Klaus Wüst
Mikroprozessortechnik
Vieweg Verlag 2006, 2. Auflage

Dieses Ergänzungsdokument zum Buch enthält die Abschnitte der ersten Auflage, die in der zweiten Auflage nicht mehr enthalten sind.

Gießen, September 2006

Inhaltsverzeichnis

1 Halbleiterbauelemente **3**
1.1 Die Herstellung integrierter Schaltkreise . . . 3
1.1.1 Grundideen, Begriffe, Basistechnologien . . . 3
1.1.2 Die Herstellung von Siliziumscheiben . . . 3
1.1.3 Oxidmasken, Lithografie . . . 5
1.1.4 Dotierung . . . 6
1.1.5 Depositionsverfahren, Epitaxie, Metallisierung . . . 7
1.1.6 Funktionstest und Montage der Chips . . . 8
1.2 Aufgaben und Testfragen . . . 10

2 Einfache Mikroprozessoren **12**
2.1 Fallbeispiel: Motorola 6800 . . . 12
2.1.1 Übersicht . . . 12
2.1.2 Anschlussleitungen . . . 13
2.1.3 Interner Aufbau . . . 14
2.1.4 Befehlssatz . . . 15
2.1.5 Programmierung . . . 16
2.2 Fallbeispiel: Intel 8086 . . . 17
2.2.1 Adressbildung und Speichersegmentierung . . . 18
2.2.2 Interruptkonzept und Betriebssystem-Anbindung . . . 20
2.2.3 Systemstart . . . 22
2.2.4 Adresspufferung, Buscontroller, Prefetch Queue, Wortausrichtung . . . 22
2.2.5 Der Intel 8088 . . . 24
2.3 Aufgaben und Testfragen . . . 25

Lösungen zu den Aufgaben und Testfragen **26**

Literaturverzeichnis **27**

1 Halbleiterbauelemente

1.1 Die Herstellung integrierter Schaltkreise

1.1.1 Grundideen, Begriffe, Basistechnologien

Ein integrierter Schaltkreis (IC) ist ein Baustein, in dem viele Bauteile auf einem einzigen Trägerplättchen zusammengefasst sind. Dies kann man erreichen, indem man die Leiterbahnen durch Aufdampfen von Silber oder Gold in kleinsten Abmessungen auf Keramikträger aufbringt. Darauf können dann weitere Bauelemente gelötet werden (Dünnschichttechnik) .

Eine viel stärkere Integration erreicht man aber mit der *Monolithtechnik* (Volumentechnik). Dabei wird die ganze Schaltung in einem einzigen Stück Halbleiterkristall hergestellt. Dies geschieht durch Herstellung entsprechend vieler p- und/oder n-leitender Bereiche in diesem Kristall, der damit zum integrierten Schaltkreis wird. Eine Kombination der Dünnschichttechnik mit der Monolithtechnik ist die *Hybridtechnik*.

Die größte Bedeutung hat heute die Monolithtechnik, auf die wir nun näher eingehen wollen. Der Herstellungsvorgang soll dabei nicht nur zu einer maximalen Integration führen, sondern möglichst auch die Fertigung aller denkbaren Schaltungen in einem einheitlichen Prozess ermöglichen. So muss es z.B. möglich sein mit den gleichen Arbeitsschritten neben Dioden und Transistoren auch Widerstände und Kondensatoren in dem Halbleiterkristall zu erzeugen. Die Basistechnologien dabei sind:

- Die Reinstdarstellung von Silizium,
- die Diffusion (Wanderung von Teilchen) und Epitaxie (Schichtenwachstum),
- die Ionenimplantation,
- die Lithografie,
- die Metallisierung,
- die Herstellung von Dünnschichtoxiden.

1.1.2 Die Herstellung von Siliziumscheiben

Silizium ist auf der Erde in großen Mengen als Quarzsand, Siliziumdioxid, vorhanden. Daraus kann durch Reduktion mit Kohlenstoff Rohsilizium gewonnen werden. Aus dem Rohsilizium muss nun reines, einkristallines Silizium werden. Im Folgenden sind einige typische und weit

verbreitete Verfahren geschildert. Dazu stellt man aus dem Rohsilizium zunächst Trichlorsilan, $SiHCl_3$ her. Das Trichlorsilan leitet man in ein Vakuumgefäß ein, wo es bei Anwesenheit von viel Wasserstoff zu elementarem Silizium und Salzsäure zerfällt

$$SiHCl_3 + H_2 \rightarrow Si + 3HCl$$

Das elementare Silizium schlägt sich an einem geheiztem Siliziumstab, der sog. *Siliziumseele*, bei ca. 1100 ^{o}C ab (Abb. 1.1). Dieser Stab wächst dabei bis zu einer gewünschten Dicke an und

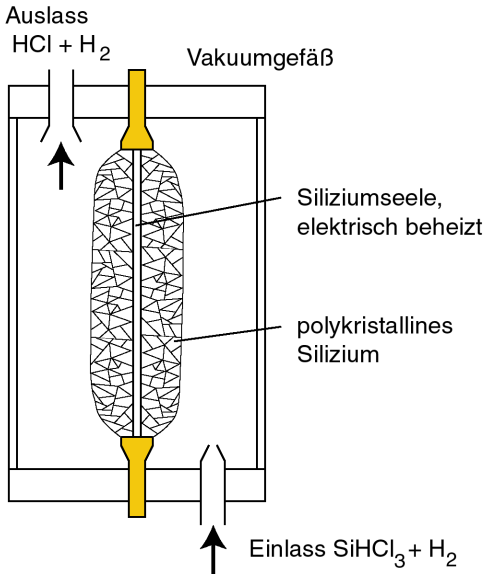

Abbildung 1.1: Die Herstellung des polykristallinen Siliziumstabes.

zwar mit einer Gesamtreinheit von ca. $10^{-9} \ldots 10^{-10}$. Ein weitere Verbesserung der Reinheit erhält man mit der zonenweisen Aufheizung durch ein hochfrequentes elektromagnetisches Feld, der sog. *Zonenreinigung*, die ähnlich wie das unten beschriebene Zonenziehen verläuft. Der Stab ist nun zwar sehr rein, aber immer noch polykristallin, d.h. seine Gitterordnung ist auf kleine Bereiche beschränkt. Ein Einkristall kann z.B. aus der Schmelze gezogen werden (Czochralski-Verfahren), wobei sich aber immer auch Verunreinigungen aus den Wänden des Tiegels lösen und in den Kristall mischen. Hochreine Einkristalle erhält man durch *tiegelfreies Zonenziehen*. Dabei wird der polykristalline Siliziumstab in einem Vakuumgefäß an beiden Enden eingespannt (Abb. 1.2). An dem einem Ende berührt er fast einen vorhandenen kleinen, reinen Silizium-Einkristall, dem sog. Impfling. Eine Hochfrequenz-Heizspule (Induktionsspule) erhitzt nun eine kleine Zone des polykristallinen Stabs an diesem Ende. Der polykristalline Stab schmilzt dort, wölbt sich auf und berührt den Impfling. Die Heizspule bewegt sich nun langsam zum anderen Ende. Beim Erstarren an der Zonengrenze nimmt das Silizium die Kristallordnung des Impflings an. Um Inhomogenitäten zu vermeiden, rotiert der ganze Stab langsam um die Längsachse. Wenn der Kristall eine durchgehende Dotierung erhalten soll, kann während des Zonenziehens ein gasförmiger Dotierungsstoff zugegeben werden, der sich in die Schmelze mischt und gleichmäßig verteilt. Wenn die Heizspule am anderen Ende angekommen ist, ist der ganze Stab einkristallin. Außerdem werden dabei Verunreinigungen zum Stabende transportiert und die Reinheit des Kristallstabes verbessert sich weiter.

Der so hergestellte Si-Einkristallstab wird mit einer Innenloch-Diamantsäge unter Beachtung der Kristallorientierung zersägt. Die rauhe Oberfläche wird durch Läppen, Ätzen und Polieren

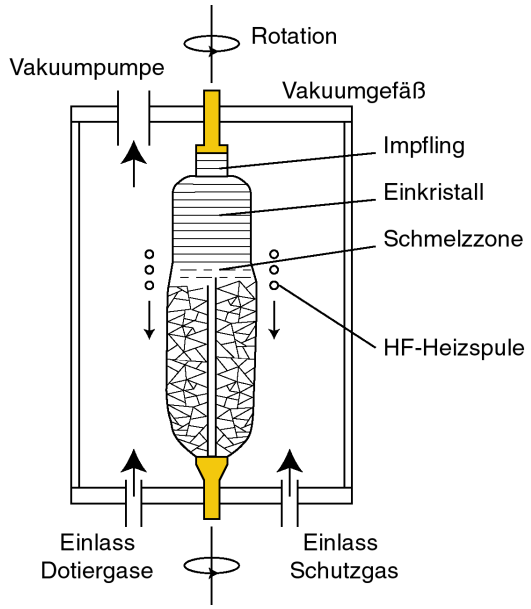

Abbildung 1.2: Tiegelfreies Zonenziehen.

geglättet und man erhält die *Wafer*, dünne, glatte und saubere Scheiben aus einkristallinem Silizium. Die Wafer haben Durchmesser zwischen 10 cm und 30 cm und eine Dicke von weniger als einem mm; in jedem Wafer werden später viele integrierte Schaltkreise erzeugt.

1.1.3 Oxidmasken, Lithografie

Zur Erzeugung der p- und n-leitenden Bereiche müssen Teile der Oberfläche abgedeckt werden. Die frei bleibenden Bereiche sind durchlässig für die Dotierungsstoffe, die abgedeckten nicht. So können ganz gezielt bestimmte Bereiche dotiert werden. Vor jedem Dotierungsschritt muss eine neue Abdeckungsmaske aufgebracht werden. Wegen der hohen Integrationsdichte braucht man dabei kleinste Strukturen. Die Abdeckung geschieht mit *Siliziumdioxid* SiO_2, ein amorphes glasartiges Material, das undurchläßig für Dotierungsstoffe und hitzebeständig bis 1000 °C ist. Die Oxidation kann thermisch geschehen, dazu lässt man Sauerstoff über das ungefähr 1000 °C heiße Silizium strömen. Die Bildung des Siliziumdioxids lässt sich sehr genau steuern, was für die Herstellung des Gateoxids bei Feldeffekttransistoren und des Tunneloxids bei EEPROM-Speichern wichtig ist.

