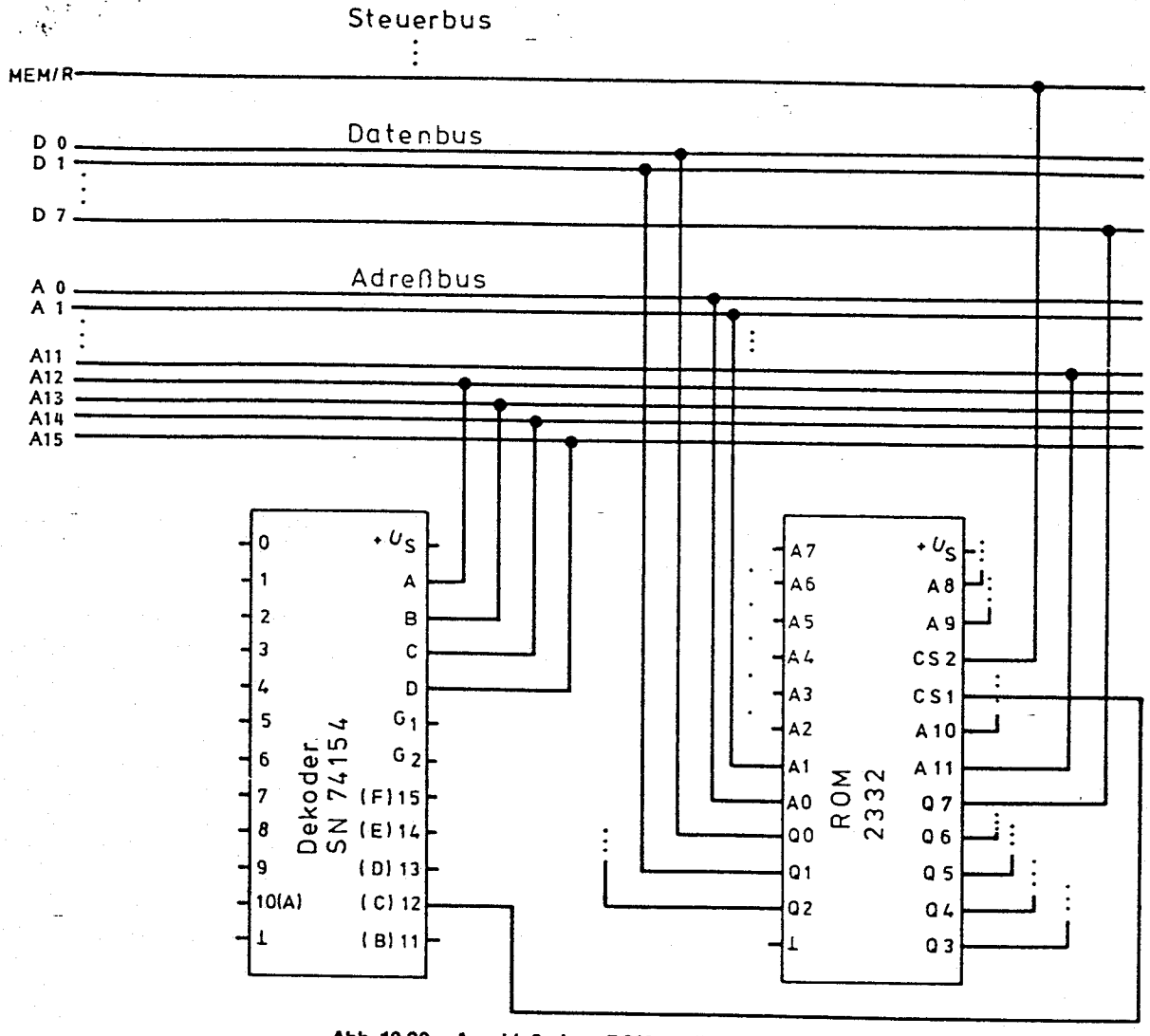


Abb. 10.20 – Anschluß eines ROM an die Bussysteme

Solange man keine Speicher-IC mit einer Kapazität von 64 KByte verwendet, ist immer eine Adreßdekodierung erforderlich. Je kleiner die Kapazität der einzelnen IC ist, desto aufwendiger wird die Adreßkodierung. Bei voll ausgenutztem Adreßbereich und einer Speicherbausteinkapazität je Baustein, die kleiner als 4 KByte ist, wird die Adreßdekodierung mehrstufig.

Ähnlich wie die ROM werden auch die RAM mit den Bussystemen verbunden. Es gibt aber einige Unterschiede. Zum einen haben die derzeitigen RAM meist nur eine Datenbreite von einem oder vier Bit. Es sind also acht oder zwei IC mit allen Steuer- und Adreßeingängen parallelzuschalten, damit gleichzeitig ein Byte ausgelesen oder eingeschrieben werden kann. Zum anderen haben die RAM noch einen weiteren Steuereingang, über den dem RAM mitgeteilt wird, ob gerade gelesen oder geschrieben werden soll. Dieser zusätzliche Eingang ist mit der Leitung MEM/W des Steuerbusses verbunden.

10.7.2 PPIA als Beispiel für einen E/A-Baustein

Ein PPIA (programmable peripheral interface adapter = programmierbare, periphere Schnittstellenanpassung) ist ein LSI-IC, das einen einfachen Anschluß von peripheren Geräten über 8-Bit-Busleitungen ermöglicht, wobei die Busleitungen als Ein- und Ausgänge programmiert werden können. Es gibt eine Vielzahl anderer E/A-Bausteine, angefangen vom einfachen Pufferspeicher bis hin zum sehr aufwendigen Parallel-/Serienumsetzer für synchrone und asynchrone Datenübertragung. Der PPIA-Baustein dient hier als Beispiel, weil er sehr universell ist und häufig eingesetzt wird.

Abb. 10.21 zeigt das Blockschaltbild eines PPIA. Die Ein- und Ausgänge am oberen Rand führen zu den peripheren Geräten, die am unteren Rand zum Mikroprozessor. Der als Beispiel verwendete PPIA besitzt zwei E/A-Kanäle, Port A und Port B, die beide symmetrisch aufgebaut sind. Die Verbindung zum peripheren Gerät besteht aus jeweils acht bidirektionalen Datenleitungen (PA0...PA7 und PB0...PB7) und zwei Steuerleitungen für Unterbrechungsanforderung (IRQ) und Unterbrechungsquittung (INTA). Zu jedem Kanal gehört neben der Unterbrechungssteuerung ein Datenrichtungsregister (DDA und DDB), ein Datenregister (DA und DB) sowie der bidirektionale periphere Datenpuffer. Über das Datenrichtungsregister wird festgelegt, ob die einzelnen Leitungen PA0...PA7 bzw. PB0...PB7 als Eingang oder Ausgang arbeiten. Eine 0 in einer Bitposition des Datenrichtungsregisters macht die zugehörige Leitung zu einem Eingang, eine 1 läßt sie als Ausgang arbeiten. In eine als Ausgang definierte Leitung

