4. Grundlagen der Mikroprogrammsteuerung

4.1 Zur Entwicklungsgeschichte

Das Prinzip der Mikroprogrammsteuerung wurde ursprünglich entwickelt, um das Entwerfen der Steuerwerke von Universalprozessoren zu vereinfachen. Die Betrachtungen beginnen deshalb damit, kurz zu beschreiben, wie solche Steuerwerke aufgebaut sind und herkömmlicherweise entworfen werden.

4.1.1 Das Steuerwerk als Entwurfsaufgabe

Das Steuerwerk bestimmt, wie und wozu die Register, Datenwege, Verarbeitungsschaltungen und Speicheranordnungen des Prozessors in jedem einzelnen Maschinentakt genutzt werden. Es ist über Steuer-, Bedingungs- und Zustandssignale mit dem Verarbeitungswerk, den Speichereinrichtungen und den E-A-Schnittstellen verbunden (Abb. 4.1 und 4.2).

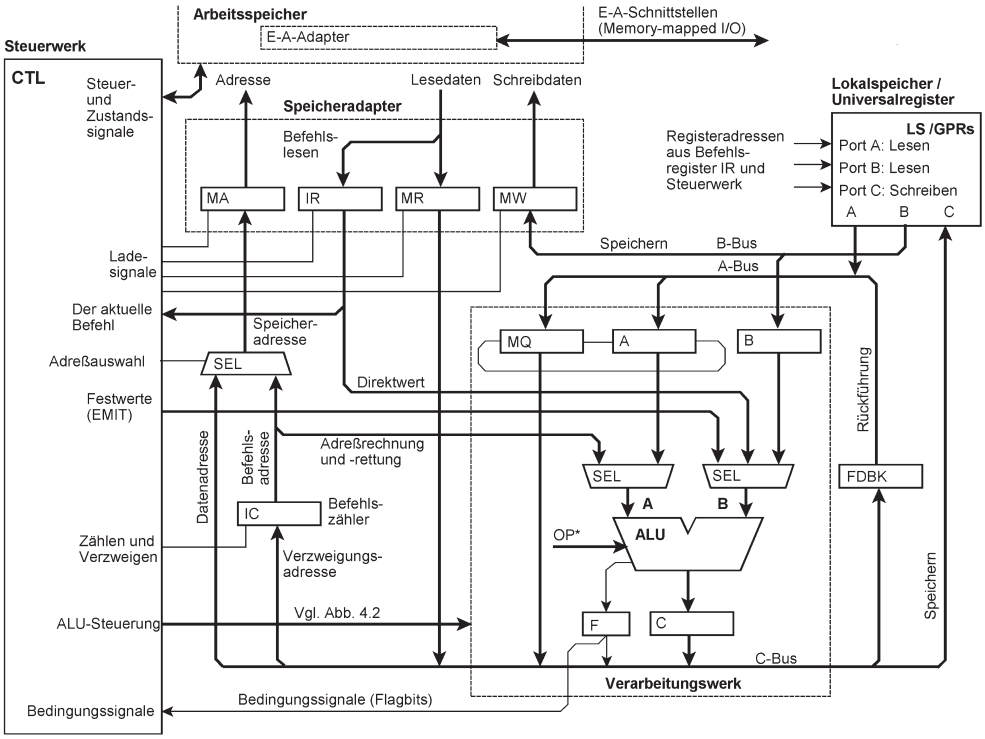


Abb. 4.1 Prozessor mit Steuerwerk. Ein typischer Prozessor mittlerer Komplexität.

Die Maschine hat drei unabhängige Zugriffswege zum Registerspeicher (Triple-Port Access). Sie kann Ergebnisse mit Registeroperanden nach dem Schema <C> := <A> **op** <B> in einem

einzigen Maschinenzyklus bilden und Speicherzugriffe mit Adressen ausführen, die nach dem Schema <Basisadreßregister> + Displacement erzeugt werden, wobei die Adreßrechnung über das Verarbeitungswerk läuft. Die Maschine kann als CISC- oder als RISC-Prozessor ausgelegt werden[1]. Das ist eine Frage der Befehlsformate und der Befehlsablaufsteuerung.
Abb. 4.2 veranschaulicht die Steuerung des Verarbeitungswerks. Jedes der LD-Steuersignale bewirkt, daß das betreffende Register mit den Daten geladen wird, die vom A-Bus, vom B-Bus oder von der ALU geliefert werden, jedes der ST-Steuersignale bewirkt, daß das betreffende Register auf den C-Bus (Ergebnisbus) aufgeschaltet wird. Die Auswahlsignale ASEL und BSEL wählen die Operanden an den Eingängen der ALU aus, die SHIFT-Signale steuern das bitweise Verschieben in den Registern MQ und A, die OPSEL-Signale (Operation Select) bestimmen, welche Operation die Verarbeitungs- und Rechenschaltungen der ALU ausführen. Das Steuerwerk empfängt die Bedingungssignale der ALU, die sich aus der jeweils ausgeführten Operation ergeben (FLAGS = Flagbits im F-Register).

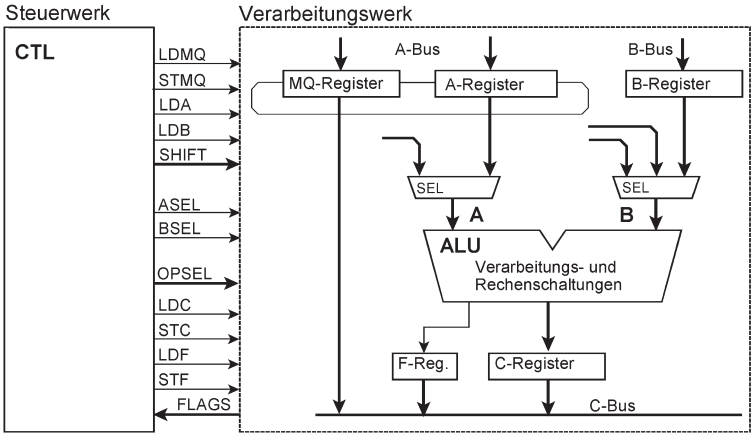


Abb. 4.2 Die Steuersignale der Verarbeitungseinheit.