Für die Erzeugung der Öffnungen in der Oxidschicht wird eine Maske aus Lack benutzt, die mit Hilfe der *Fotolithografie*, erstellt wird, wie in Abb. 1.3 gezeigt ist.[1] Nachdem der Siliziumkristall (1) mit Siliziumdioxid überzogen ist (2) wird *Fotolack* auf das Oxid aufgetragen (3). Man gibt den Fotolack in die Mitte des rotierenden Wafers und die Zentrifugalkraft verteilt ihn nach außen. Ein Positiv-Fotolack löst sich bei der Entwicklung an den belichteten Stellen auf, ein Negativ-Fotolack an den unbelichteten. Fotolacke haben eine gewisse Beständigkeit gegenüber ätzenden Lösungsmitteln und heißen deshalb auch *Fotoresist*. Nun wird die Strukturmaske auf den Fotolack optisch abgebildet (4). Das bevorzugte Verfahren dazu ist heute das Step-and-Repeat-Verfahren, bei dem die hochwertige Projektionsoptik eines sog. Wafer-

[1] ursprüngliche Bedeutung von Lithografie: Steindruck.

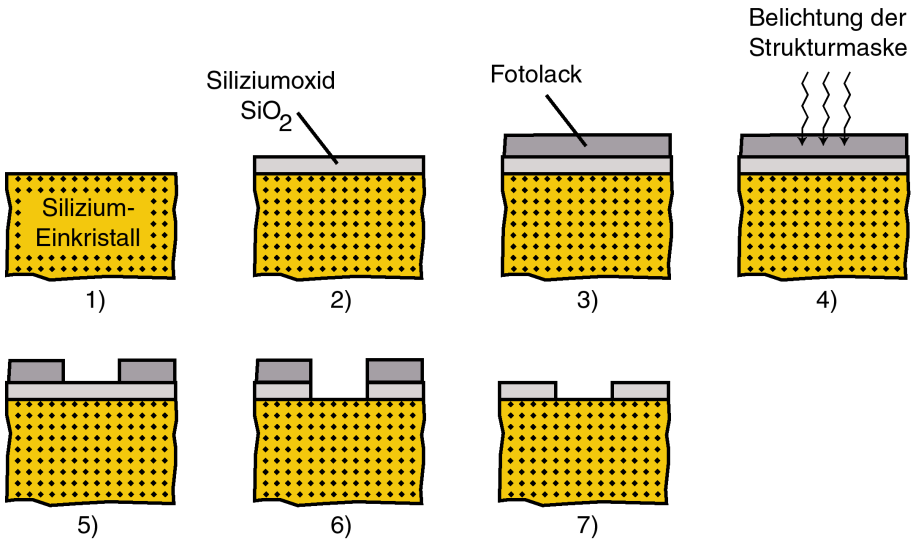

Abbildung 1.3: Fotolithografische Erzeugung einer Siliziumdioxid-Maske mit Öffnungen.

steppers in jedem Schritt die Maskenstruktur *eines* Chips auf den Fotolack abbildet.[2] Danach rückt der Waferstepper um eine Position weiter und belichtet die Struktur des nächsten Chips, der auf dem Wafer entstehen soll. Wegen der heutigen kleinen Strukturbreiten kann dabei kein sichtbares Licht mehr benutzt werden, sondern es wird mit kurzwelligem Licht im tiefen Ultraviolett gearbeitet (Deep UV, DUV).[3] Der Lack wird anschließend entwickelt, der belichtete Lack herausgespült, der verbleibende Lack evtl. gehärtet. Die Lackmaske ist nun fertig (5). Durch die Löcher in der Lackmaske wird nun das darunterliegende Siliziumdioxid weggeätzt, so dass dort der Siliziumkristall frei liegt (6). Das Ätzen kann z.B. mit Flußsäure erfolgen. Mit einem Lösungsmittel wird nun der restliche Fotolack entfernt, die Oxidmaske ist fertig und hat die gewünschten Fenster, durch die Dotierungsstoffe in den Kristall eindringen können (7).

1.1.4 Dotierung

Um dotierte Bereiche in dem Halbleiterkristall zu erzeugen, müssen die Dotierungsstoffe von der Oberfläche bis in eine gewisse Tiefe des Siliziumkristalls gelangen. Eine einfache Methode dafür ist die *Diffusion*. Diffusion ist die selbstständige Ausgleichsbewegung von Teilchen zu Gebieten geringerer Konzentration hin. Die Diffusionsbewegung entsteht letztlich aus der thermischen Eigenbewegung der Teilchen und führt dazu, dass Dotierungsatome von der Oberfläche aus in den Kristall eindringen und sich dort allmählich ausbreiten. Der Diffusionsstrom wird beschrieben durch

$$\vec{j} = -D\nabla n$$

[2] Andere Verfahren sind die Kontaktbelichtung und die Abstandsbelichtung.

[3] Wegen der Wellennatur des Lichtes, kann ein optisches Gerät keine Struktur abbilden, die kleiner ist als die Wellenlänge des benutzten Lichtes. Bei weiterer Verkleinerung der Strukturen muss mit der Elektronenstrahl-Lithografie gearbeitet werden.

wobei D die *Diffusionskonstante* ist, n die Teilchenkonzentration und ∇ der Gradientenoperator, der die Richtung der stärksten Zunahme ergibt. Die örtliche Veränderung der Teilchendichte ist

$$\frac{\partial n}{\partial t} = D\nabla^2 n$$

Die Diffusionskonstante steigt mit der Temperatur an. Dotierungsstoffe breiten sich also in einem heißen Kristall viel stärker aus, als in einem kalten, so dass sich die Diffusion gut durch die Temperatur steuern lässt. Man muss dafür sorgen, dass eindiffundierte Dotierungsstoffe sich nicht ungewollt weiter ausbreiten. Daher wählt man folgende Reihenfolge: Der Dotierungsstoff mit dem kleinsten Diffusionskoeffizienten wird zuerst – bei hoher Temperatur – eindiffundiert. Danach folgt der mit dem zweithöchsten Diffusionskoeffizienten bei etwas niedrigerer Temperatur usw. bis zum Schluss der Stoff mit dem höchsten Diffusionskoeffizienten bei der kleinsten Temperatur kommt. Dabei wird natürlich für jeden Dotierungsstoff eine andere Maske verwendet, d.h. es muss jedesmal die alte SiO_2-Maske entfernt werden und eine neue Maske lithografisch aufgebracht werden.

Für die praktische Ausführung der Diffusion werden die Wafer in einem Ofen aufgestellt und von einem Trägergas wie z.B. Stickstoff umströmt. Das Trägergas wird entsprechend mit den gasförmigen Dotierungsstoffen angereichert, die durch die Öffnungen in der Oxidmaske in die Wafer eindringen.

Als Alternative zur Diffusion steht die *Ionenimplantation* zur Verfügung, bei der die Dotierungsstoffe ionisiert und beschleunigt werden, so dass sie regelrecht in den Kristall „eingeschossen“ werden. Durch die Geschwindigkeit beim Einschuss lässt sich dabei die Eindringtiefe steuern und so kann – anders als bei der Diffusion – auch eine Anreicherung in tieferen Schichten erreicht werden.

1.1.5 Depositionsverfahren, Epitaxie, Metallisierung

Das Ziel von Depositionsverfahren ist die Abscheidung von Materialschichten auf der Oberfläche des Kristalls. Wenn Material so abgeschieden wird, dass es die Gitterstruktur der Unterlage übernimmt und als Einkristall weiter aufwächst, spricht man von *Epitaxie*. Bei der Gasphasenepitaxie wird Siliziumtetrachlorid chemisch zerlegt und frei werdendes Silizium lagert sich genau an den freien Stellen der Oberfläche des Kristallgitters an. So ist es möglich, einen Siliziumeinkristall weiter aufwachsen zu lassen. Dabei können nach Wunsch Dotierungsstoffe zugemischt werden, die in den neuen Bereich mit einwachsen. Ein Vorteil liegt darin, dass die Dotierung sehr flexibel gesteuert werden kann, z.B. n-p-n. Durch Änderung der Umgebungsparameter können auch polykristalline Siliziumschichten erzeugt werden. Daraus wird oft das Gate der Feldeffekttransistoren hergestellt.

Verbreitet sind die verschiedenen Varianten der *Chemical Vapour Deposition* (CVD). Die CVD basiert auf der Zersetzung von Molekülen eines Gases, das in der Summe alle Komponenten für die gewünschte Schicht enthält. Die Zersetzung kann thermisch bei mehr oder weniger hohem Gasdruck erfolgen oder durch das Plasma einer kontrollierten Gasentladung. Aus den Zersetzungsprodukten bildet sich durch eine entsprechende chemische Reaktionskette am Ende der gewünschte Stoff und lagert sich auf der Oberfläche ab.

Neben den bisher genannten chemischen Verfahren gibt es noch die physikalischen Verfahren. So kann man auch mit einem Molekularstrahl ein epitaktisches Wachstum erzeugen.

Wichtig ist die Erzeugung metallischer Schichten für Leiterbahnen und Kontakte, die *Metallisierung*. Diese Schichten erzeugt man z.B. durch thermisches Aufdampfen oder Kathodenzerstäubung (Sputtern). Um die n- oder p-dotierten funktionalen Bereiche der Schaltungen zu kontaktieren, werden Metallschichten direkt dort aufgebracht. Diese werden dann meist als Leiterbahnen weitergeführt und verbinden die Schaltungselemente miteinander. Auch für die Verbindung zu den Anschlussstiften des Gehäuses werden metallische Leiterbahnen gebraucht. Sie werden von den betreffenden Schaltungselementen zum Rand des Chips geführt, wo sie sich zu Kontaktflächen (Pads) aufweiten. Auf diesen wird dann ein feiner Draht befestigt (Bonding), der zu den Anschlussstiften führt. Im letzten Schritt der Herstellung werden die Chips mit Schutzoxid überzogen, das nur die Metallkontakte frei lässt.

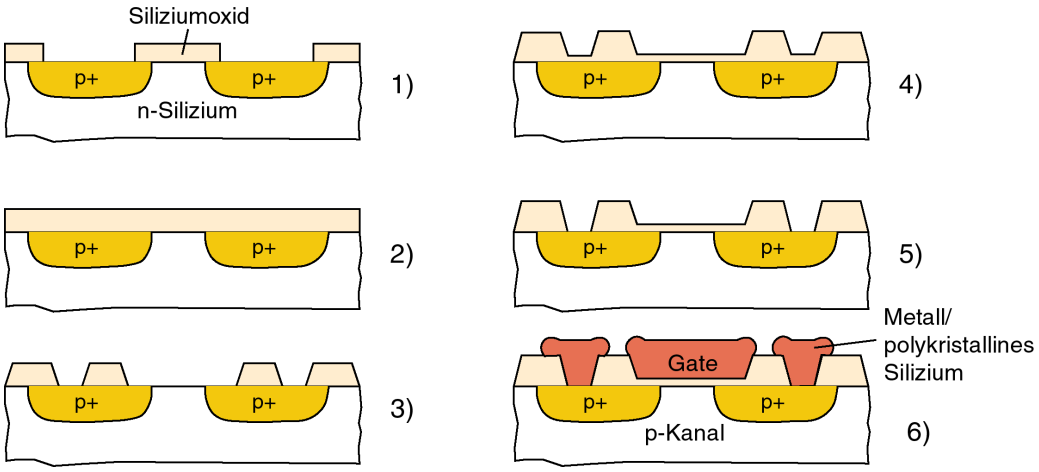

Abbildung 1.4: Die Herstellung eines Feldeffekttransistors in einem integrierten Schaltkreis (Erläuterung im Text).