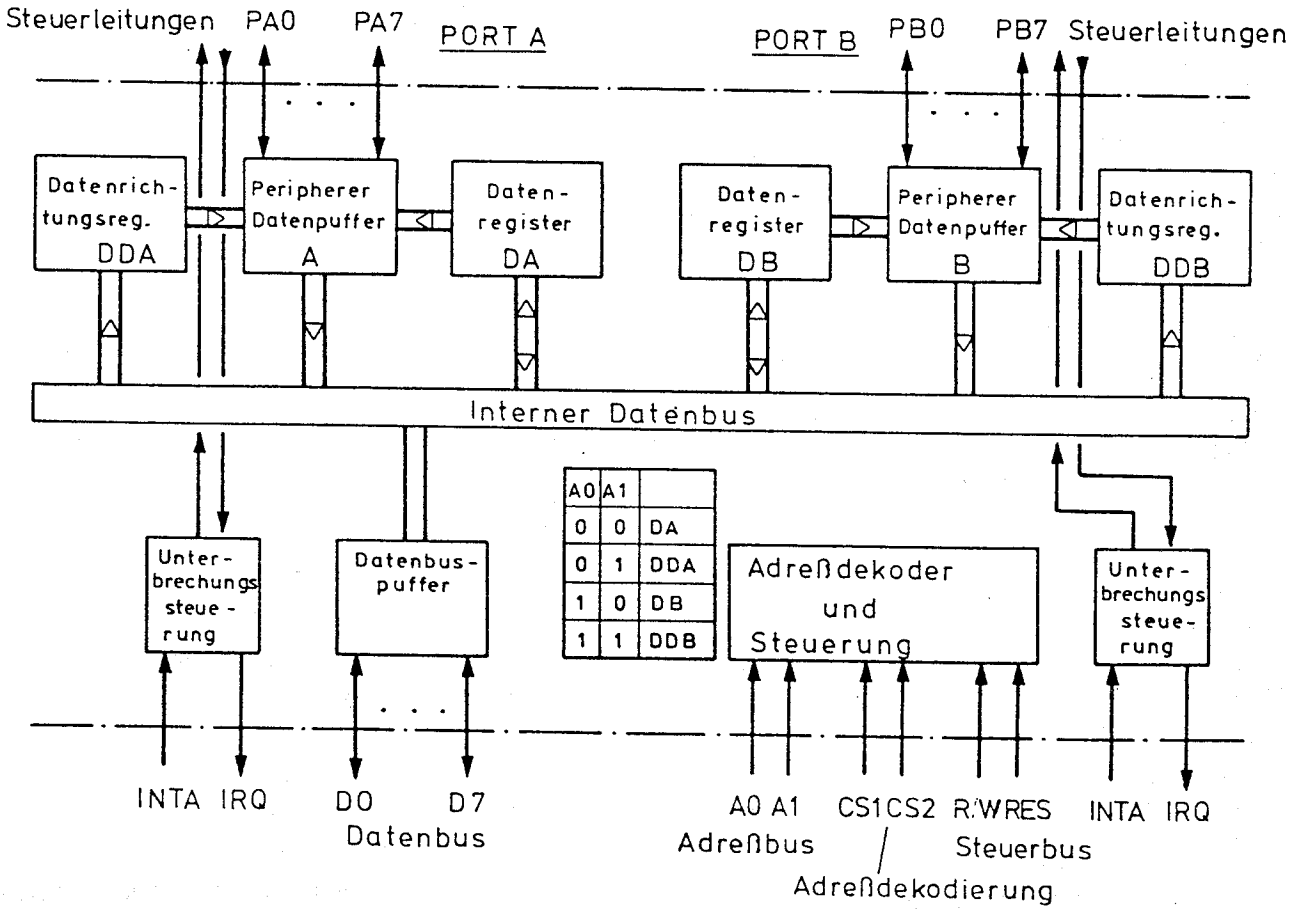


Abb. 10.21 – Blockschaltbild eines PPIA

wird ständig der Zustand des entsprechenden Bit im Datenregister ausgegeben. Über eine als Eingang arbeitende Leitung steuert das periphere Gerät die entsprechende Bitposition im Datenregister in den gewünschten Zustand. Die übrigen Ein- und Ausgänge haben folgende Aufgaben:

Adreßleitungen A0 und A1:
Über diese beiden Adreßleitungen steuert der Mikroprozessor, ob der Datenbus D0 bis D7 zu einem der beiden Datenrichtungsregister oder einem der beiden Datenregister durchgeschaltet wird. Die Zuordnung zwischen Adresse und Register ist in Abb. 10.21 angegeben.

Bausteinauswahlleitungen CS1 und CS2:
Über diese Eingänge wird der Baustein aktiviert. Sie sind mit der Adreßdekodierung verbunden. Damit wird sichergestellt, daß der Baustein nur angesprochen werden kann, wenn seine Adresse auf dem Adreßbus liegt.

R/W:
Über den Steuerbus und diesen Eingang teilt der Mikroprozessor dem PPIA mit, ob er in das adressierte Register schreiben oder aus ihm lesen will.

RES:
Eine wirksame Ansteuerung dieses Eingangs setzt alle Register auf Null. Damit werden alle Port-Leitungen zu Eingängen.

Einige dieser programmierbaren Ausgangsbausteine lassen sich sehr gut für den Datenaustausch zwischen Computer und Peripherie bzw. zwischen Computern einsetzen. Für diese Übertragungen sind im Laufe der Zeit Standards entwickelt worden. Der serielle Datenaustausch über das öffentliche Telekommunikationsnetz ist weitgehend durch die CCITT-Empfehlungen geregelt. Für die parallele Übertragung gilt der IEC-Bus als internationaler Standard. Weit verbreitet in der Computerwelt ist die **CENTRONICS**-Schnittstelle. Über sie wird der parallele Datenaustausch zwischen Rechner und Drucker abgewickelt. Sie wurde von der Firma CENTRONICS für ihre Drucker entwickelt und von fast allen anderen Firmen übernommen. Leider bestehen verschiedene Normen, so daß immer wieder Anpassungsprobleme auftreten.

Die asynchrone serielle Standardschnittstelle **RS 232 C** (V.24) ist die meistverwendete Geräteverbindung. Trotz der Normung durch verschiedene Institutionen [CCITT (V.24), EIA (RS 232C), DIN (66020)] ist auch ihre Anwendung nicht immer problemlos.

Die RS 232C läßt sich sehr gut aus den Portbausteinen 8250 bzw. 8251 herstellen. Sie dient zur seriellen Datenübertragung zwischen einer Datenendeinrichtung (DEE, Terminal) und einer Datenübertragungseinrichtung (DÜE, Modem → Modulator / Demodulator) oder zwischen zwei DEE, z. B. zwischen Computer und Drucker.