Die Steuersignale werden vom Steuerwerk erregt. Zu diesen Signalen gehören:

- Übernahme-Erlaubnissignale (Clock Enables),
- Aufschalt-Erlaubnissignale (Output Enables),
- Auswahlsignale,
- Adreßsignale,
- Steuerimpulse (Strobes).

1. In RISC-Architekturen finden in den Operationsbefehlen keine Speicherzugriffe statt und in den Speicherzugriffsbefehlen keine Operandenverknüpfungen (Load-Store-Architektur). Deshalb kann das Verarbeitungswerk zur Adreßrechnung verwendet werden, ohne daß dies zusätzliche Maschinenzyklen erfordert, also die Verarbeitungsleistung vermindert.

Bedingungs- und Zustandssignale sind Eingänge des Steuerwerks. Hierzu gehören:

- Typ-, Längen- und Operationscodes, Bits der Adressierungssteuerung usw. aus dem Befehlsregister,
- die Bedingungssignale der Verarbeitungseinheit (Flagbits),
- Zustandssignale der Schnittstellen (Speicher und E-A),
- Fehlersignale von Überwachungsschaltungen.

Grundabläufe der Prozessorsteuerung betreffen vor allem

- das Adressieren und Holen der Befehle,
- das Holen der Operanden,
- die Verarbeitungsabläufe,
- die internen Transportvorgänge,
- das Speichern der Ergebnisse,
- die Unterbrechungsabläufe,
- das Reagieren im Fehlerfall (Maschinenfehlerbehandlung).

Das Steuerwerk kann als Zustandsautomat entworfen werden, dessen Eingänge die Bedingungs- und Zustandssignale und dessen Ausgänge die Steuersignale sind. Typische Prinzipien der Prozessorsteuerung sind die direkte Steuerung, die sequentielle Steuerung (Folgesteuerung) und die Mikroprogrammsteuerung.

4.1.2 Direkte Steuerung

Alle Steuersignale werden mit Schaltnetzen gebildet (Abb. 4.3). Um ohne Sequencer auszukommen, muß sich die Architektur auf Befehle beschränken, die in einem einzigen Maschinenzyklus ausgeführt werden. Die direkte Steuerung ist somit das typische Steuerprinzip der RISC-Architekturen.

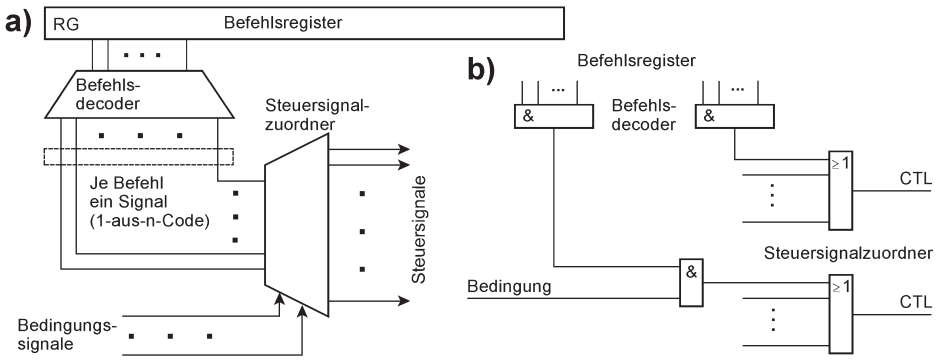


Abb. 4.3 Direkte Steuerung. a) Prinzip, b) Bildung der Steuersignale mit UND-ODER-Verknüpfungen. Der Befehlsdecoder besteht aus UND-Gattern, der Steuersignalzuordner aus ODER-Gattern, denen bedarfsweise UND-Verknüpfungen mit Bedingungssignalen vorgeschaltet sind.

Zu Beginn des Maschinenzyklus wird das Befehlsregister geladen. Danach wird der Befehlszählerinhalt erhöht (Adressierung des Folgebefehls). Am Ende des Zyklus werden die Verknüpfungsresultate gespeichert. Aus technischer Sicht kann es notwendig sein, den Maschinenzyklus in mehrere Taktphasen aufzulösen, beispielsweise für Speicherzugriffe oder Verzweigungen, und die einzelnen Phasen durch Wartezustände (Wait States) zu verlängern.
Die Befehle werden decodiert. Alle Befehlswirkungen liegen dann im 1-aus-n-Code vor. Das ist dem Prinzip nach mit UND-Gattern zu erledigen. Um das Prinzip zu erläutern, zeigt Abb. 4.4 einige einfache Befehlsformate.

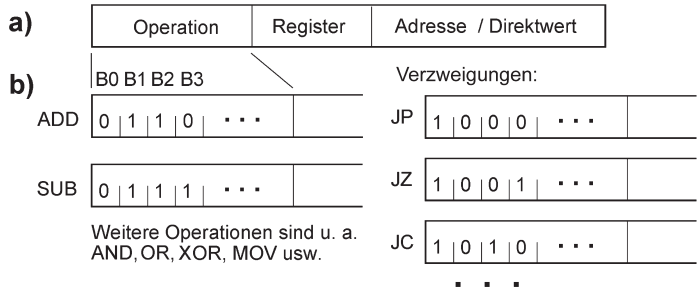


Abb. 4.4 Befehlsbeispiele. Sie dienen nur zur Erläuterung des Prinzips der direkten Steuerung. a) Das grundsätzliche Format mit Feldern, die den Operationscode, Registeradressen sowie eine Speicheradresse oder einen Direktwert enthalten. b) Mehrere Befehle als Beispiele. Die Operationscodes umfassen hier jeweils vier Bits B0 bis B3.