Als abschließendes Beispiel zeigen wie hier die schrittweise Herstellung eines Feldeffekttransistors in einem integrierten Schaltkreis (Abb. 1.4). In Schritt 1) wird fotolithografisch eine Maske erzeugt, die zwei Öffnungen enthält für Drain und Source. Durch diese Maske werden in das schwach n-leitende Substrat zwei stark p-leitende Bereiche eindotiert. In den Schritten 2) und 3) werden Öffnungen für die späteren Metallkontakte von Drain und Source vorbereitet. Das Oxid über dem Kanal wird dabei zunächst vollständig abgetragen, um in Schritt 4) dort ein sehr dünnes Gateoxid abzulagern. In Schritt 5) werden die dabei verschlossenen Öffnungen für Drain und Source wieder freigelegt. In Schritt 6) werden durch Metallisierung das Gate und die Kontakte für Drain und Source erzeugt – der Feldeffekttransistor ist fertig. Das Einbringen der funktionalen Elemente und ihrer Anschlüsse von einer Seite aus nennt man *Planartechnik*. Abb. 1.5 zeigt Mikroskopaufnahmen von integrierten Schaltkreisen im Fertigungsprozess.

1.1.6 Funktionstest und Montage der Chips

Nach der Herstellung wird die Güte des Herstellungsprozesses an speziellen Testschaltungen überprüft, die alle kritischen Elemente der zu fertigenden Schaltkreise enthalten und zwischen diese platziert werden (Abb. 1.6). Dazu kann ein Nadelbettadapter von oben auf den Wafer

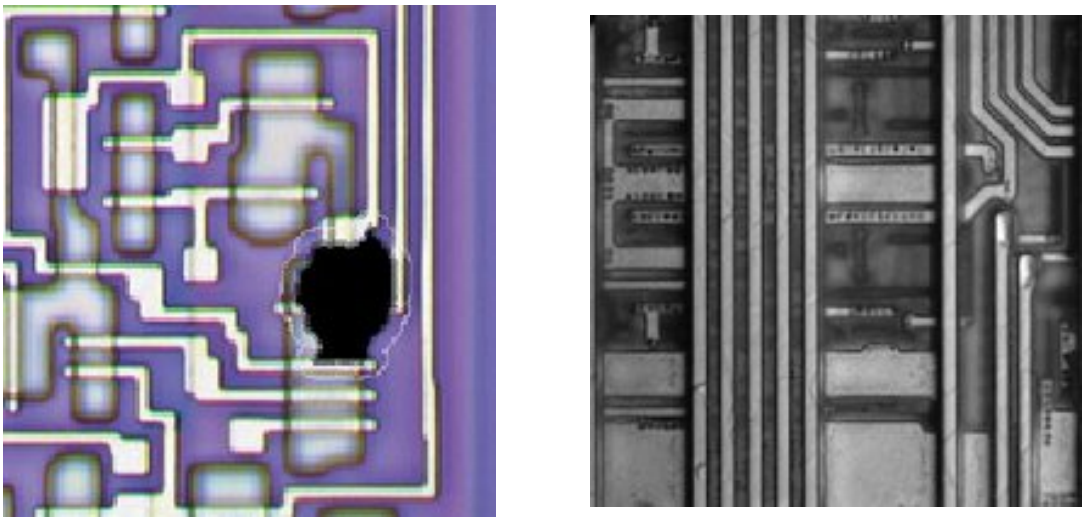

Abbildung 1.5: Mikroskopaufnahmen von integrierten Schaltkreisen auf einem Wafer im Fertigungsprozess.
Linkes Bild: Mehrere Feldeffekttransistoren sind in Entstehung. Die verwaschen-hellen Bereiche sind dotiertes Silizium und bilden Source und Drain. Die schmalen hellen Streifen sind Leiterbahnen aus polykristallinem Silizium mit einer Breite von 1.2 μm; diese stellen auch das Gate der Transistoren dar. Die quadratischen Aufweitungen der Bahnen sind für die spätere Kontaktierung vorgesehen. Der schwarze Bereich ist ein Defektpartikel, der von einer speziellen Bildverarbeitungs-Software gefunden, schwarz ausgefüllt und weiß umrahmt wurde.
Rechtes Bild: Mehrere Ebenen aus Silizium, polykristallinem Silizium und Metall in unterschiedlicher Höhenlage. In einem speziellen Verfahren (Multifokus) werden nacheinander mehrere Ebenen scharf gestellt und durch einen Computer zu einem einzigen Bild zusammengesetzt. Die Breite der Leiterbahnen ist 2.0 μm, die Aufnahme wurde mit sehr kurzwelligem ultravioletten Licht (Deep UV) gemacht.
Abbildungen mit freundlicher Unterstützung der Fa. Leica Microsystems, Wetzlar.

gesetzt werden, dessen Nadeln genau die Kontaktflächen der Testschaltungen treffen. Durch Staubpartikel, die während des Fertigungsprozesses auf den Wafer gelangen, können einzelne Schaltkreise unbrauchbar werden (s. Abb. 1.5, linkes Bild). Der ganze Fertigungsprozess wird daher unter extremen Reinraumbedingungen durchgeführt. Um die trotzdem noch auftretenden Defekte zu entdecken, müssen daher ständig Wafer-Inspektionen und weitere Tests an den Schaltkreisen erfolgen.

Wenn alle Tests erfolgreich waren, beginnt die Montage. Die Chips werden mit einem ultradünnen Sägeblatt von 25 μm Dicke aus dem Wafer gesägt (Abb. 1.7).[4] Die vereinzelten Chips werden nun auf Trägermaterial befestigt. An den Kontaktflächen werden dünne Gold- oder Aluminiumdrähte befestigt, meist durch das sog. *Bonding*: Eine Verschweißung der Materialien ohne Aufschmelzung durch Ultraschall oder Thermokompression. Es gibt sogar Verfahren, die alle Kontakte in einem Schritt herstellen.

Abschließend wird der Chipträger samt Chip in ein Kunststoffgehäuse mit Anschlussstiften im Rastermaß eingesetzt und die Bondingdrähte mit den Anschlussstiften verbunden. Zum

[4] Das früher übliche Ritzen und Herausbrechen der Chips wird nur noch selten angewandt, weil die heutigen großen Waferscheiben zu dick dafür sind.

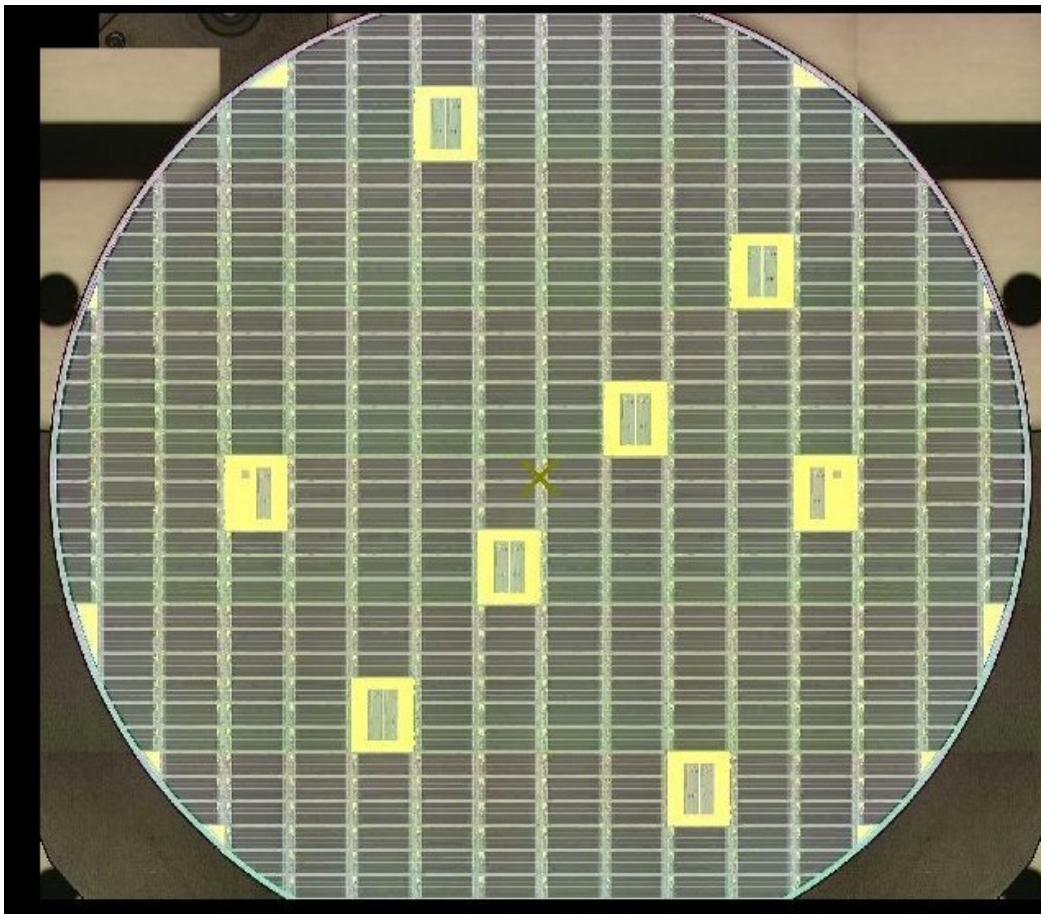

Abbildung 1.6: Übersichtsaufnahme eines 200mm-Wafers. Die hellen Linien zwischen den Schaltungen sind die Gräben an denen die Schaltungen später getrennt werden. Die vereinzelten hell hervorgehobenen Strukturen sind Testschaltungen. Abb. mit freundlicher Unterstützung der Fa. Leica Microsystems, Wetzlar.

Schutz gegen mechanische Belastungen sowie Feuchtigkeit werden die Chips mit Kunststoff umspritzt. Nach dem Anbringen der Beschriftung auf das Gehäuse ist der IC nun in der handelsüblichen Form und kann in einen Sockel gesteckt oder direkt auf eine Platine gelötet werden.

1.2 Aufgaben und Testfragen

1. Was ist Lithografie und Epitaxie?
2. In einem Wafer soll ein Bereich dotiert werden. Ergänzen Sie die fehlenden Arbeitsschritte in der folgenden Aufzählung: Auftragung von Fotolack – ? – unbelichteten Fotolack herauslösen – ? – restlichen Fotolack entfernen – ? – restliches Oxid entfernen

Lösungen auf Seite 26.