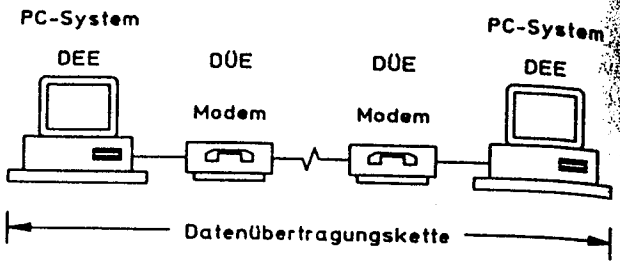
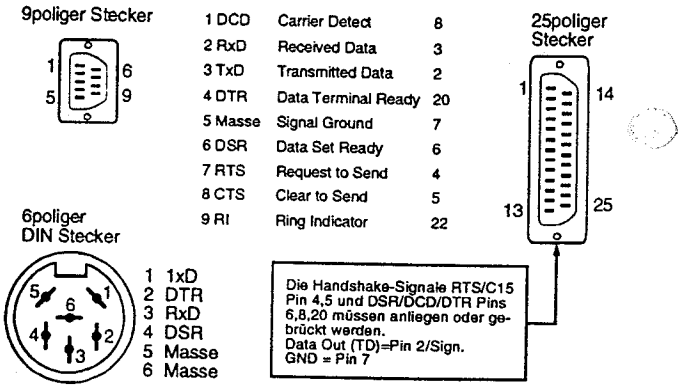


Abb. 10.22 – Datenübertragungskette

Die Daten liegen im Computersystem in paralleler Form vor. Zum seriellen Senden werden sie über ein Schieberegister in einem Parallel-/Seriell-Wandler geschickt. Das Modem wandelt sie dann in analoge Tonfrequenzen um. Hier tauchen dann schon wieder Probleme auf. Die Normung legt nur die Bedingungen zwischen DEE und DÜE fest, so daß man bei den Modems auf verschiedene Typen treffen kann. Da die Reichweite einer RS-232C-Schnittstelle im Durchschnitt bei ca. 30 m liegt, wird ein Modem meistens glücklicherweise nur für Datenaustausch auf Telefonleitungen benötigt. Verbindet man einen Computer mit einem Drucker oder einem anderen Computer über kleine Entfernungen, kann auf ein Modem verzichtet werden. Die Kopplung erfolgt dann über ein sogenanntes „Nullmodem"-Kabel und wird durch ein Datenübertragungsprogramm gesteuert. In diesem müssen die beiden Datenleitungen und die Quittungsleitungen gekreuzt werden, da die jeweilige Sendeleitung am anderen Ende auf eine Empfangsleitung treffen muß.

In der Abb. 10.23 wird die Pin-Belegung der 25poligen und 9poligen Stecker für Personalcomputer dargestellt.



DCD ⟶ Empfangssignalpegel
DSR ⟶ Betriebsbereitschaft
CTS ⟶ Sendebereitschaft
RTS ⟶ Sendeteil einschalten
RxD ⟶ Empfangsdaten
TxD ⟶ Sendedaten
DTR ⟶ Datenendeinrichtung betriebsbereit

Abb. 10.23 – Steckerbelegung RS 232C

Der 25polige Stecker ist der Standard. In den meisten modernen Systemen wird aber der 9polige eingesetzt. Die Abbildung zeigt Unterschiede in der Pinbelegung. Die Funktion der einzelnen Leitungen ist identisch. Etwas ungewöhnlich ist die Verbindung mit dem 6poligen DIN-Stecker. Er wird noch bei einigen Druckern verwendet.

Ein Drucker mit **paralleler** Schnittstelle benötigt mindestens 8 Datenleitungen zur Zeichenübertragung und 2 Leitungen zum Quittungsaustausch. Eine Leitung meldet, daß der Drucker noch beschäftigt ist (Busy-Leitung) und somit keine neuen Daten aufnehmen kann. Die andere (Strobe-Leitung) weist den Drucker an, gültige Daten vom Ausgang des Computers zu übernehmen.

Die Abb. 10.24 zeigt die wichtigsten Signale und ihre Richtung.

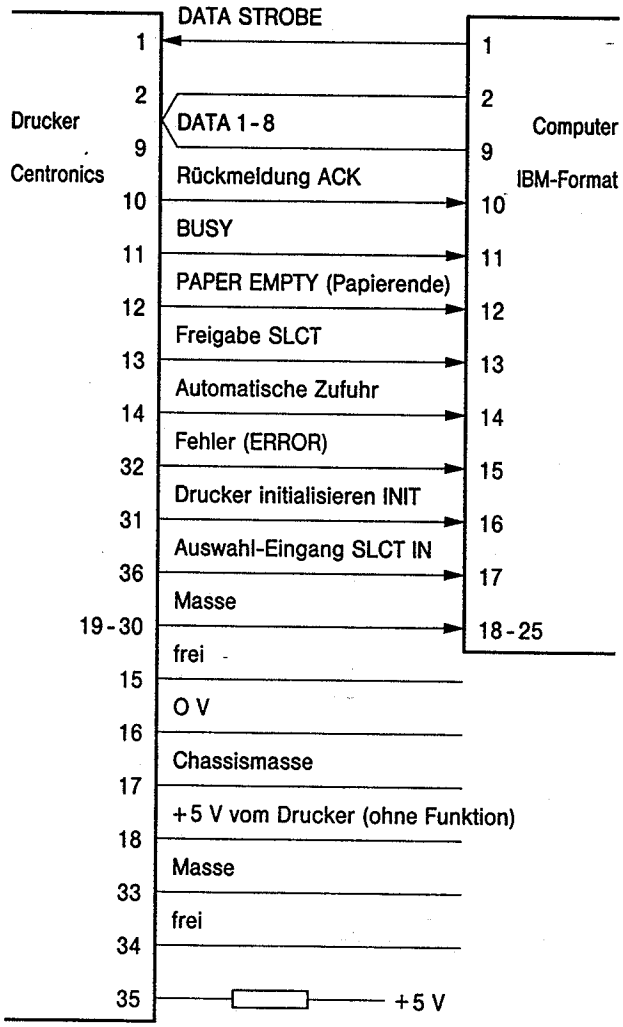
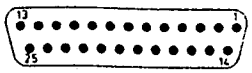


Abb. 10.24 – Steckverbindung Centronic auf IBM

Die Abb. 10.25 zeigt beispielhaft die Steckerbelegung eines 25poligen Steckers.

Druckeranschluß 25polige Buchse



1 STROBE	7 DATA 5	13 SLCT
2 DATA 0	8 DATA 6	14 AUTO FEED
3 DATA 1	9 DATA 7	15 ERROR
4 DATA 2	10 ACK	16 INIT
5 DATA 3	11 BUSY	17 SLCT IN
6 DATA 4	12 PE	18 - 25 GND

Abb. 10.25 – Anschlußbelegung nach IBM-Format

10.8 Grundsätzliche Anwendungsmöglichkeiten für Mikrocomputer

Aus Mikroprozessoren kann man nur eins machen, nämlich einen Mikrocomputer, und mit einem Mikrocomputer kann man ziemlich viel machen. Die Anwendungen der Mikrocomputer liegen auf zwei Gebieten:

a) Steuerungen und
b) frei programmierbare Rechner.

10.8.1 Steuerungen

Ein elektronischer Würfel, der auf Tastendruck startet und nach Loslassen der Taste eine Augenzahl anzeigt, ist leicht aus einem Zähler, einem Taktgenerator und einigen Verknüpfungsschaltungen aufzubauen. Aus den vorhergehenden Abschnitten ergibt sich, daß diese Würfelsteuerung ebenso einfach durch einen entsprechend programmierten Mikrocomputer zu realisieren ist. Obwohl aus Kostengründen eine so einfache Steuerung wohl immer ohne Mikroprozessor aufgebaut werden wird, so zeigt dieses Beispiel doch, wie ein Mikrocomputer eine reine Hardwaresteuerung ersetzen kann. Der Einsatz von Mikrocomputern für Steuerzwecke bietet sich immer dann an, wenn

a) die Steuerung sehr komplex ist oder
b) arithmetische Operationen erforderlich sind oder
c) häufig wechselnde Eingangsdaten zu verarbeiten sind oder
d) die Steuerung für verschiedene Zwecke verwendet werden soll.