Im Beispiel von Abb. 4.4 werden die Befehle folgendermaßen decodiert:

$$ADD = \overline{B0} \bullet B1 \bullet B2 \bullet \overline{B3} \qquad SUB = \overline{B0} \bullet B1 \bullet B2 \bullet B3 \qquad JP = B0 \bullet \overline{B1} \bullet \overline{B2} \bullet \overline{B3}$$
$$JZ = B0 \bullet \overline{B1} \bullet \overline{B2} \bullet B3 \qquad JC = B0 \bullet \overline{B1} \bullet B2 \bullet \overline{B3} \qquad \text{usw.}$$

Die Steuersignale ergeben sich dann durch ODER-Verknüpfung der Befehle, in denen sie zu erregen sind[2]:

$$LDC = ADD \vee SUB \vee AND \vee OR \vee XOR \vee MOV \text{ usw.}$$

Die Befehlssignale können mit Bedingungssignalen konjunktiv verknüpft werden:

$$LD\_JMP\_ADRS = JP \vee JZ \bullet ZF \vee JC \bullet CF \text{ usw.}$$

JP, JZ, JC usw. sind mit UND-Gattern zu decodieren. Die Decodierung der Befehlsbits (B0, B1 usw.) kann mit dem zugehörigen Bedingungssignal (ZF, CF usw.) in jeweils einem einzigen UND-Gatter zusammengefaßt werden:

$$JZ\_EX = ZF \bullet B0 \bullet \overline{B1} \bullet \overline{B2} \bullet B3; \quad JC\_EX = CF \bullet B0 \bullet \overline{B1} \bullet B2 \bullet \overline{B3} \text{ usw.}$$

2. Genauer: in den Maschinenzyklen, in denen die betreffenden Befehle ausgeführt werden. Es folgen Beispiele, die das Prinzip veranschaulichen sollen. LDC = Laden des C-Registers, LD_JMP_ADRS = Laden der Verzweigungsadresse in den Befehlszähler, JZ_EX = den JZ-Befehl ausführen usw.

Damit ergibt sich ein zweistufiges UND-ODER-Schaltnetz:

LD_JMP_ADRS = JP ∨ JZ_EX ∨ JC_EX usw.

Daß man solche Schaltnetze minimieren kann, versteht sich von selbst.

4.1.3 Sequentielle Steuerung

Dauern die Abläufe länger als jeweils einen einzigen Maschinenzyklus, so müssen sie mit Zustandsautomaten gesteuert werden. Diese Art der Steuerung war traditionell das Gebiet der mehr oder weniger undurchschaubaren Tricks. Getrickst wurde aber nicht nur, um originell zu sein und womöglich Patente anzumelden oder zu umgehen, sondern vor allem, um zu sparen. Heutige Lösungen wird man systematisch entwickeln und sorgfältig strukturieren[3]. Abb. 4.5 veranschaulicht eine naheliegende Struktur mit einem Zustandsautomaten je Ablauf.

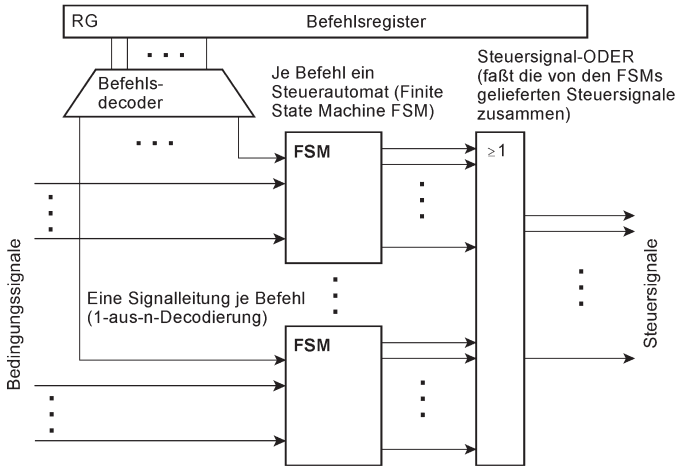


Abb. 4.5 Sequentielle Steuerung. Hier mit jeweils einem Zustandsautomaten (Finite State Machine; FSM) je Befehl oder Sonderablauf (Unterbrechung, Maschinenfehlerbehandlung usw.). Die Zustandsautomaten sind zumeist einfache Sequencer oder Branch Sequencer.

4.1.4 Steuerketten

Die Steuerkette ist die klassische Grundlage des systematischen Entwerfens sequentieller Steuerungen von Prozessoren. Die Zustandsübergänge sind typischerweise so einfach, daß sie von Steuerautomaten ähnlich Abb. 2.20 (S. 44) erledigt werden können. Es ist zumeist ein Weiterschalten zum nächsten Zustand, das manchmal durch eine Wartebedingung aufgehalten werden

3. Sowohl die programmierbaren Schaltkreise als auch die Synthesewerkzeuge sind von Grund auf darauf abgestellt. Wenn man versucht, um jeden Preis Flipflops zu sparen, braucht man womöglich mehr Logikzellen als wenn man gleichsam lehrbuchmäßig entwirft und sich an den jeweiligen Codierrichtlinen (Coding Styles) orientiert. Tricks mit monostabilen Multivibratoren, Laufzeitketten usw. kommen ohnehin nicht mehr in Betracht.

kann, gelegentlich auch ein Verzweigen zu einem anderem Zustand. Die binäre Zustandscodierung erfordert aber einen beträchtlichen Decodieraufwand, um die einzelnen Steuersignale zu gewinnen. Deshalb bevorzugt man die 1-aus-n-Codierung. Da die meisten Zustandsübergänge ein bloßes Weitergeben sind, werden die Zustandsautomaten von Abb. 4.5 zu Schieberegistern, durch die eine einzige Eins läuft (Abb. 4.6 und 4.7). In solchen Steuerketten können auch kompliziertere Zustandsübergänge auftreten, die den Charakter von Verzweigungen haben (Abb. 4.8 und 4.9).