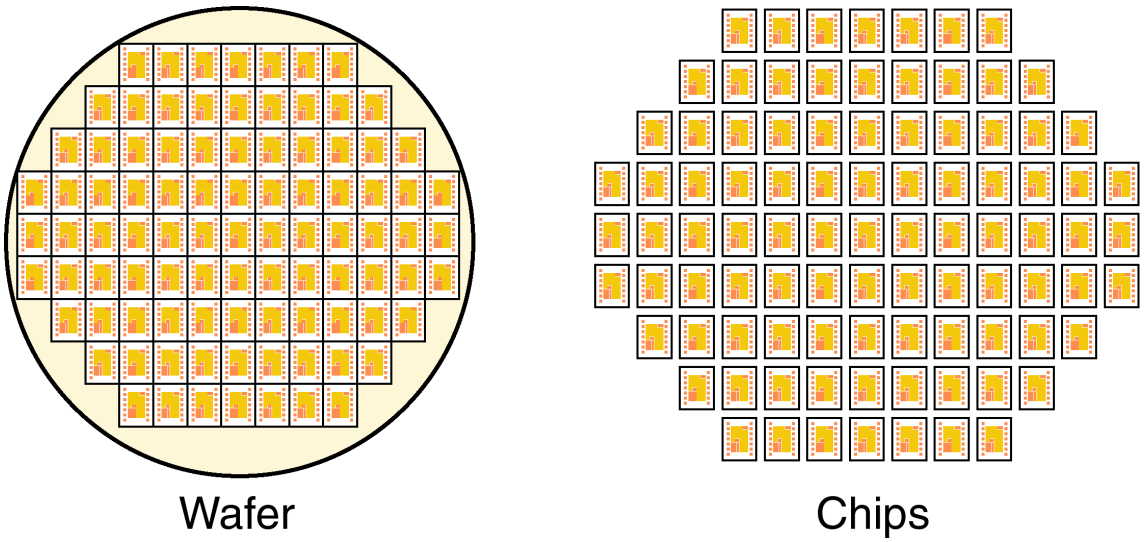

Abbildung 1.7: Aus dem Wafer werden die Chips herausgetrennt.

2 Einfache Mikroprozessoren

2.1 Fallbeispiel: Motorola 6800

2.1.1 Übersicht

Der Mikroprozessor 6800 des Herstellers Motorola (kurz: M6800) ist ein wirklich alter Prozessor. Wir wollen kurz auf ihn eingehen, weil er einfach aufgebaut ist und ein gutes Beispiel ist für das, was bisher über einfache Mikroprozessoren gesagt wurde. Der M6800 war der erste Prozessor der 68XX-Familie. Er wurde 1974/75 eingeführt und ist der „Urahn“ vieler Prozessoren und Controller, die heute noch aktuell sind.

Der M6800 ist ein 8-Bit-Prozessor, d.h. seine Verarbeitungs- und Datenbusbreite sind 8 Bit. Er hat 16 Adressleitungen und somit 64 Kbyte Adressraum. Ein-/Ausgabe wird mit speicherbezogenen Adressen innerhalb dieses Adressraumes durchgeführt. Die Systemsteuerung erfolgt über elf Steuerleitungen, darunter befinden sich ein maskierbarer Interrupteingang (IRQ) und ein nicht maskierbarer Interrupteingang (NMI). Der M6800 kennt mehrere Betriebszustände. Bei der normalen Befehlsabarbeitung wird über das PC-Register der Maschinencode eingelesen und bearbeitet. Nach einem Reset (durch Power-On oder RESET-Signal) liest der M6800 eine Adresse von den Speicherplätzen FFFEh und FFFFh (Restart-Vektor) in den PC und beginnt mit der Programmausführung an dieser Adresse. Der Restart-Vektor und die Boot-Routine, auf die er verweist, müssen in einem ROM-Speicher liegen. Es gibt drei Arten von Interrupts: Maskierbarer Interrupt IRQ, nicht-maskierbarer Interrupt NMI und Software-Interrupt über den Befehl SWI. Für jede der drei Interruptarten ist an einer vereinbarten Speicheradresse die Adresse (der Vektor) einer Behandlungsroutine hinterlegt. Alle Interrupt-Behandlungsroutinen werden durch den Befehl RTI (Return from Interrupt) abgeschlossen. Die Interruptquellen können z.B. in einer Daisy-Chain organisiert sein, die dann von den Behandlungsroutinen abgefragt wird. [3] Ein Signal am $\overline{HALT}$-Eingang versetzt den M6800 nach Beendigung des laufenden Befehls in einen Wartezustand. Die folgende Tabelle gibt einen Überblick über diese Betriebszustände.

Betriebszustand	Bearbeitung
Normale Befehlsabarbeitung	
RESET	laden des Restart-Vektors von FFFEh, FFFFh
Nicht maskierbarer Interrupt NMI	laden des NMI-Vektors von FFFCh, FFFDh
Software-Interrupt (Befehl SWI)	laden des Restart-Vektors von FFFAh, FFFBh
Maskierbarer Interrupt IRQ	laden des Restart-Vektors von FFF8h, FFF9h
HALT-Zustand	Prozessor geht in Warteschleife bis HALT-Signal deaktiviert

Die ALU des M6800 setzt Flags für Übertrag (Carry, C), Überlauf (Overflow, O), Null (Zero, Z), Negativ (N) und Halbübertrag (Half Carry, H). Die möglichen Adressierungsarten sind: unmittelbar, direkt, erweitert und registerindirekt. Der Prozessortakt liegt zwischen 100 kHz und 2 MHz. Der Prozessor ist in einem 40-poligen Gehäuse untergebracht, wobei nicht einmal alle Leitungen benutzt werden [2].

2.1.2 Anschlussleitungen

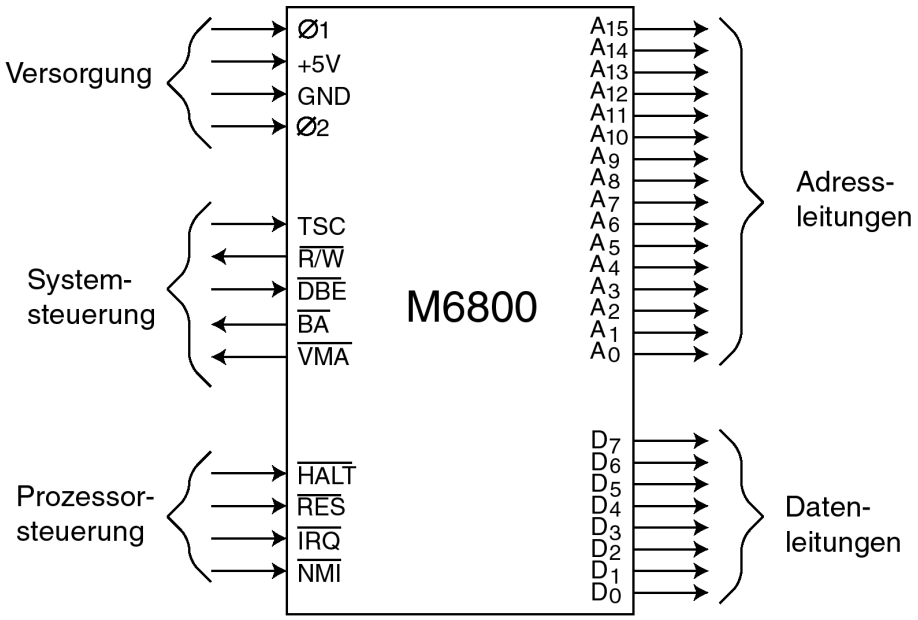

Abbildung 2.1: Der M6800 mit gruppenweise gezeichneten Anschlussleitungen.

Beim M6800 haben alle Leitungen eine eindeutige Bedeutung, es gibt kein Leitungs-Multiplexing und die Leitungen lassen sich einfach nach Gruppen ordnen (Abb. 2.1). Hier eine kurze Beschreibung der Anschlussleitungen, die alle TTL-Pegel führen:

A0 – A15 Adressleitungs-Ausgänge, Adressierung des Speichers und der E/A-Bausteine.

D0-D7 Datenleitungen, bilden den bidirektionalen Datenbus.

R/W Read/Write (Schreiben/Lesen), zeigt schreibenden oder lesenden Buszugriff an.

IRQ Interrupt Request (Unterbrechungsanforderung) maskierbarer Interrupt-Eingang.

NMI Non Maskable Interrupt, nicht maskierbarer Interrupt-Eingang.

HALT versetzt den Mikroprozessor nach Beendigung des laufenden Befehls in einen Wartezustand.

RES RESET, zurücksetzen in den Initialisierungzustand.

VMA Valid Memory Adress (Gültige Speicheradresse), zeigt gültige Speicheradressen an.

BA Bus Available, zeigt hochohmige (verfügbare) Busleitungen an.

DBE Data Bus Enable (Datenbus aktivieren), Datenbustreiber der Busschnittstelle extern freischalten.

TSC Tristate Control, versetzt den Adressbus und R/W in den hochohmigen Zustand.

ϕ_1, ϕ_2 Takteingänge, phasenverschoben und nicht-überlappend.

+5V Pluspol der Versorgungsspannung.

GND Ground (Masse), Minuspol der Versorgungsspannung.

2.1.3 Interner Aufbau

Der Registersatz des M6800 ist einfach gehalten (Abb. 2.2). Für arithmetische Operationen werden die *8 Bit-Akkumulatoren A und B* benutzt. Die Register des Adresswerks dagegen sind wegen der 16 Adressleitungen 16 Bit breit: Das *Stackpointer-Register* SP übernimmt die Stack-Verwaltung und wird bei PUSH- und POP-Befehlen automatisch nachgeführt. Das *Indexregister* X ermöglicht die registerindirekte Adressierung (Index-Adressierung) des Speichers, der *Program Counter* PC ist der Programmzähler, der den Maschinencode adressiert.

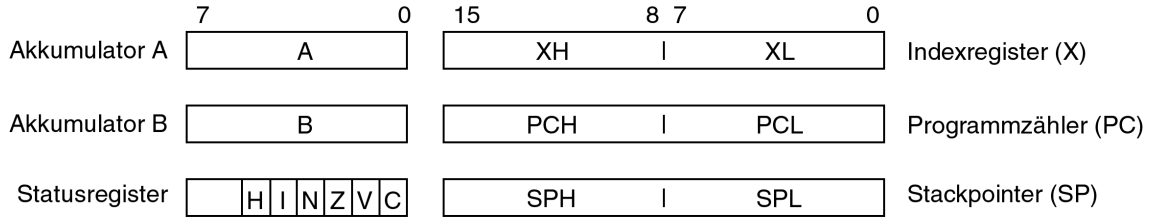

Abbildung 2.2: Die Register des M6800.