Anwendungsbereiche und Beispiele sind

Konsumelektronik:	Registerkassen, Münzwechsler, Spielautomaten ...
Medizin:	Diagnosegeräte, Testsysteme, Laborauswertungen ...
Meßtechnik:	Oszilloskope, Logikanalysatoren, automatische Meßbereichsumschaltung ...
Verkehr:	Ampelsteuerungen, Bordcomputer in Fahrzeugen aller Art ...
Industrie:	Prozeßkontrolle, Aufzugssteuerung, Werkzeugmaschinensteuerungen ...
Datenverarbeitung:	Steuerung von Datensichtstationen, Floppy-Disk-Laufwerken, Bandgeräten, Druckern, Datenkonzentratoren, E/A-Steuerungen ...
Fernmeldetechnik:	Steuerung von Vermittlungssätzen, Prüfgeräten, Nebenstellenanlagen ...

10.8.2 Frei programmierbare Rechner

Besonders mit den neuen 32-Bit-Mikroprozessoren lassen sich sehr leistungsfähige und trotzdem preiswerte Mikrocomputer bauen, die als frei programmierbare Rechner die Leistung von Minicomputern erreicht haben. Sie haben als Personalcomputer bereits eine große Verbreitung gefunden. Durch umfangreiches Hard- und Softwarezubehör wie Massenspeicher, preiswerte E/A-Geräte, Interpreter und Compiler für höhere Programmiersprachen usw. bieten die Personalcomputer einen sehr hohen Komfort.

10.9 Auswahlkriterien für Mikroprozessoren

Wenn die Entscheidung gefallen ist, zur Lösung eines Problems einen Mikrocomputer einzusetzen, entsteht als nächstes Problem die Frage, welcher der vielen Typen verwendet werden soll. Die Auswahl richtet sich hauptsächlich nach folgenden Gesichtspunkten:

Datenformat

Die meisten Mikroprozessoren haben eine Wortlänge von 8, 16 oder 32 Bit. Für Steuerungen mit rein numerischen Problemen genügt ein 8-Bit-Prozessor. Auch bei der Verarbeitung von Analog/Digital-Umwandlungen und alphanumerischen Größen ist ein 8-Bit-Mikroprozessor ausreichend. Soll der Mikrocomputer in der allgemeinen Datenverarbeitung Verwendung finden, sind die 16- oder 32-Bit-Mikroprozessoren am geeignetsten.

Befehlsvorrat und Adressierungsmöglichkeiten

Ein vielseitiger und umfangreicher Befehlsvorrat und ein darauf abgestimmter Satz von Adressierungsmöglichkeiten verkürzen die Entwicklungszeit, weil Programme schneller erstellt werden können, und eventuell die Verarbeitungszeit, weil das gleiche Problem mit weniger Befehlen verarbeitet werden kann.

CPU-Architektur

Hierunter sind alle Hardware-Merkmale des Mikroprozessors zu verstehen, wie Wortlänge, Art und Anzahl der internen Register, Organisation der Busse usw. Je besser die CPU zu dem zu lösenden Problem paßt, je vielseitiger sie ist, desto kürzer werden wieder Entwicklungs- und Verarbeitungszeit.

Verarbeitungsgeschwindigkeit

Zum einen hängt die Geschwindigkeit eines Mikrocomputers von der Zeit ab, die für die Ausführung eines Befehls benötigt wird. Diese sog. Befehlszykluszeit hängt wiederum ab von der Taktfrequenz und dem Aufbau der Ablaufsteuerung. Zum anderen hängt die Geschwindigkeit eines Mikrocomputers auch ab von dem Befehlsvorrat, den Adressierungsmöglichkeiten und der CPU-Struktur. Weil die Verarbeitungsgeschwindigkeit von so vielen verschiedenen Größen abhängt, ist es schwierig, ein Maß für sie zu finden. Man hat zu diesem Zweck Bewertungsprogramme (benchmark programs) geschrieben, mit denen die einzelnen Mikrocomputer verglichen werden. Mit ihnen wird festgestellt, wie lange ein Computer mit der Erledigung einer bestimmten Aufgabe beschäftigt ist. Die mit derartigen Programmen ermittelten Ergebnisse sind nicht für alle Anwendungen eines Mikrocomputers gültig.

Bei den modernen Prozessoren lassen sich die Ausführungszeiten der Befehle eigentlich nicht exakt vorausbestimmen. Dieses Problem entsteht dadurch, daß ständig Prozesse parallel ablaufen. Der ursprüngliche Von-Neumann-Zyklus wird nicht mehr eingehalten (1. Befehl holen, vollständig abarbeiten → 2. Befehl holen, vollständig abarbeiten usw.). Jetzt werden die durch Prefetch (mehrere Befehle werden im voraus in den Prozessor geholt und dort in Puffern gespeichert) geholten Befehle (immer je 32 Bit) in parallel arbeitende Pipelines eingegeben. Sie „fließen" durch die Stufen, bis sie in der entsprechenden ALU angekommen sind. Somit beinhalten die Prozessoren auch mehrere Rechenwerke. Auf diese Weise wird die Befehlsausführungszeit erheblich verbessert.

Eine weitere Beschleunigung der Verarbeitung wird dadurch erreicht, daß die Prozessoren ein eingebautes Cache erhalten.

Der Begriff Cache bedeutet soviel wie „Verstecken". Der technische Einfluß dieses Versteckens soll hier in kurzer Form dargestellt werden.

Ein Cache ist ein sehr schneller Zwischenspeicher (Pufferspeicher) für Befehle und/oder Daten. Er soll den Dialog zwischen Prozessor und Speicher beschleunigen. Unterschieden wird zwischen internen und externen Caches. Vor der Einführung des integrierten Caches wurden bei Kleincomputern nur externe eingesetzt.

Im Cache werden die zuletzt verwendeten Befehle oder Daten mit den zugehörigen Speicheradressen abgelegt. Die daraus resultierenden Vorteile unterscheiden sich geringfügig bei externen und On-Chip-Caches.

Bei den externen ermöglicht der Pufferspeicher eine bessere Anpassung der Geschwindigkeiten von Prozessor und Speicherbausteinen. Werden als Hauptspeicher preiswerte, langsame DRAM eingesetzt, muß der Prozessor immer wieder auf den Speicher warten. Verwendet man sehr schnelle SRAM, muß man relativ viel Geld ausgeben. Hier bildet ein Cache aus einem Hochgeschwindigkeits-RAM mit relativ kleiner Speicherkapazität einen guten Kompromiß. Dieser wird parallel zum Hauptspeicher an den Systembus geschaltet. Beim Holen von Befehlen werden diese mit ihren **Adressen** in den Cache kopiert.