Ein derartiges Zustandsdiagramm ergibt sich praktisch 1:1 aus einem hinreichend genauen Flußdiagramm des jeweiligen Ablaufs, wobei jedem Anweisungskästchen ein Zustand entspricht. In der Schaltung wird für jeden Zustand ein Flipflop vorgesehen. Für jeden Ablauf kann man auf einfache Weise – durch entsprechendes Anschließen der Steuersignale – festlegen, was in den Maschinenzyklen geschieht. Der Steuerungsentwurf entspricht somit schon einer Art Mikroprogrammierung[4]. Wichtig ist, sich die Abläufe auszudenken; der eigentliche Schaltungsentwurf ist dann eine schematische, routinemäßige Angelegenheit.

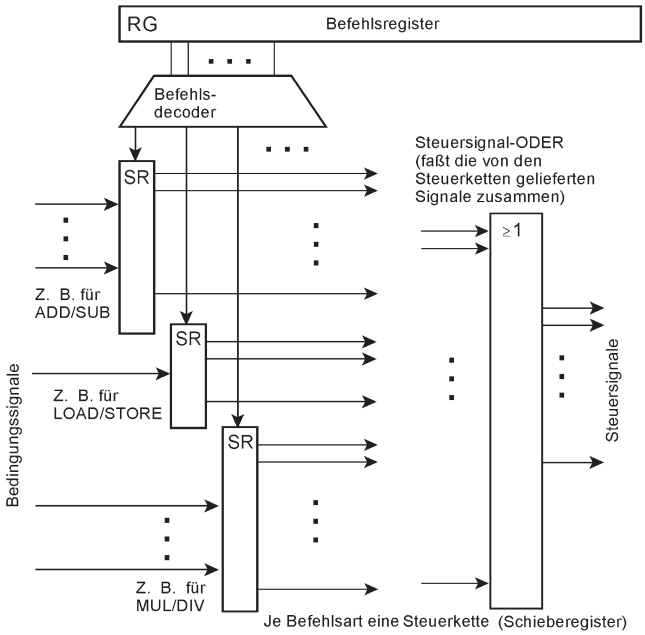


Abb. 4.6 Sequentielle Steuerung mit Steuerketten.

4. Das historisch erste Beispiel einer derartigen Steuerung: Konrad Zuses Z3. Hier wurden – dem damaligen Stand der Technik entsprechend – anstelle der Schieberegister Schrittschaltwerke eingesetzt ([41], [89]). Nur auf Grundlage eines derart systematischen Steuerungsentwurfs war es überhaupt möglich, die Entwurfsaufgaben zu bewältigen. Zu historischen Anwendungsbeispielen von Steuerketten im Sinne einer Mikroprogrammsteuerung siehe beispielsweise [90].

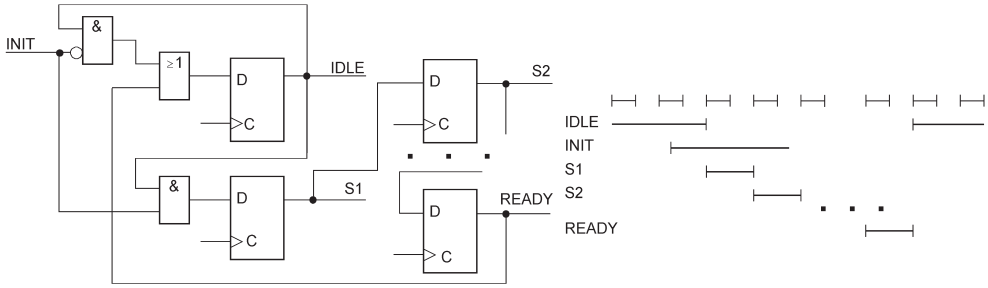


Abb. 4.7 Steuerkette als Schieberegister. Im Beispiel als Zustandsautomat mit der Codierung 1-aus-n (One-Hot Encoding, OHE). Das Startsignal INIT bewirkt, daß der Ruhezustand (IDLE) verlassen wird und eine Eins durch die Schiebekette läuft. Im letzten Zustand wird das Endesignal READY erregt. Daraufhin gelangt die Steuerkette wieder in den Ruhezustand. Einzelheiten (Takt, Anfangsrücksetzen usw.) weggelassen. Der Anfangszustand: IDLE = 1, alle anderen Flipflops = 0.

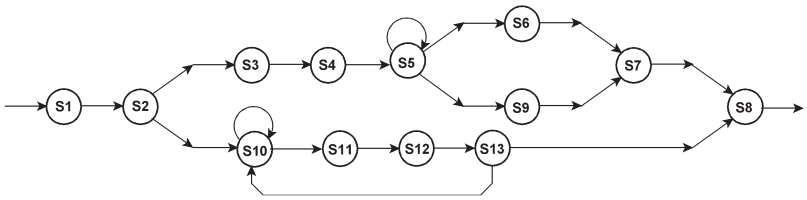


Abb. 4.8 Zustandsdiagramm einer Steuerkette. Ein fiktives Beispiel, in dem Wartezustände auftreten und alternative Zweige durchlaufen werden.

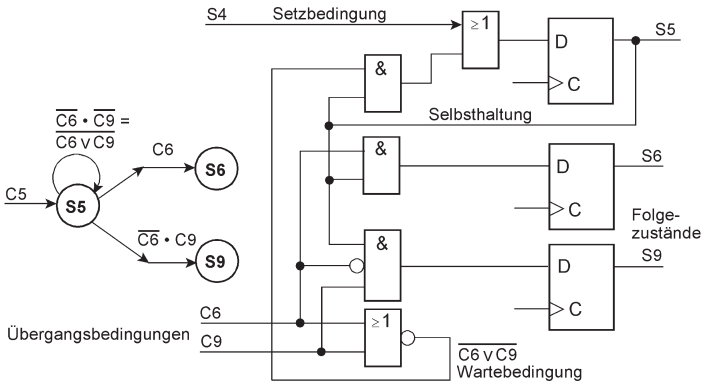


Abb. 4.9 Die Implementierung eines komplizierteren Zustandsübergangs. Beispiel: die Zustände S5, S6, S9 in Abb. 4.8. Auslegung als OHE-Zustandsautomat ähnlich Abb. 4.7. Verzweigung in zwei Richtungen mit Vorrangregelung und Warten. Der vorhergehende Zustand (S4) bewirkt, daß S5 aktiv wird. Die Bedingungen (Conditions) C6 und C9 veranlassen den Übergang in den jeweiligen Folgezustand. Im Beispiel hat C6 Vorrang vor C9. Die Zustände S6 und S9 dauern jeweils nur einen einzigen Taktzyklus. Deshalb brauchen diese Flipflops keine Selbsthaltung.