Der M6800 ist ein einfaches Beispiel für die Realisierung eines Prozessors mit den Bestandteilen Steuerwerk, Operationswerk, Registersatz, Adresswerk und Businterface. In Abb. 2.3 ist das allgemeine Blockschaltbild eines Mikroprozessors [3] entsprechend konkretisiert.

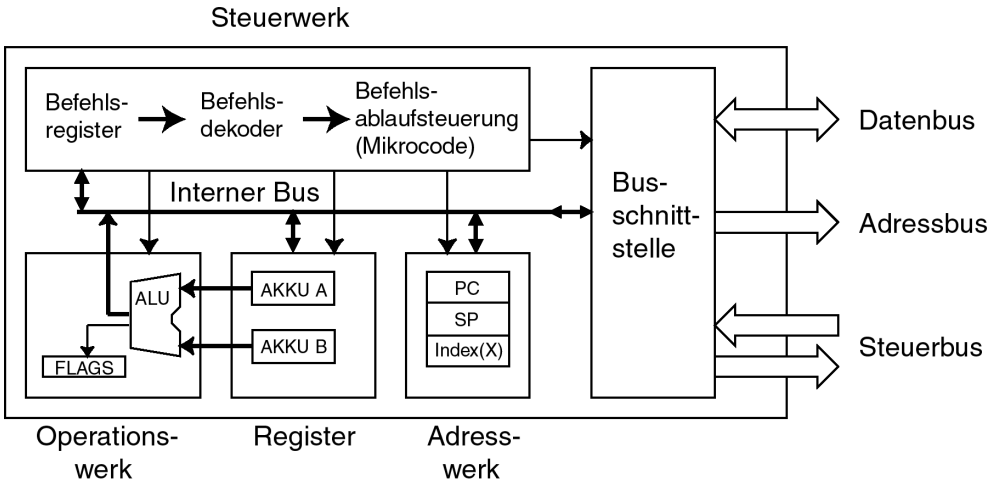

Abbildung 2.3: Der interne Aufbau des 6800 nach Funktionsblöcken.

2.1.4 Befehlssatz

Der M6800 ist ein CISC-Prozessor mit einem relativ großen Befehlssatz, den wir hier nicht vollständig behandeln können. Wir wollen aber zu den in [3] vorgestellten Befehlsgruppen einige Beispiele geben. Der Buchstabe X steht hier stellvertretend für ein A oder B und bezeichnet einen der beiden Akkumulatoren.

Transportbefehle
LDA X, Load accu A, Transport vom Speicher in den Akku.
STA X, Store accu B, Transport vom Akku in den Speicher.
LDX, Load index register, Laden des Indexregisters.
STS, Store stack pointer, Stack Pointer speichern.
PSH X, Push data on Stack, Daten auf Stack ablegen.

Arithmetische Befehle
ABA, Add accu B to accu A, Addiere Akku B zu Akku A.
ADD X, Add accu and memory, Speicherinhalt zu Akkumulator X addieren.
INC X, Increment accu X, Inkrementiere Akkumulator X.
DEX, Decrement index register, Dekrementiere Indexregister.
NEG X, Negate accu X, Negieren (Zweierkomplement) von Akkumulator X.
CMP X, Compare accu with memory, Akku mit Speicher vergleichen.
Es gibt keinen Multiplikations- und keinen Divisionsbefehl. Gleitkommazahlen kann der M6800 ebenfalls nicht bearbeiten.

Bitweise Logische Befehle
AND X, And accu, Logisches UND mit Speicheroperand und Accu X.
ORA X, Or accu, Logisches ODER mit Speicheroperand und Accu X.
EOR X, Exclusive Or accu, Logisches exklusives ODER mit Speicheroperand und Accu X.

Schiebe- und Rotationsbefehle
LSR X, Logical shift right Accumulator X, Akkumulator X bitweise nach links schieben.
ROL X, Rotate Left Accumulator X, Akkumulator X bitweise nach links rotieren.

Sprungbefehle
BRA, Branch always, unbedingt verzweigen (springen).
BEQ, Branch if equal, verzweige wenn gleich (ZF=1).
JSR, Jump to subroutine, Unterprogrammaufruf.
RTS, Return form Subroutine, Rücksprung aus Unterprogramm.
SWI, Software-Interrupt.

Prozessorsteuerung
CLC, Clear carry, Carry-Flag löschen.
SEI, Set interrupt mask, Interrupt-Freischaltungsbit setzen.

Der M6800 kennt vier *Adressierungsarten*, die wir am Beispiel von LDA (Load Accumulator A) zeigen wollen:

`LDA A #25` Unmittelbare Adressierung: Den Wert 25 in den Akku A laden.

`LDA A $3F` Direkte Adressierung: Inhalt der Speicherzelle 3Fh in den Akku A laden.

`LDA A $4A00` Erweiterte direkte Adressierung: Inhalt der Speicherzelle 4A00h in den Akku A laden.

`LDA A 10,X` Index-Adressierung: Die Summe aus Indexregister und Displacement 10 ist die Adresse der Speicherzelle, die in den Akku A geladen wird.

2.1.5 Programmierung

Ein Programm wird zunächst in *Assemblersprache* geschrieben und dann von einem *Assembler* in binären Maschinencode übersetzt. Dieser wird vom Prozessor erkannt und ausgeführt.

Beispiel 1 Es wird 9+32-8 berechnet und auf die Variable 'Summe' gespeichert. Für dieses erste Beispiel ist in der Art eines Übersetzungsprotokolls (Assemblierungsprotokoll) nicht nur der Assemblercode samt Kommentar, sondern auch der Maschinencode mit zugehörigen Speicheradressen angegeben:

Adresse im Prog.speich.	Masch.code, Operanden	Name für Speicherplatz	Assemblerbefehl	Kommentar
		ORG 000Ah		;Origin: Programm wird
				;bei 000Ah abgelegt
000A	0001	Summe	RMB 1	;Reserve Memory Byte
000B	86 09		LDA A \#9	;9 in Accu A laden
000D	8B 20		ADD A \#32	;32 zu Accu A addieren
000F	80 08		SUB A \#8	;8 von Accu A subtrah.
0011	97 0A		STA A Summe	;Ergebnis in Variable
				;Summe speichern
			END	;Ende der Übersetzung

Beispiel 2 Das folgende Programm löscht 32 Speicherplätze ab Adresse 500h.

```
    LDX #500h       ;Lade Indexregister mit 500h
L1  CLR 0,X         ;Indexadressierung: Löschen von Speicherplatz (500+0)
    INX             ;Increment Indexregister
    CPX #520h       ;Compare Index, vergleiche Indexregister mit 520h
    BNE L1          ;Branch if not equal to L1
    END             ;Ende der Übersetzung
```

Aus dem Motorola M6800 wurden die Prozessoren 6802 und 6809 sowie die Microcontroller 6801, 6805 und 6811 abgeleitet, deren Kern dem 6800 entspricht. In Abb. 2.4 ist die Systemplatine eines mobilen Roboters zu sehen, der durch einen Motorola 68HC11-Mikrocontroller gesteuert wird.

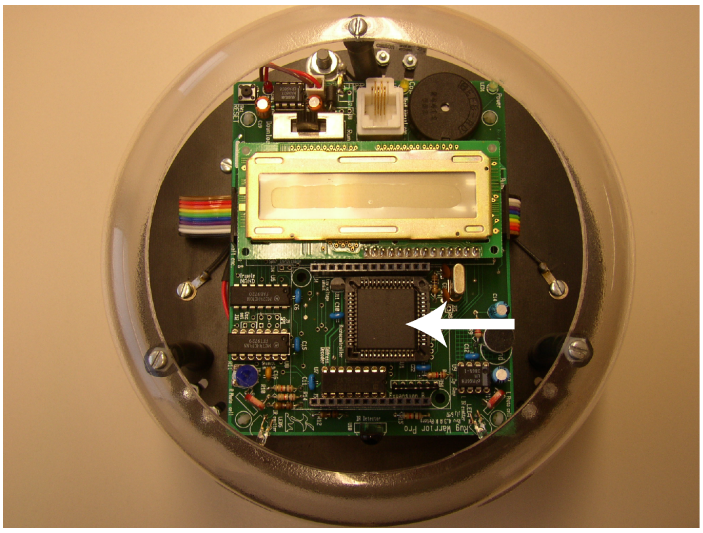

Abbildung 2.4:
Die Systemplatine eines mobilen Roboters mit 68HC11-Mikrocontroller (quadratischer Chip, s. Pfeil).

2.2 Fallbeispiel: Intel 8086

Der Intel 8086 war Intels erster 16-Bit-Prozessor. Er wurde 1978 auf den Markt gebracht und wurde sehr bekannt, weil die Fa. IBM ihren ersten *Personal Computer* (persönlicher Computer, *PC*) damit ausstattete. Der IBM-PC war das erste Modell der heutzutage allgegenwärtigen PCs und der Aufbau des 8086 hat die PC-Welt bis heute beeinflusst. Alle Befehle und internen Strukturen des 8086 sind bis zum heutigen Tag (und vermutlich noch lange) in allen modernen PC-Prozessoren vorhanden. Der 8086 erscheint uns vielleicht heute als einfacher Prozessor, bei seinem Erscheinen war er sehr fortschrittlich.

Der 8086 kann Daten in 8- oder 16-Bit-Breite laden, speichern und verarbeiten. Er besitzt einen 20-Bit-Adressbus, mit dem er 1 Mbyte Speicher adressieren kann – für damalige Verhältnisse enorm viel. Alle gängigen Befehle der Ganzzahlarithmetik für vorzeichenlose Zahlen und für Zahlen im Zweierkomplement sind vorhanden. Eine Besonderheit sind die Blockbefehle („Stringbefehle“) des 8086, mit denen er bis zu 64 Kbyte Daten in einem Befehl bearbeiten kann.

Der 8086 wurde in CMOS und NMOS-Technik hergestellt, er enthält 28000 Transistoren und wurde anfangs mit 5, später mit 8, 10 und mehr MHz getaktet. Viele Leitungen sind gemultiplext. Weil man ihn sowohl für leistungsfähige Mikrocomputer als auch für einfache Steuerungen gedacht hatte, lässt er sich in zwei Betriebsarten betreiben: Minimum- und Maximumbetrieb. Im Maximumbetrieb braucht er zwar einen speziellen Buscontroller-Baustein, kann dann aber auch mit weiteren Bausteinen, wie dem Gleitkomma-Koprozessor 8087 und dem DMA-Controller 8237A, zusammenarbeiten. Der 8086 besitzt eine *Prefetch Queue* (Befehls-Warteschlange) von sechs Byte Größe zur Geschwindigkeitssteigerung.