Dort bleiben sie auch nach der Abarbeitung gespeichert. Beim nächsten Holen prüft das System, ob die aktuelle **Adresse** (und damit auch der Befehl) im Cache vorhanden ist. Um hierbei eine hohe Geschwindigkeit zu erreichen, kann der Vorgang nicht durch Software gesteuert werden. Man verwendet daher einen oder mehrere Komparatoren (Hardware-Vergleicher), die bei einem Cachezugriff alle aktiviert werden. So wird die aktuelle **Adresse** parallel mit allen im Chip gespeicherten verglichen. Erzielt das System einen Treffer (hit), hat es den Befehl gefunden. Dieser wird daraufhin aus dem schnellen Pufferspeicher geholt und nicht aus dem langsamen Hauptspeicher. War ein Vergleichsvorgang erfolglos, spricht man von einem Fehlgriff (miss). Daraufhin wird der Holvorgang mit dem Programmspeicher fortgesetzt. Dabei wird der neue Befehl sofort in den Cache geschrieben. Wird dieser in einem der nächsten Schritte noch einmal benötigt, landet das System einen Treffer und arbeitet den Befehl schneller ab, als beim ersten Mal. Die Umschaltung zwischen Cache und Hauptspeicher wird durch die Cachesteuerung (bei einigen Prozessoren im Chip enthalten, sonst extern vorhanden) automatisch vorgenommen. In Zählschleifen, die keinen umfangreicheren Platz als die Cachekapazität benötigen, ist die Trefferquote selbstverständlich am größten. Ein Cache-Einsatz ist nach diesen Erkenntnissen nur sinnvoll, wenn die Geschwindigkeiten von µP und RAM nicht übereinstimmen. Der MC68020 besaß als erster einen 256 Byte großen Cache-Speicher im Chip. Immer wenn sich der Prozessor von außen einen Befehl (Befehlscode und Operanden) holt, wird dieser parallel zum Pipelining mit seiner Originaladresse im Cache abgelegt. Beim nächsten Befehlszyklus prüft der µP (genau wie beim externen Cache), ob die Befehlsadresse schon intern gespeichert ist. Bei einem Treffer holt er sich den Befehl aus dem Cache. Er muß dafür keinerlei Busaktivitäten entwickeln. Besonders wirkungsvoll ist dieser Ablauf bei Schleifen, die nur mit internen Registern arbeiten, weil dann vollständige Programmteile im Pufferspeicher stehen können. Durch den On-Chip-Cache wird der Bus erheblich entlastet. Die Prozeßbeschleunigung ist bei Zeitschleifen natürlich sinnlos, da diese den Programmablauf ja gerade verzögern sollen. Bei Abfrageschleifen (z. B. Abfrage der Sensoren in schnellen Steuerungsschaltungen) hingegegen entsteht ein großer Vorteil.

Bei modernen Rechnern wird der Datenbus in einen Daten- und einen Programmbus aufgeteilt. Damit können Daten und Befehle parallel verarbeitet werden. Das hat zur Folge, daß auch jedem Bussystem ein eigener Cache zugeordnet wurde. Der MC 68040 besaß als erster Prozessor ein 4-KByte-Daten-Cache und ein 4-KByte-Befehls-Cache.

Speicherkonzepte

Neben dem adressierbaren Speicherbereich ist vor allem wichtig, ob es möglich ist, auch schnelle Speicher-IC anzuschalten.

Weitere Kriterien sind zusätzliche Leistungsmerkmale wie die Programm-Unterbrechungen und direkter Speicherzugriff, Schnittstelleneigenschaften (TTL-Kompatibilität, Belastbarkeit), Anzahl der benötigten Versorgungsspannungen, Verfügbarkeit von Bausteinen, Unterstützung für Systementwicklung und Dokumentation über Bausteine, Software und Applikationen.

Multitasking

Alle 68000er, der 80386 und der 80486 sind Mikroprozessoren für Multitasking und Multiuser-Betrieb. Multitasking bzw. Multiprogramming bedeutet, daß mehrere Programme oder Programmteile scheinbar gleichzeitig von einem Prozessor in einem Computer abgearbeitet werden. Multiuser-Betrieb bedeutet, daß mehrere Terminals (Tastatur + Bildschirm) an einen Computer angeschlossen sind. Dabei können mehrere Benutzer (User) scheinbar gleichzeitig an einem Computer arbeiten.

Scheinbar bedeutet, daß natürlich in Wirklichkeit immer nur ein Task oder User zur Zeit arbeitet. Der Prozessor schaltet aber in gewissen zeitlichen Abständen zwischen den Anwendern (Anwendungen) hin und her und täuscht damit ein gleichzeitiges Arbeiten vor.

Unter Task versteht man ein lauffähiges Programm (oder Programmteil). Es benötigt immer noch ein übergeordnetes Programm, das die Tasks in der richtigen Reihenfolge verwaltet.