4.1.5 Mikroprogrammsteuerung

Es ist nicht grundsätzlich schwierig, Steuerschaltungen auf Grundlage von Steuerketten zu entwerfen. Sind aber umfangreiche Befehlslisten und komplizierte Abläufe zu implementieren (CISC-Architekturen), kann der Aufwand extrem hoch werden. 200 Befehle erfordern auf den ersten Blick 200 Steuerketten. Nun wird man Abläufe zusammenfassen und anderweitig tricksen (Entwurfsoptimierung). Muß alles von Hand entworfen werden, ist mit vielen Fehlern zu rechnen. Die fertige Schaltung läßt sich nicht mehr ohne weiteres ändern, um Fehler zu beseitigen, Befehlsabläufe zu verbessern und neue Befehle hinzuzufügen.

Der entscheidende Gedanke

Das Steuerwerk muß in jedem Maschinenzyklus Steuersignale liefern und Bedingungs- oder Zustandssignale auswerten. Die Bitmuster der Steuersignale werden nicht von Schaltnetzen gebildet, sondern in einem Festwertspeicher untergebracht, der in jedem Maschinenzyklus adressiert und gelesen wird (Abb. 4.10). Die gespeicherten Wörter werden als Mikrobefehle bezeichnet. Deren grundsätzliches Format besteht aus zwei Teilen. Die Bits des ersten Teil dienen zum Erregen der Steuersignale, die des zweiten zum Auswählen des nachfolgenden Mikrobefehls. In die Folgeadresse können Bedingungs- und Zustandssignale einfließen.

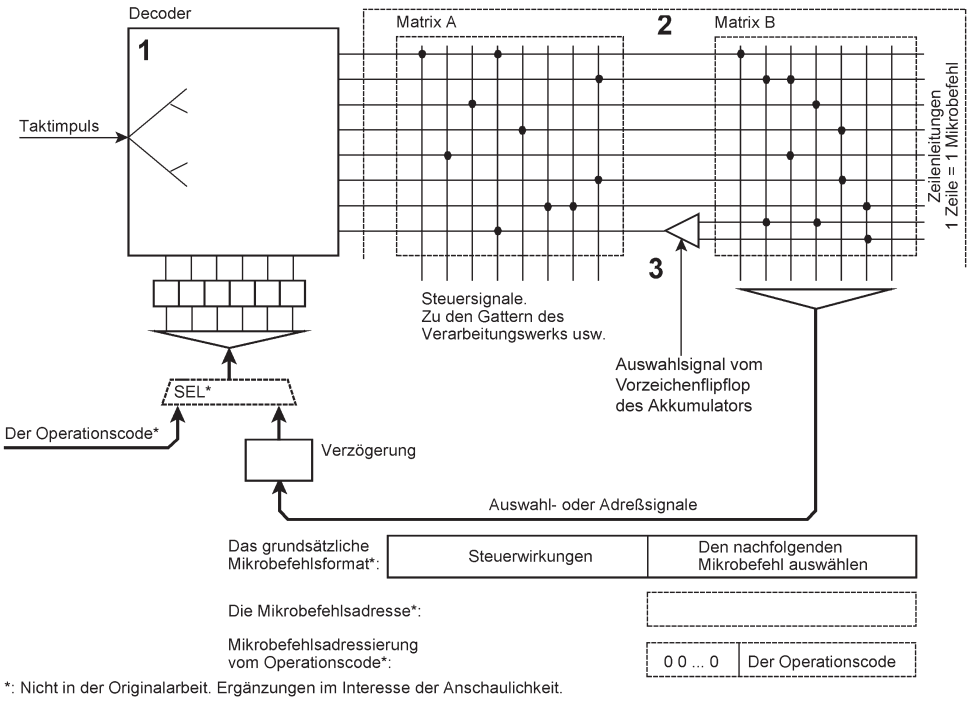


Abb. 4.10 Das Prinzip der Mikroprogrammsteuerung, wie es in der Pionierarbeit von Maurice M. Wilkes skizziert wurde ([91] bis [93]).

Abb. 4.10 wurde aus dem geradezu legendären Blockschaltbild in der Pionierarbeit von Wilkes abgeleitet. Der Decoder 1 adressiert den Festwertspeicher 2, der aus zwei Matrizen A, B aufgebaut ist. Man denke beispielsweise an Diodenmatrizen. Die Matrix A enthält die Bitmuster der Steuersignale, die Matrix B die Folgeadressen. Ist ein Befehl auszuführen, wird zunächst der Operationscode an den Decoder 1 angelegt. Der erste Mikrobefehl eines Befehlsablaufs wird somit vom Operationscode adressiert. Wenn der Taktimpuls des Maschinenzyklus den Decoder 1 durchläuft, erregt er die zugehörige Zeilenleitung des Festwertspeichers 2. Die Matrix A liefert somit die ersten Steuersignale. Aus der Matrix B kommt die Adresse des nächsten Mikrobefehls, die zum Decoder 1 zurückgeführt wird. Es ist möglich, Bedingungs- und Zustandssignale auf die Adreßbildung einwirken zu lassen. In Abb. 4.10 ist das anhand eines Beispiels gezeigt. Hier sind einer Zeilenleitung der Matrix A über eine Auswahlstufe 3 zwei Zeilenleitungen der Matrix B nachgeordnet. Auf diese Weise werden zwei alternative Folgeadressen gespeichert. Die Auswahlstufe 3 wird hier vom Vorzeichenflipflop des Akkumulators gesteuert. Ist das Vorzeichen beispielsweise positiv, wird der nächste Mikrobefehl von der ersten der beiden Folgeadressen gelesen, ist das Vorzeichen negativ, von der zweiten.
Es hat einige Zeit gedauert, bis das Prinzip in Maschinen eingesetzt wurde, die in Serie gefertigt wurden. Am Anfang war vor allem das Problem zu lösen, kostengünstige Mikroprogrammspeicher zu schaffen. Wenn man die Mikrobefehle mit Dioden, Ferritkernen oder Kondensatoren speichern muß, wird es teuer und trägt auf. Die Kritik war vor allem auf den Grundaufwand gerichtet, aber auch auf die Leistungsschwächen. Wenn man von der Architekturentwicklung an alles aufeinander abstimmt und optimiert – die Befehlswirkungen nicht zu komplex, die Schaltungen auf Geschwindigkeit ausgelegt usw. – ist eine Maschine mit sequentieller Steuerung kostengünstiger und schneller als eine mit Mikroprogrammsteuerung.