In Abb. 2.5 sind die Register des Intel 8086 gezeigt. Er besitzt weit mehr Register als sein Vorgänger 8085. Wie bei diesem sind aber die Register nicht gleichberechtigt, sondern haben bestimmte Spezialaufgaben. Alle Register haben eine Breite von 16 Bit, der 8086 ist ja ein 16-Bit-Prozessor. Eine Besonderheit ist, dass die Universalregister AX,BX,CX und DX auch in Hälften zu 8 Bit ansprechbar sind. Register AX setzt sich beispielsweise aus AH (High Byte) und AL (Low Byte) zusammen. Diese Registerhälften können völlig unabhängig programmiert

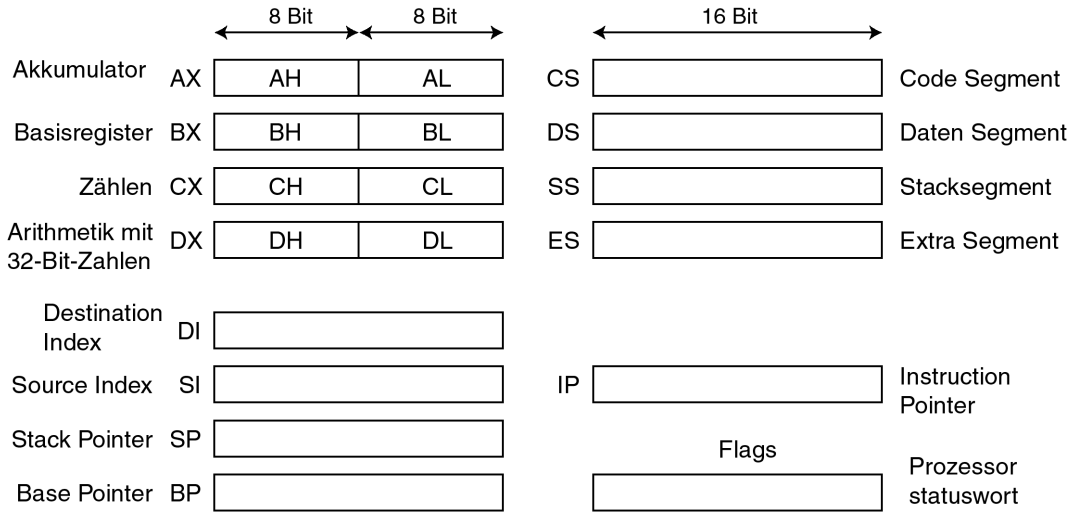

Abbildung 2.5: Die Register des Intel 8086. Jedes Register ist für bestimmte Spezialaufgaben vorgesehen.

werden z.B. mit `add bh,al`. Damit sollte die Portierung von 8-Bit-Programmen erleichtert werden. Die *Segmentregister* sind dem Adresswerk zuzurechnen.

2.2.1 Adressbildung und Speichersegmentierung

Aus Abb. 2.5 geht hervor, dass der Intel 8086 nur 16-Bit-Register hat, obwohl er einen 20-Bit-Adressbus besitzt. Im Gegensatz zu älteren und einfacheren Prozessoren – man vergleiche z.B. mit dem M6800 im vorigen Abschnitt – passt hier also eine Speicheradresse in keines der Register! Der Grund dafür ist, dass der 8086 eine einfache Form der *Speichersegmentierung* [3] unterstützt. Die Idee der Segmentierung ist, dass innerhalb der Segmente mit der Adressierung immer bei Adresse 0 begonnen wird. Dadurch können Segmente an beliebiger Adresse im Hauptspeicher eingeladen oder dorthin verschoben werden, ohne das sich die segmentinterne Adresse ändert. Außerdem kann man für Code, Daten und Stack eigene Segmente führen, was die Übersicht verbessert. Die Adressen bestehen nun allerdings aus zwei Anteilen: Der *Segmentanteil* beschreibt, welches Segment gemeint ist und der *Offset* beschreibt, wo die Adresse innerhalb des Segmentes liegt. Diese zweiteilige Adresse nennt man *logische Adresse* [3], sie wird beim 8086 in der Form `Segment:Offset` geschrieben. Im Programm arbeitet man immer mit logischen Adressen, dadurch ist man unabhängig von der tatsächlichen Lokation der Segmente zur Laufzeit.

Der Offset kann eine Konstante sein, er kann aber auch zur Laufzeit aus Registerinhalten bestimmt werden. Dafür stellt der 8086 eine komfortable basis-indizierte registerindirekte Adressierung [3] nach der Formel *Offset = Basisregister + Indexregister + Displacement* zur Verfügung. Die Basisregister sind BX und BP, die Indexregister sind DI und SI. Das Displacement kann 8 oder 16 Bit groß sein. Der Segmentanteil liegt beim 8086 in einem der vier *Segmentregister* CS, DS, SS oder ES. Aus Segmentanteil und Offset wird dann die real auf dem Adressbus wirkende *physikalische Adresse* vom Adresswerk gebildet, wie in Abb. 2.6 dargestellt ist.

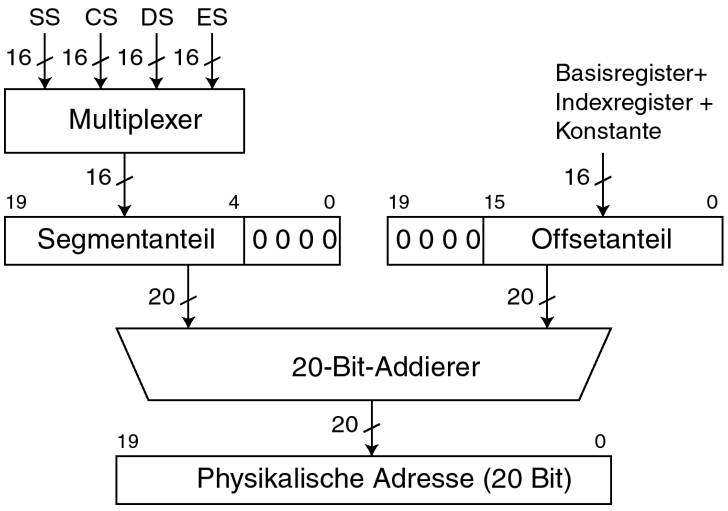

Abbildung 2.6: Die Adressbildung aus Segmentanteil und Offset beim Intel 8086.

Die Verschiebung des Segmentanteils um vier Bit nach links entspricht einer Multiplikation mit 16, die Adressbildung kann daher formelmäßig so ausgedrückt werden:

```
Physikalische Adresse = 16*Segmentanteil + Offsetanteil.
```

Die Adressbildung des 8086 zieht auch eine Strukturierung des Speichers nach sich. Die erste Adresse eines Segmentes hat den Offset 0, die letzte den Offset FFFFh. Das Segment hat also eine Größe von 64 Kbyte. Der Anfang der Segmente liegt physikalisch immer auf einem Vielfachen von 16.

Jede logische Adresse besteht also aus zwei unabhängigen 16-Bit-Anteilen. So wird z.B. der Programmzähler durch das Registerpaar CS:IP dargestellt. Das bringt natürlich Zusatzaufwand mit sich. Bei Änderung einer Adresse müssen jedesmal zwei Worte geändert werden. Wird z.B. ein Sprung, Rücksprung oder eine Interrupt Service Routine ausgeführt, muss das Register CS *und* das Register IP verändert werden. Dies bringt wiederum besondere Probleme mit sich, wenn zwischen den beiden Befehlen zur Umsetzung ein Interrupt ausgelöst wird, denn genau dann ist der Zeiger ungültig. Ein weiteres Problem ist der Vergleich von logischen Adressen, denn die Berechnung der logischen Adresse aus der physikalischen Adresse ist mehrdeutig.

Beispiel Aus der logischen Adresse B800:0012 ergibt sich die physikalische Adresse B8012h ebenso wie aus der logischen Adresse B007:7FA2 und vielen weiteren Kombinationen.

Um die Gleichheit von logischen Adressen festzustellen, muss man sie zuerst in eine Standardform bringen (normalisieren).

Die Segmentierung des 8086 war trotzdem ein guter Ansatz. Allerdings entwickelte sich die Computertechnik nach Erscheinen des 8086 sehr schnell weiter und die Größe der Programme wuchs in einer Geschwindigkeit, die niemand erwartet hatte. Nun wurde aus der 8086-Speichersegmentierung ein Hemmschuh. Eine Segmentgröße von 64 Kbyte reichte nicht aus und es fehlte die Möglichkeit, den Speicher unsegmentiert („flach“) zu adressieren. Man begann nun, mehrere Segmente zu größeren Daten- oder Codebereichen zusammenzusetzen. Das bedeutete aber, dass man mehrere Arten von Zeigern hatte, intra-segmentale Zeiger, die nur den Offset angeben (NEAR Pointer) und inter-segmentale Zeiger, die aus Segmentanteil und Offset bestehen (FAR Pointer). Diese Teilung zog sich durch Daten

und Code und führte zu den Speichermodellen wie SMALL, COMPACT, LARGE, HUGE usw. – wenig angenehm für die Programmierer!

Leider fehlte auch jede Art von Zugriffsschutz, so dass es schwer war, irgendeine Art von Multitasking zu implementieren. Genau das verlangte aber der Markt. Außerdem war der Adressraum wegen des etwas knappen 20-Bit-Adressbusses schon rein physikalisch auf 1 Mbyte begrenzt und stellte schon bald einen argen Flaschenhals dar. Intel reagierte auf diese Engpässe mit der Entwicklung des 80286, der 1982 auf den Markt kam. Der 80286 kann 16 Mbyte physikalisch und 1 Gbyte virtuell adressieren und besitzt außerdem ein Schutzkonzept um Multitasking zu unterstützen.

2.2.2 Interruptkonzept und Betriebssystem-Anbindung

Auf den Speicheradressen von 00000h – 003FF befindet sich die 1024 Byte lange *Interrupt-Vektoren-Tabelle*. Dort liegen die Interruptvektoren (Einsprungadressen) für die 256 möglichen Interrupts; jeder Interruptvektor besteht aus Segment und Offset und belegt vier Byte (Abb. 2.7).