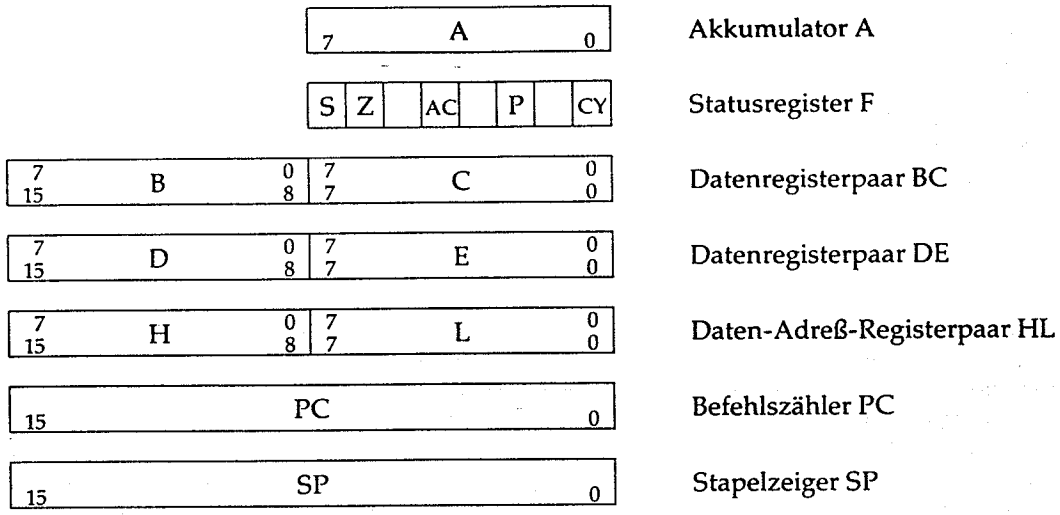


Abb. A36 – Prozessortyp 8080/8085

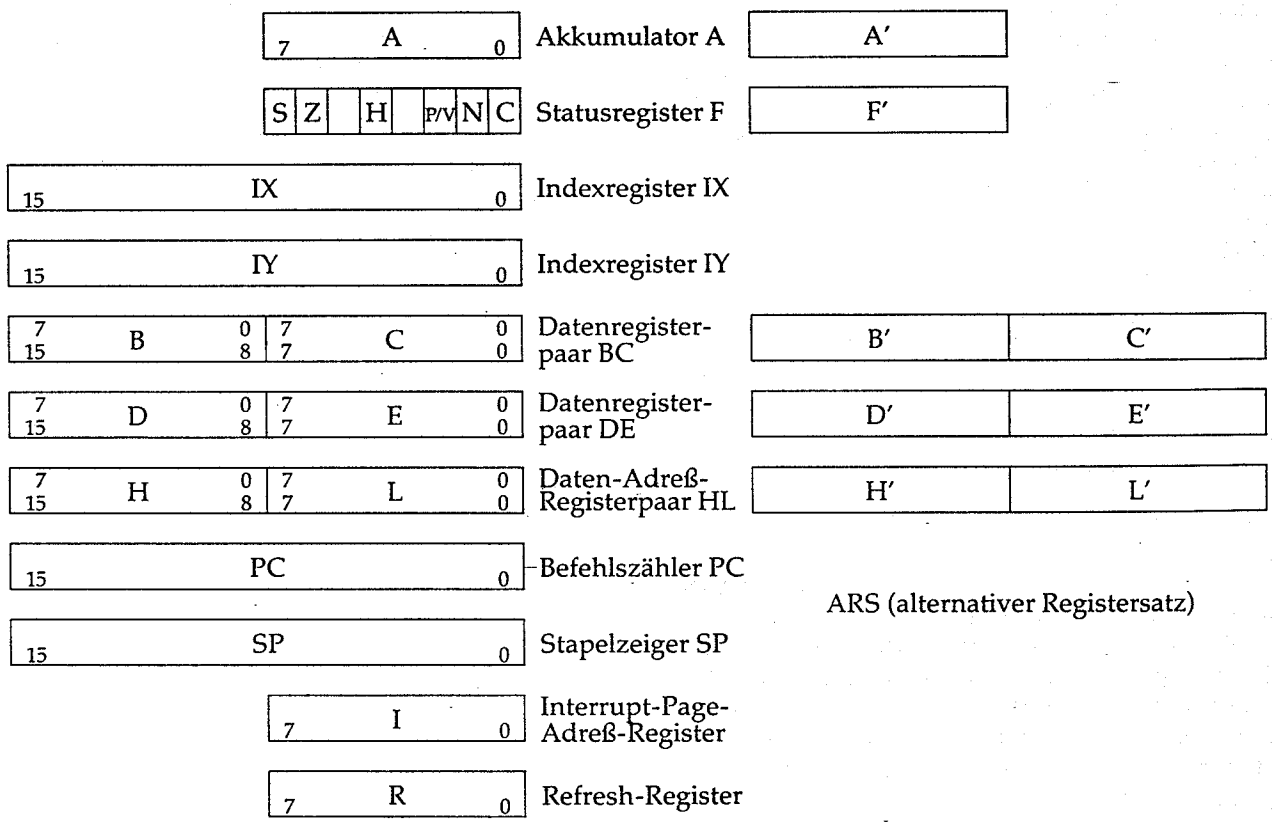


Abb. A37 – Prozessortyp Z80

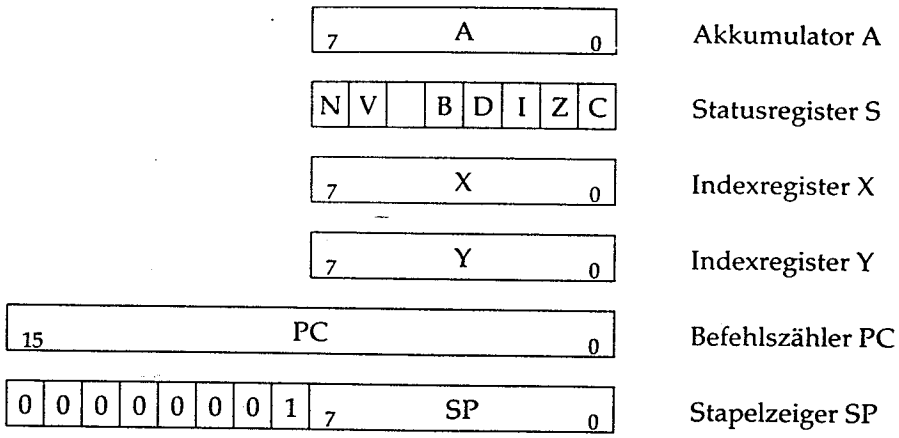


Abb. A38 – Prozessortyp 6502

Bezeichnung	6502	8080/ 8085	Z80	Erläuterung
Übertrags-(Carry-)Flag	C	CY	C	C = 1, wenn bei arithmetischen Operationen ein Übertrag entsteht.
Null-(Zero-)Flag	Z	Z	Z	Z = 1, wenn das Ergebnis einer Operation 0 ist.
Vorzeichen-(Sign-, Negativ-) Flag	N	S	S	S = 1, (N = 1) wenn das Ergebnis einer Operation negativ ist.
Paritätsflag	–	P	P	P = 1, wenn die Anzahl der Bit im Zustand 1 gradzahlig ist.
Hilfsübertragsflag	–	AC	H	H = 1, (AC = 1) wenn bei arithmetischen Operationen im unteren Halbbyte ein Übertrag entsteht.
Dezimalflag	D	–	–	D = 1, wenn der Prozessor arithmetische Operationen im Dezimalsystem durchführt.
Überlauf-(Overflow-)Flag	V	–	V	V = 1, wenn das Ergebnis außerhalb des zulässigen Zahlenbereichs liegt.
Subtraktionsflag	–	–	N	N = 1, wenn der Prozessor eine Subtraktion ausführt.
Interruptflag	I	–	–	I = 1, verhindert eine Interruptauslösung über IRQ.
Breakflag	B	–	–	B = 1, wenn durch den BREAK-Befehl softwaremäßig ein Interrupt ausgelöst wurde.

Tabelle A15 – Bedeutung der Flags

SIEMENS

High-Performance
8-Bit CMOS Single-Chip Microcontroller

SAB 80C515/80C535

Preliminary

SAB 80C515/80C515-16 CMOS microcontroller with factory mask-programmable ROM
SAB 80C535/80C535-16 CMOS microcontroller for external ROM

- 8 K × 8 ROM (SAB 80C515 only)
- 256 × 8 RAM
- Six 8-bit I/O ports, one input port for digital or analog input
- Three 16-bit timer/counters
- Highly flexible reload, capture, compare capabilities
- Full-duplex serial channel
- Twelve interrupt vectors, four priority levels
- 8-bit A/D converter with 8 multiplexed inputs and programmable internal reference voltages
- 16-bit watchdog timer
- Boolean processor
- Most instructions execute in 1 µs (750 ns)
- 4 µs (3 µs) multiply and divide
- External memory expandable up to 128 Kbytes
- Backwardly compatible with SAB 8051
- Functionally compatible with SAB 80515
- Idle and power-down mode
- Plastic leaded chip carrier package: P-LCC-68
- Plastic Metric Quad Flat Package P-MQFP-80
- Two temperature ranges available:
  0 to 70 °C (for 12, 16, 20 MHz)
  – 40 to 85 °C (for 12, 16 MHz)