Komplexe Architekturen, komplexe Maschinenbefehle

Diese Entwicklungstendenzen haben die Mikroprogrammsteuerung zeitweilig zur Vorzugslösung werden lassen. Am Anfang stand der Gedanke, nicht mehr spezielle Maschinenarchitekturen für bestimmte Anwendungsgebiete zu entwickeln – zum Geld zählen (soll heißen: für das Bankwesen, die Buchhaltung usw.), zum wissenschaftlichen Rechnen, zum Steuern von Produktionsanlagen usw. – sondern Maschinen, die alles können. Es wurde überwiegend mit Maschinenbefehlen programmiert (Assemblerprogrammierung). Die Maschinenbefehle sollten leistungsfähig, flexibel und bequem nutzbar sein. Das führte zu Architekturen mit weit über 100 Befehlstypen, vielen Adressierungsarten und Ablaufvarianten. Mit sequentiellen Steuerwerken war das nicht mehr zu schaffen, zumindest nicht in kleineren und mittleren Maschinen. Deshalb hat man mit viel Entwicklungsaufwand das Prinzip der Mikroprogrammsteuerung praktisch anwendbar gemacht. Die legendäre Pionierleistung in dieser Hinsicht war das System /360 vom IBM. Die Typenbezeichnung war eine Anspielung auf den Vollkreis mit seinen 360°. Hiermit sollte die total universelle Nutzbarkeit der Architektur veranschaulicht werden. In dieser Architekturfamilie wurde die Mikroprogrammsteuerung nicht nur in den Prozessoren, sondern auch in den komplexeren peripheren Steuereinheiten eingesetzt. Rechnerarchitekturen mit komplizierten Wirkprinzipien kann man praktisch nur auf Grundlage der Mikroprogrammsteuerung

kostengünstig implementieren, weil sie auf reguläre Strukturen führt und somit die Kompliziertheit beherrschbar macht. Historische Beispiele sind die Systemfamilie S/360 und deren Nachfolger, die VAX-Maschinen der Fa. Digital Equipment Corporation (DEC) und die 68000er Mikroprozessoren von Motorola, aber auch die Prozessoren der Personalcomputer.

Grundbegriffe

Es ist nichts standardisiert. In Lehrwerken und Handbüchern findet man unterschiedliche Bezeichnungen für im Grunde gleichartige Funktionseinheiten, Mikrobefehlswirkungen und Abläufe. Unsere Begriffe (Tabelle 4.1) sind teil generische allgemeine Bezeichnungen, teils orientieren sie sich an Pionierleistungen der Entwicklungsgeschichte. Das betrifft vor allem Fachbegriffe, die mit der Einführung des Systems /360 aufgekommen sind[5].

Unsere Bezeichnungen	Typische Abkürzungen und Begriffe der englischen Fachsprache
Mikroprogrammspeicher, Steuerspeicher	CS = Control Storage
Mikrobefehlsregister	CSDR = Control Storage Data Register
Mikrobefehlsadreßregister	CSAR = Control Storage Address Register
Mikrobefehl, Steuerwort	Microinstruction, Control Word
Mikroanweisung	Micro-Order, Micro-Command
Voreinstellregister und -flipflops	Staticizers (STATs)
Mikrobefehlsadressierung (Funktionseinheit)	Microinstruction Address Sequencer
Befehlsregister	IR = Instruction Register
Befehlszähler	IC = Instruction Counter
Direktwertfeld, Emit-Feld	Emit-Field, Literal, Immediate

Tabelle 4.1 Grundbegriffe im Bereich der Mikroprogrammsteuerwerke (eine kleine Auswahl).

Die Mikroprogrammsteuerung als Vorzugslösung

Die Prinzipien der Mikroprogrammierung wurden nach und nach zum allgemeinen Fachwissen. Als dann kostengünstige Speicherschaltkreise verfügbar waren, wurden viele Prozessorkerne, Gerätesteuereinheiten usw. mit Mikroprogrammsteuerung implementiert. Das vor allem deshalb, um mit der Hardware schnell fertig zu werden. Die Schaltungslösung ist vergleichsweise einfach; die eigentliche Komplexität des Anwendungsproblems wird ins Mikroprogramm verlagert. Was zunehmend wichtig wurde, war das Ändern beim Kunden. IBM hatte die ersten Disketten nur zu diesem Zweck eingeführt[6]. Man schickt keinen Techniker mehr zum Kunden,

5. Das vor allem deshalb, weil seinerzeit viele Grundsatzlösungen der Mikroprogrammsteuerung geschaffen wurden, die sich über Jahrzehnte bewährt haben. Prinzip: einfacher und zweckmäßiger geht's nicht ... Manchen Konkurrenten blieb seinerzeit nichts anders übrig als zu warten, bis die Grundsatzpatente abgelaufen waren. In der Zwischenzeit mußten sie sich mit Umgehungslösungen behelfen ...
6. Erst später haben sie die Lochkarten in der Datenerfassung abgelöst, noch später wurden sie die ersten Massenspeicher der "persönlichen" Computer.

sondern versendet eine neue Diskette mit der Post[7]. Mit den heutigen Mikrocontrollern und RISC-Prozessoren ist es eigentlich nicht anders, es ist der gleiche Grundgedanke der Problemlösung mittels Software. Nicht entwerfen (schwierig, will verstanden sein), sondern programmieren, also so lange hacken, bis man sich damit halbwegs sehen lassen kann ... Im Gegensatz zu den herkömmlichen Mikroprogrammsteuerwerken sind die heutigen Prozessoren viel bequemer zu programmieren[8], trotz aller Exzentrizitäten ihrer Wirkprinzipien. Sie weisen aber keineswegs das starre, exakte Zeitverhalten auf – bis auf den Maschinentakt genau – das man mit einer vernünftig entworfenen Mikroprogrammsteuerung ohne weitere erreichen kann.