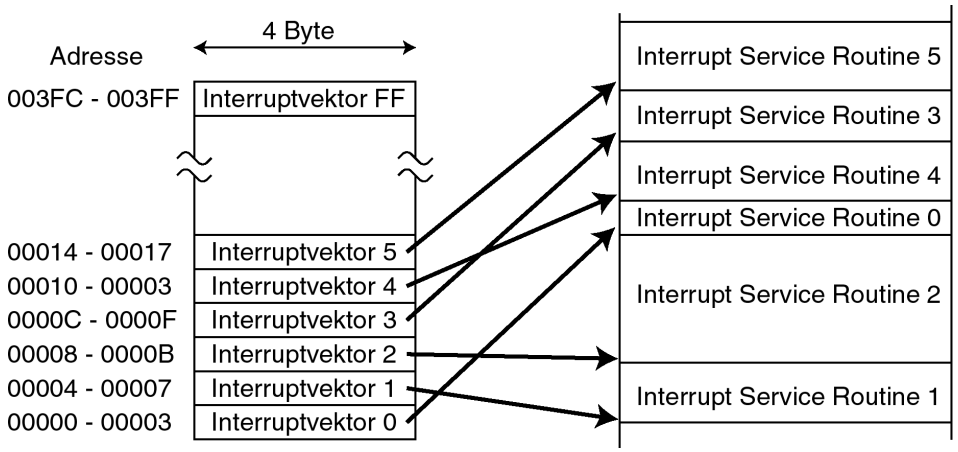

Abbildung 2.7: Die Interrupt-Service-Routinen des 8086 werden über die Interrupt-Vektoren-Tabelle aufgefunden.

Der 8086 verfügt über zwei Interrupt-Eingänge: Den vektorisierten, maskierbaren Eingang INTR (Interupt Request) und den nicht maskierbaren und nicht vektorisierten Eingang NMI (Non maskable Interrupt). NMI ist für schwerwiegende Störungen gedacht, z.B. für Fehler in der Speicherparitätslogik. Über INTR wird die „Routinearbeit“ abgewickelt, d.h. die Interrupt-Anforderungen eines normalen Betriebs. Intel hat dafür einen eigenen Interrupt-Controller entwickelt, den programmierbaren Interrupt-Controller PIC 8259A.

Die Zusammmensetzung von Adressen aus zwei Anteilen ruft eine Reihe von speziellen Problemen bei Interrupt-Service-Routinen hervor. Betrachten wir z.B. eine Befehlsfolge, die den aus SS und SP bestehenden Zeiger auf den Stack umsetzt:

```
mov sp,[stackpointer2]          ; unterbrechbar
mov ss,[stacksegment2]
```

Das Problem ist, dass diese Befehlsfolge durch einen Interrupt unterbrochen werden könnte. Dann ist also das SP-Register schon umgesetzt, das SS-Register noch nicht. Die aus SS und

SP berechnete phys. Adresse des Stackpointers ist also ungültig. Wenn die Interrupt-Service-Routine also den Stack benutzt, was häufig geschieht, überschreibt sie unter Umständen fremde Daten und der Rechner kann abstürzen.

Um das zu vermeiden, muss man die Reihenfolge der beiden Befehle umstellen, der 8086 sperrt nämlich wegen dieses Problems nach dem Laden eines Segmentregisters für den folgenden Befehl alle Interrupts:

```
mov ss,[stacksegment2]          ; nicht unterbrechbar
mov sp,[stackpointer2]
```

Ein ähnliches Problem entsteht beim Umsetzen von Interruptvektoren. Hier kann der Prozessor aber nicht erkennen, um was es geht. Der Programmierer muss daher vor der Umsetzung durch den Befehl `CLI` (Clear Interrupt Enable Flag) alle Interrupts global sperren und sie anschließend mit `STI` (Set Interrupt Enable Flag) wieder freigeben. Ähnliches gilt für alle Datenstrukturen, die für Unterbrechungen relevant sind, und mehr als 16 Bit umfassen. Eine Besonderheit im 8086-Code sind die Präfix-Bytes, mit denen für den nachfolgenden Befehl die Adressierung umgeschaltet wird.[3] Würde nach dem Einlesen eines Präfix-Bytes eine Unterbrechung erfolgen, würde des Präfix ungewollt auf einen anderen Befehl angewandt. Auch hier sperrt der 8086 automatisch für die Dauer des folgenden Befehls alle Unterbrechungen .

Die Größe der Interrupt-Vektoren-Tabelle erlaubt 256 Interrupt-Service-Routinen. In der Regel sind aber sehr viel weniger interruptfähige Geräte angeschlossen, typisch sind 8 – 15. Einige weitere ISRs sind den Ausnahmen zugeordnet.[3] Die restlichen ISRs werden für *Software-Interrupts* genutzt, die hauptsächlich dazu gedacht sind, elementare Betriebssystemfunktionen (System Calls) zu implementieren. Software-Interrupts werden mit dem `INT`-Befehl aufgerufen. Dabei wird als einziger Parameter die Nummer der gewünschten ISR angegeben, z.B. `INT 21h` um die ISR 21h (33d) aufzurufen. Diese Art des Aufrufs bietet viele Vorteile:

1. Ein Anwender kann Betriebssystemaufrufe vornehmen, ohne dass er Adressen der Interrupt-Service-Routinen kennen muss.
2. Bei einem Betriebssystem-Update werden die ISRs geändert und die Interrupt-Vektoren-Tabelle angepasst; die Anwenderprogramme können unverändert bleiben.
3. Ein Anwender kann eigene Interrupt-Service-Routinen, d.h. Betriebssystemerweiterungen, leicht erstellen. Er muss nur den Code im Speicher ablegen und die Anfangsadresse in der Interrupt-Vektoren-Tabelle eintragen.

Auf diese Art wurden z.B. die Betriebssysteme für die ersten PCs aufgebaut, wobei man die hardwarenahen Routinen im BIOS (Basic Input/Output System) und die höher liegenden Schichten im DOS (Disk Operating System, PC-DOS, MS-DOS) gruppierte.

Die folgende Aufzählung deutet an, wie in den ersten PCs die Software-Interrupts benutzt wurden um ein vollständiges Betriebssystem zu installieren.[1]

Ausnahmen, ISR 0 – 7 Divisionsfehler, Einzelschrittbetrieb, NMI, Breakpoint, ungültiger Opcode, kein Koprozessor vorhanden.

[1] Die genaue Belegung der Software-Interrupts ist vom Typ des Rechners und der Betriebssystemversion abhängig.

Hardware-Interrupts, ISR 8 – Fh Zeitgeberbaustein, Tastatur, serielle Schnittstellen COM1 und COM2, Festplatte, Diskettenlaufwerk, parallele Schnittstelle.

Software-Interrupts des BIOS, ISR 10h – 1Fh Bildschirmsteuerung, Systemkonfiguration ermitteln, Speichergröße ermitteln, serielle Schnittstelle bedienen, Drucker bedienen, Tastatur abfragen u.a.m.

Software-Interrupts von DOS, ISR ab 20h Tastaturfunktionen, Dateiverwaltung, Verzeichnisverwaltung, Laufwerksverwaltung, Speicherverwaltung, Programmverwaltung, Gerätesteuerung, Standardein- und ausgabe, Schnittstellen u.a.m.

Anwenderdefinierte ISRs, z.B. ISR 60h – 67h Diese Software-Interrupts sind für Anwenderprogramme frei verfügbar. Funktionen oder Programme, die hier hinterlegt werden, sind ebenso erreichbar wie die Betriebssystemroutinen. Man kann dadurch einfach Betriebssystemerweiterungen installieren.

Beispiel Im folgenden Code ist die Verwendung eines Betriebssystemaufrufs in Assembler gezeigt: Der Aufruf des Bildschirmtreibers Int 10h (BIOS) zur Bestimmung der Cursor-Position.

```
mov ah,03       ; Funktion 3: Lese Cursorposition
mov bh,0        ; Bildschirmseite 0
int 10h         ; Aufruf Software-Interrupts Nr.10h (Videotreiber
mov [Reihe],dh  ; Verwertung der Ergebnisse
mov [Spalte],dl ; Reihen- und Spaltenpos. im Speicher abgelegen
```

2.2.3 Systemstart

Nach dem Reset haben die Register folgenden Inhalt: DS=SS=ES=0000h, Flags=0002h, CS=F000h, IP=FFF0h. Der erste Befehl wird also geladen von der physikalischen Adresse F0000h+FFF0h = FFFF0h. Dort muss ein Sprungbefehl stehen, der in das *Urladeprogramm* (Bootprogramm, Bootstrap) verzweigt. Das Urladeprogramm lädt dann von einem Laufwerk das eigentliche Betriebssystem mit den Kernstücken Interrupt-Service-Routinen und Interrupt-Vektoren-Tabelle (Bootvorgang). Das Urladeprogramm muss zwangsläufig in einem nicht-flüchtigen Speicher stehen, damit es direkt nach dem Einschalten zur Verfügung steht. Das BIOS ist daher bei einem PC meistens in einem EEPROM auf der Systemplatine untergebracht.

Der erste Interrupt darf erst erfolgen, wenn der Bootvorgang abgeschlossen ist. Ein Interrupt während des Bootvorgangs könnte dazu führen, dass der Prozessor einen ungültigen Interrupt-Vektor lädt oder in einen Speicherbereich verzweigt, in dem noch keine ISR liegt. Das System verfängt sich dann meist in einer Endlosschleife und reagiert nicht mehr. Man hätte also das System schon beim Booten zum „Absturz“ gebracht. Daher ist nach dem Reset auch das Interrupt-Enable-Flag gelöscht und somit alle Interrupts über den IRQ-Eingang gesperrt. Ein Problem stellt der NMI-Eingang dar, der sich ja nicht sperren lässt. Daher wird im PC der NMI-Eingang über ein externes Gatter geführt und in der Bootphase gesperrt.

2.2.4 Adresspufferung, Buscontroller, Prefetch Queue, Wortausrichtung

Der 8086 hat einen gemultiplexten Adress-/Datenbus, d.h. zu bestimmten Zeiten liegen auf diesen Anschlussleitungen Adressen, zu anderen Zeiten Daten (Abb. 2.8).

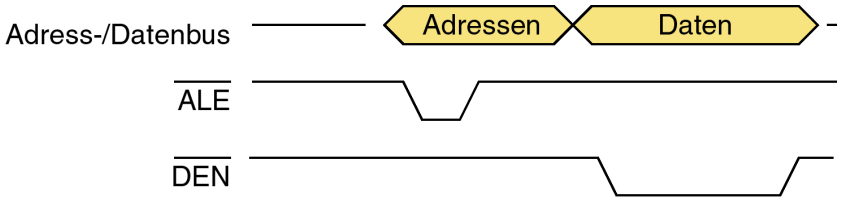

Abbildung 2.8: Der gemultiplexte Adress-/Datenbus des 8086 führt zeitweise Daten und zeitweise Adressen. Das Signal ALE (Adress Latch Enable) signalisiert die Übernahme in die Adresspuffer, das Signal DEN (Data Enable) aktiviert den Datenbustreiber.