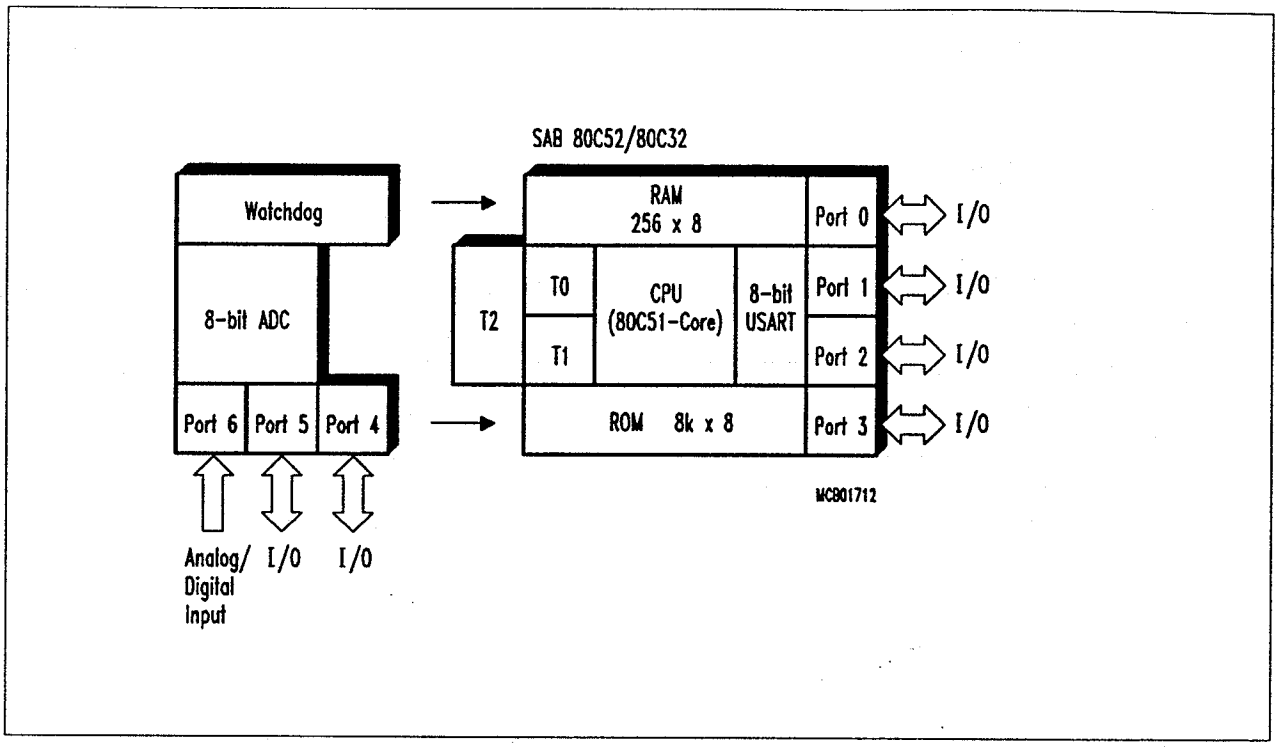
The SAB 80C515/80C535 is a powerful member of the Siemens SAB 8051 family of 8-bit microcontrollers. It is designed in Siemens ACMOS technology and is functionally compatible with the SAB 80515/80535 devices designed in MYMOS technology.

The SAB 80C515/80C535 is a stand-alone, high-performance single-chip microcontroller based on the SAB 8051/80C51 architecture. While maintaining all the SAB 80C51 operating characteristics, the SAB 80C515/80C535 incorporates several enhancements which significantly increase design flexibility and overall system performance.

In addition, the low-power properties of Siemens ACMOS technology allow applications where power consumption and dissipation are critical. Furthermore, the SAB 80C515/80C535 has two software-selectable modes of reduced activity for further power reduction: idle and power-down mode.

The SAB 80C535 is identical with the SAB 80C515 except that it lacks the on-chip program memory. The SAB 80C515/80C535 is supplied in a 68-pin plastic leaded chip carrier package (P-LCC-68) or in a plastic metric quad flat package (P-MQFP-80).

There are versions for 12, 16 and 20 MHz operation and for 16 MHz operation and for extended temperature ranges – 40 to 85 °C. Versions for extended temperature range – 40 to + 110 °C are available on request.


SAB 80C52/80C32
Watchdog
8-bit ADC
Port 6
Port 5
Port 4
Analog/
Digital
Input
I/O
I/O
RAM
256 x 8
T2
T0
T1
CPU
(80C51-Core)
8-bit
USART
ROM 8k x 8
Port 0
Port 1
Port 2
Port 3
I/O
I/O
I/O
I/O
MC801712

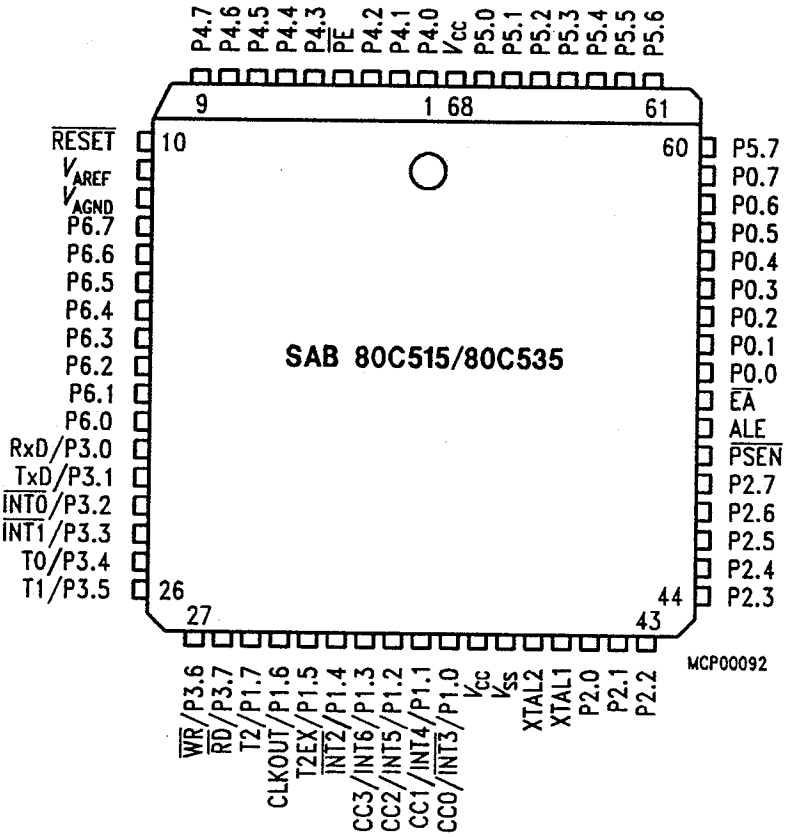
Ordering Information

Type	Ordering Code	Package	Description 8-Bit CMOS Microcontroller
SAB 80C515-N	Q67120-DXXXX	P-LCC-68	with mask-programmable ROM, 12 MHz
SAB 80C535-N	Q67120-C0508	P-LCC-68	for external memory, 12 MHz
SAB 80C515-N-T40/85	Q67120-DXXXX	P-LCC-68	with mask-programmable ROM, 12 MHz ext. temperature – 40 to + 85 °C
SAB 80C535-N-T40/85	Q67120-C0510	P-LCC-68	for external memory, 12 MHz ext. temperature – 40 to + 85 °C
SAB 80C515-16-N	Q67120-DXXXX	P-LCC-68	with mask-programmable ROM, 16 MHz
SAB 80C535-16-N	Q67120-C0509	P-LCC-68	for external memory, 16 MHz
SAB 80C535-16-N-T40/85	Q67120-C0562	P-LCC-68	for external memory, 16 MHz ext. temperature – 40 to + 85 °C
SAB 80C535-20-N	Q67120-C0778	P-LCC-68	for external memory, 20 MHz
SAB 80C535-M	Q67120-C0857	P-MQFP-80	for external memory, 12 MHz
SAB 80C515-M	Q67120-DXXXX	P-MQFP-80	with mask-programmable ROM, 12 MHz
SAB 80C535-M-T40/85	Q67120-C0937	P-MQFP-80	for external memory, 12 MHz ext. temperature – 40 to + 85 °C
SAB 80C515-M-T40/85	Q67120-DXXXX	P-MQFP-80	with mask-programmable ROM, 12 MHz ext. temperature – 40 to + 85 °C

Notes: Versions for extended temperature range – 40 to + 110 °C on request.
The ordering number of ROM types (DXXXX extension) is defined after program release (verification) of the customer.