4.2 Der Computer im Computer

Das typische Mikroprogrammsteuerwerk umfaßt einen Mikroprogramm- oder Steuerspeicher, ein Mikrobefehlsregister, ein Mikrobefehlsadreßregister sowie Schaltmittel zur Mikrobefehlsadressierung und zur Ablaufsteuerung. Die jeweils auszuführenden Mikrobefehle werden aus dem Steuerspeicher in das Mikrobefehlsregister geladen. Dieses Register wirkt direkt auf die zu steuernden Schaltungen. Abb. 4.11 zeigt ein grundsätzliches Blockschaltbild. Es verwendet Symbole, die allgemein üblich sind, entspricht aber im Grunde der ursprünglichen Darstellung von Maurice M.Wilkes.

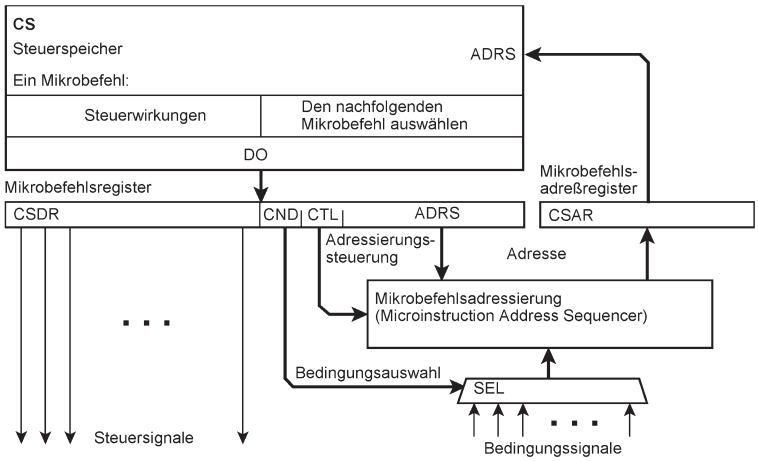


Abb. 4.11 Das Mikroprogrammsteuerwerk im grundsätzlichen Blockschaltbild.

Zu jedem Maschinenbefehl gehört ein Mikroprogramm. Zudem gibt es Mikroprogramme für das Befehlslesen, die Unterbrechungsauslösung, die Maschinenfehlerbehandlung usw. Die typische Mikroprogrammschleife der Befehlsausführung holt den Befehl ins Befehlsregister und

7. Das ursprüngliche Diskettenformat (8") wurde eigens so festgelegt, daß die Diskette in einen Briefumschlag (nach US-Norm) paßt.
8. Und beim Kunden zu ändern – ganz ohne Postversand und nicht selten auch so, daß er es gar nicht merkt ...

führt mit dessen Operationscode eine Funktionsverzweigung zu dem Mikroprogramm aus, das den eigentlichen Befehlsablauf im einzelnen steuert. Man spricht davon, daß das Mikroprogramm die Zielarchitektur (Target Architecture) bzw. Zielmaschine emuliert. Der typische Mikrobefehl steuert, was in einem Zyklus des Maschinentaktes abläuft.
Das Mikroprogrammsteuerwerk kann als ein einfacher Computer aufgefaßt werden, der in einen komplizierteren Computer eingebaut ist (Abb. 4.12). Es soll vor allem Befehlsabläufe steuern. Der aktuelle Befehl steht in einem Befehlsregister (Instruction Register IR). Es wird beim Befehlslesen geladen. Während der Befehlsausführung liefert es die Parameter des Befehlsablaufs. Das Mikroprogrammsteuerwerk entnimmt daraus den Operationscode, die Adressen oder Direktwerte, die Adressierungsarten usw.