Um Speicher- und E/A-Zugriffe durchführen zu können, müssen im System aber Daten und Adressen auf getrennten Busleitungen liegen. Man muss also den gemultiplexten Bus demultiplexen. Dazu werden die Adressen im richtigen Zeitpunkt vom gemultiplexten Bus des 8086 abgenommen und in Zwischenspeicher gelegt, die *Adresspuffer* (Adress Latches). An den Ausgängen der Adresspuffer ist direkt der System-Adressbus angeschlossen (Abb. 2.9). Zu Beginn eines Buszyklus erzeugt also der 8086 die physikalische Adresse und gibt sie über den gemultiplexten Bus aus. Über das spezielle Signal ALE (Adress Latch Enable) wird an den Adresspuffern eine Datenübernahme ausgelöst, so dass die Adresse jetzt zwischengespeichert ist. Nun schaltet der 8086 um und benutzt den gemultiplexten Bus für Daten. Diese laufen über einen bidirektionalen Bustreiber, der durch das Signal DEN (Data Enable) angesteuert wird.

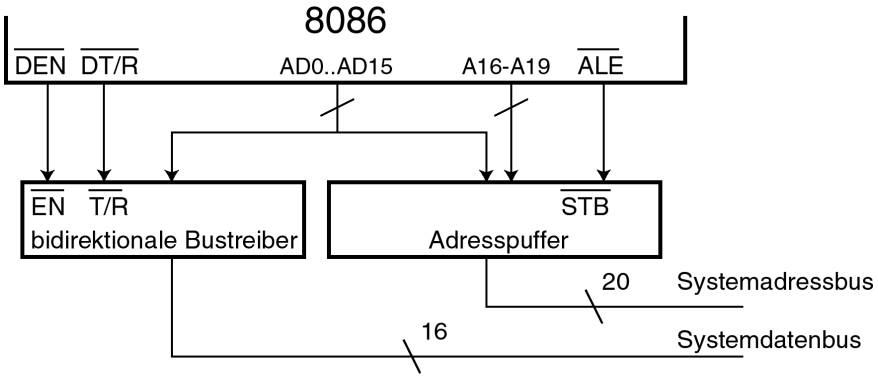

Abbildung 2.9: Das Demultiplexen des Adress-/Datenbusses beim 8086.

Im Minimum-Betrieb erzeugt der 8086 alle notwendigen Steuersignale für die Busse. Im Maximumbetrieb dagegen werden einige Anschlussleitungen für die mögliche Kommunikation mit Koprozessor oder DMA-Controller freigehalten. Nun stehen nicht mehr genug Anschlussstifte für Steuersignale zur Verfügung. Diese Problem hat Intel durch die Einplanung des *Buscontrollers* 8288 gelöst. Der 8086 übermittelt die Art des beabsichtigten Buszugriffes mit den Statussignalen S_0, S_1, S_2 (Pins 26 – 28) an den 8288 und dieser erzeugt die Steuersignale für den Systembus in korrekter zeitlicher Abstimmung. Die folgende Tabelle zeigt die möglichen Buszyklen des Bus-Controllers 8288:

S_2	S_1	S_0	Zustand des Prozessors
0	0	0	Interrupt-Annahme
0	0	1	I/O-Port lesen
0	1	0	I/O-Port schreiben
0	1	1	Haltzustand
1	0	0	Code lesen (Prefetch)
1	0	1	Speicher lesen
1	1	0	Speicher schreiben
1	1	1	passiv

Die *Prefetch Queue* ist ein prozessorinterner Zwischenspeicher in der Busschnittstelle für den eingelesenen Maschinencode (Befehlswarteschlange). Immer wenn der Prozessor gerade interne Aktivitäten ausführt und somit der Bus unbenutzt ist, wird selbstständig die Prefetch Queue mit Maschinencode nachgeladen. Beim Intel 8086 hat die Prefetch Queue eine Größe von sechs Byte und wird wortweise nachgeladen, sobald zwei Byte frei sind. So ist dafür gesorgt, dass die nachfolgenden sechs Bytes des Maschinencodes in der Regel schon in der Prefetch Queue liegen, wenn sie gebraucht werden. Dort ist ein schneller Zugriff möglich, man braucht nicht auf den langsamen Speicher zu warten. Enthaltene Opcodes können sogar schon dekodiert werden. Man erzielt also einen Geschwindigkeitsgewinn, und das ist tatsächlich der einzige Zweck der Prefetch Queue. Dieser geht allerdings wieder verloren, wenn im Programm ein Sprung ausgeführt wird und die Prefetch Queue komplett neu geladen werden muss.[2]

Der Intel-8086 hat zwar einen 16-Bit-Datenbus, wird aber an einen Byte-strukturierten Speicher angeschlossen. Dadurch sind also Speicherplätze mit gerader Adresse über die untere Hälfte des Datenbusses (Bit0 – Bit7) erreichbar und Adressen mit ungerader Adresse über die obere Hälfte (Bit8 – Bit15). Um den Zugriff zu steuern, wird einerseits das niedrigstwertige Adressbit A_0 sowie andererseits eine Steuerleitung namens $\overline{BHE}$ (Bus High Enable) benutzt. Wenn beides aktiviert ist, erfolgt ein 16-Bit-Zugriff auf eine gerade und die nachfolgende ungerade Adresse. Man kann also in einem Buszyklus auf ein ganzes Wort zugreifen, wenn es an einer geraden Adresse beginnt. Beginnt ein Wort aber an einer ungeraden Adresse, sind zwei Zugriffe notwendig, die jeweils ein Byte transportieren. Dies wird zwar vom Prozessor automatisch erledigt, kostet aber wertvolle Zeit. Die 8086-Assembler und -Compiler verfügen daher meist über eine Direktive, die 16-Bit-Dateneinheiten auf gerade Anfangsadressen legen, auch wenn dann ein Teil der Speicherplätze frei bleibt, die sog. *Wortausrichtung*. Man opfert also Speicherplatz zugunsten eines Geschwindigkeitsgewinnes.[3]

2.2.5 Der Intel 8088

Der Intel 8088 wurde nach dem 8086 auf den Markt gebracht. Im Gegensatz zum 8086 hat er nur einen 8-Bit-Datenbus und eine 4-Byte-Prefetch-Queue. Er muss also jeden 16-Bit-Zugriff als Folge von zwei 8-Bit-Zugriffen ausführen und ist dadurch deutlich langsamer. Warum brachte man ihn auf den Markt? Der Grund ist, dass man den Aufbau von preiswerten Motherboards ermöglichen wollte. Damals waren 8-Bit-Treiberbausteine wesentlich verbreiteter

[2] Die Fortsetzung dieser Idee führt dann zu solch ausgetüftelten Prozessorfeatures wie Code-Caches, Pipelining und Sprungvorhersagen.

[3] Bei 32-Bit-Prozessoren arbeitet man mit Doppelwortausrichtung.

und billiger zu haben als 16-Bit-Bausteine. Ein 8088-Board war also langsamer, konnte aber wesentlich billiger angeboten werden. Dies war der Anfang einer Reihe „abgespeckter“ Prozessorvarianten, mit denen Intel immer wieder um das Low Cost-Marktsegment gekämpft hat (8088, 80386SX, 80486SX, Celeron, u.a.m.).

2.3 Aufgaben und Testfragen

1. a) Ergänzen Sie in dem folgenden 6800-Code den fehlenden Befehl.
   b) Welche Art der Adressierung wird in dem CLR-Befehl benutzt?

```
; Der nachfolgende Code schreibt eine 0 in die Speicherzellen A8h - B0h.
            LDX #8          Lade Indexregister mit 8
L1          CLR 0A8h,X      Löschen von Speicherplatz (Indexregister+8)
            ???             Welcher Befehl muss hier stehen?
            BNE L1          Branch if not equal to L1
            END             Ende Übersetzung
```

2. Was bewirkt das unten abgedruckte M6800-Programm?

```
            LDA B #FF       Lade Akku B mit 255
            LDX #10         Lade Indexregister mit 10
L1          STA B 0,X       Inhalt von Accu B speichern nach (0+X)
            INX             Inkrementieren Indexregister
            DEC B           Akku B dekrementieren
            BNE L1          Branch if not equal to L1
            END             Ende Übersetzung
```

3. Berechnen Sie für die folgenden logischen Adressen auf einem Intel 8086-System jeweils die physikalische Adresse: a) 4230:8040, b) 4B00:0240, c) FF0A:A100.

4. In einem 8086-System enthält des DS-Register den Wert B800h und das ES-Register den Wert C400h. Überlappen die durch die beiden Register adressierten Segmente? Wenn ja, drücken Sie die Überlappung in Kbyte und in % aus.

5. Auf welchem Speicherplatz steht der erste Befehl, der bei einem i8086 nach dem Booten ausgeführt wird, welche Art von Speicher findet man dort?

6. Wo steht beim 8086 die Einsprungadresse der Service Routine zu Interrupt Nr.6?

7. Wie kann die Prefetch Queue die Performance des 8086 verbessern?

8. Wozu benutzt der 8086 Adress Latches, wozu einen Buscontroller?

Lösungen auf Seite 26.

Lösungen zu den Aufgaben und Testfragen

Zu Kapitel 1: Halbleiterbauelemente

1. Lithografie ist eine Technik, um Fotolack mit Öffnungen auf der Siliziumoxidschicht aufzubringen; Epitaxie ist geordnetes Schichtenwachstum, z.B. auf einem Siliziumkristall.

2. Die Reihenfolge der Schritte ist: Auftragung von Fotolack – Belichtung mit der Maske – belichteten Fotolack herauslösen – freiliegendes Oxid wegätzen – restlichen Fotolack entfernen – Dotierungsstoffe einbringen – restliches Oxid entfernen.

Zu Kapitel 2: Einfache Prozessoren

1. a) Dekrementieren des Indexregisters: DEX,
   b) Index-Adressierung (indizierte Adressierung).

2. Die Zahlen von 255 bis 1 werden in absteigender Reihenfolge auf die Speicherplätze 10 bis 264 geschrieben.

3. a) 4A340h
   b) 4B240h
   c) rechnerisch: 1091A0h; da der Adressbus nur 20 Bit breit ist wird die Adresse 091A0h ausgegeben (wrap around).

4. DS-Segment: B8000h – C7FFFh, ES-Segment: C4000 – D3FFFh, Überlapp: C4000 – C7FFFh = 4000h Adressen = 16 Kbyte = 25% der Segmentgröße.

5. FFFF0h, nicht-flüchtiger Speicher.

6. Auf den Speicherplätzen 24d – 27d.

7. Vermeidet Wartezeiten beim Zugriff auf den Maschinencode.

8. Adress-Latches speichern die Adresse während des Buszyklus, weil der 8086 einen gemultiplexten Adress-Datenbus hat. Der Buscontroller erzeugt die Signale des Steuerbusses.

Literaturverzeichnis

[1] Hilleringmann, U.: *Silizium-Halbleitertechnologie*, B.G.Teubner, Stuttgart, 1996

[2] Kobitzsch, W.: *Mikroprozessoren, Aufbau und Wirkungsweise*, R.Oldenbourg Verlag, München, Wien, 1980

[3] Wüst, K.: *Mikroprozessortechnik*, Vieweg, Wiesbaden, 2006