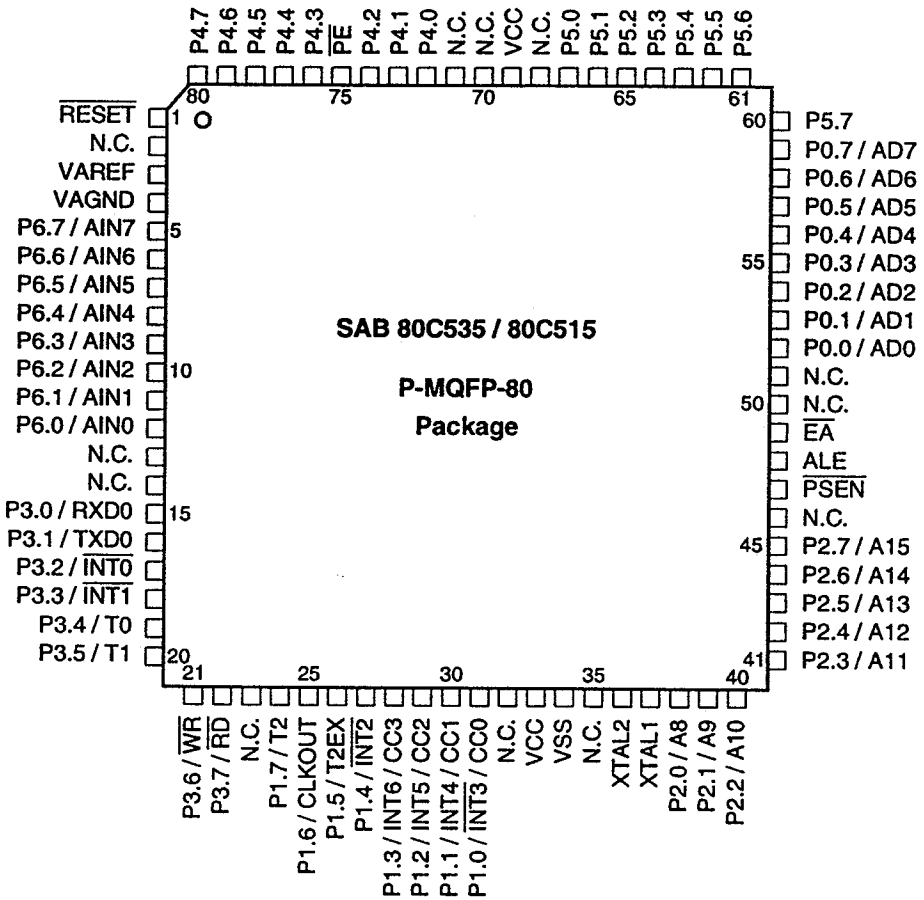
Pin Configuration
(top view)

P-LCC-68

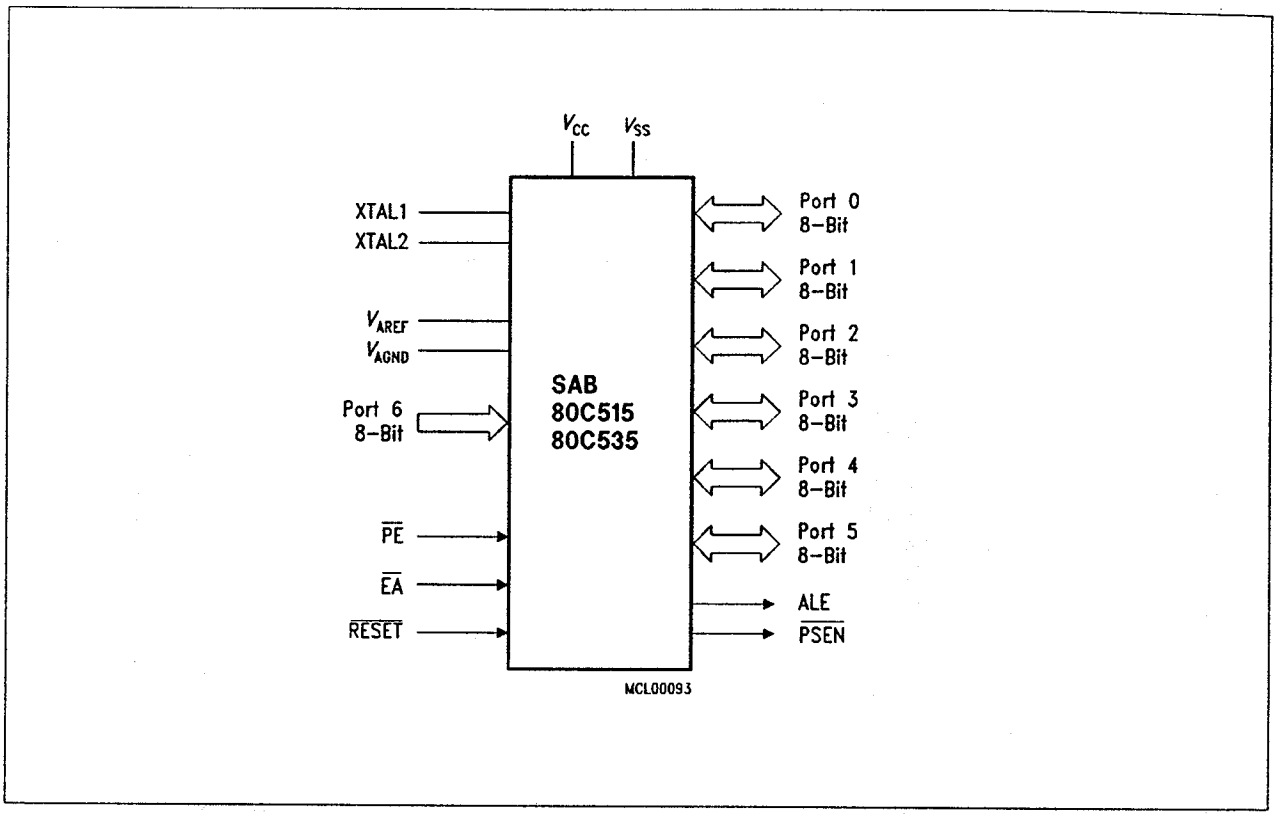


Pin Configuration
(top view)

P-MQFP-80



N.C. pins must not be connected.



Logic Symbol