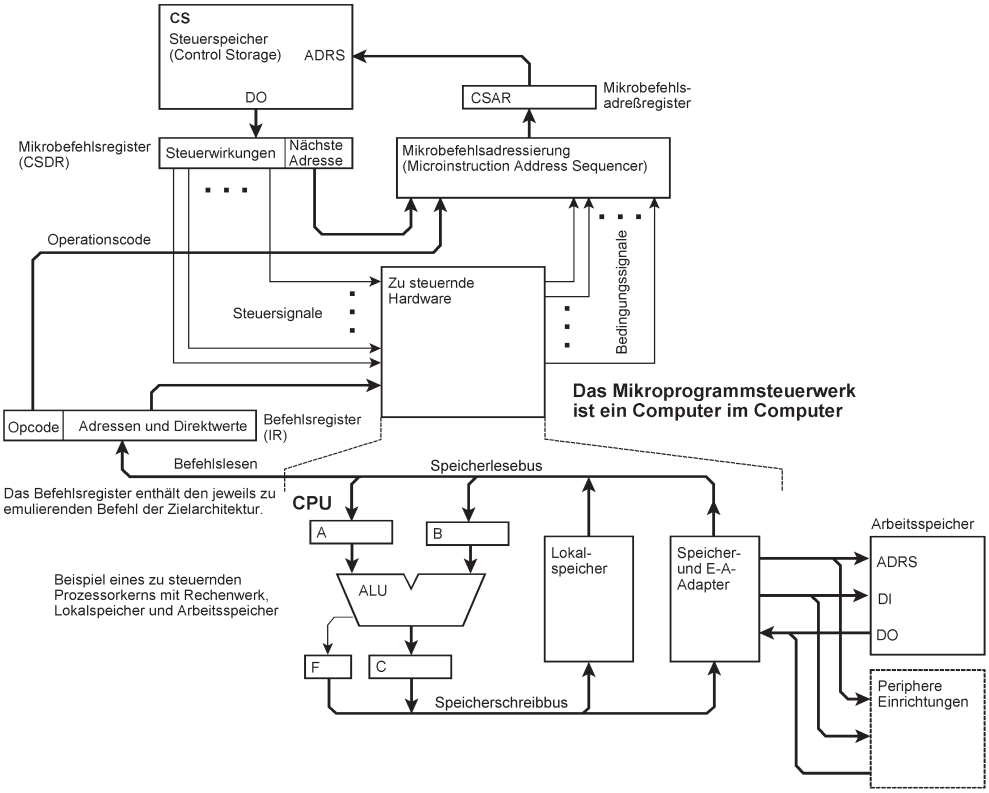


Abb. 4.12 Das Mikroprogrammsteuerwerk als Computer im Computer.

Wir wollen den Grundgedanken des einfachen Computers aufgreifen und mit zeitgemäßen Mitteln implementieren. Eine solchen Computer kann man – wie soeben beschrieben – als Steuerwerk in eine kompliziertere Maschine einbauen. Es liegt aber auch nahe, ihn von vornherein als Plattform zu verwenden, um Anwendungsprobleme kostengünstig und effektiv zu lösen. Als

autonome Plattform ist eine solche Maschine schneller und braucht weniger Ressourcen[9], als zentrales Steuerwerk bietet sie die Vorteile, die aus der Entwicklungsgeschichte bekannt sind. Vor allem macht sie die Komplexität beherrschbar. Somit wird es möglich, leistungsfähige Universal- und Spezialmaschinen selbst zu entwickeln. Mit modernen Entwicklungssystemen und programmierbaren Schaltkreisen muß das keine Utopie bleiben, auch bei knappen Zeit- und Kostenvorgaben. Der Grundgedanke: Teile und Herrsche. Eine Art universeller, programmierbarer Plattform mit angeschlossenen Funktionseinheiten, die jeweils einzeln so überschaubar sind, daß man sie mittels Verhaltensbeschreibung entwerfen kann. Ob man eine komplexe Anwendungslösung mit fertig gekauften IP Cores aufbaut, auf die hier skizzierte Weise selbst entwickelt oder beides kombiniert (Kapitel 7), ist fallweise zu entscheiden.

Computer oder Operationsautomat?

Wenn das Mikroprogrammsteuerwerk ein Computer im Computer ist, so arbeitet es auch wie ein Computer als autonomer speicherprogrammierter Automat. Die Mikrobefehle entsprechen den Befehlen. Dem Kreislauf der Befehlsadressierung[10] entspricht ein Kreislauf der Mikrobefehlsadressierung. Beides beruht auf den gleichen Grundsatzlösungen. Auch die Mikroprogramme laufen in Endlosschleifen. Die Mikrobefehlsadressen stammen aus dem Mikrobefehl selbst, aus anderen Einrichtungen (indirekte und Funktionsverzweigung), oder sie werden durch Adreßzählung gewonnen. Maschinenbefehle sollen eine bequem nutzbare Programmschnittstelle (API) der Anwendungs- und Systemprogrammierung darstellen, Mikrobefehle die Verarbeitungs- und E-A-Schaltungen effektiv steuern.

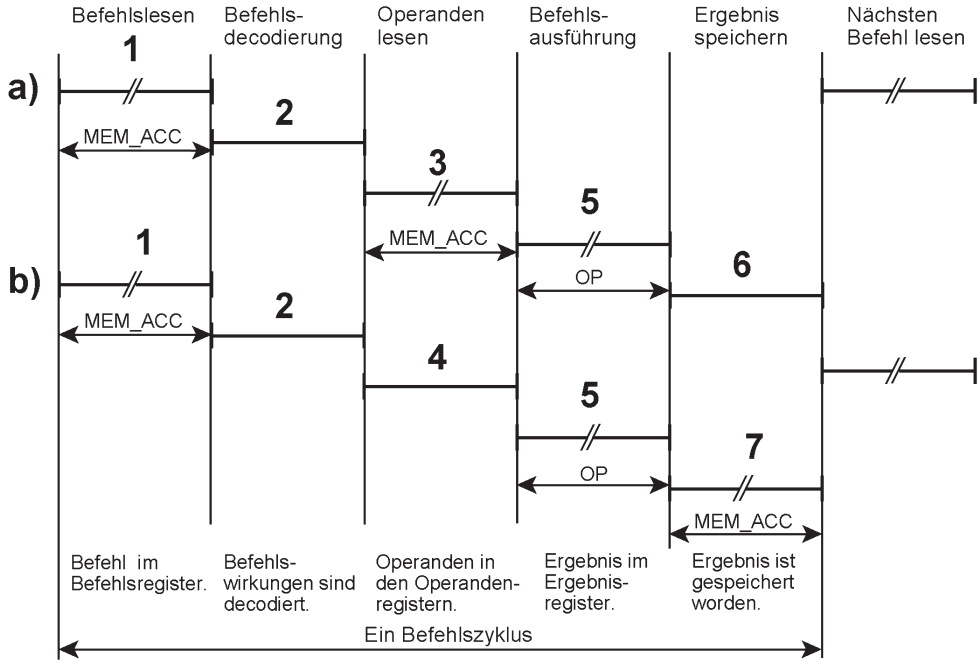
Mikrobefehle im Operationsautomaten

Ein Mikroprogramm wird gestartet und steuert ein Ablauf der Verarbeitung oder der Ein- und Ausgabe. Was aber geschieht, wenn es seine Arbeit beendet hat? In einem autonomen Automaten muß es in eine Steuerschleife zurückkehren; die Maschine muß schließlich weiterlaufen. Nun liegt der Gedanke nahe, die einzelnen Funktionseinheiten mit lokalen Mikroprogrammsteuerwerken auszurüsten. Deren Mikroprogramme werden gestartet, erledigen ihre Arbeit und halten an, wenn sie fertig sind (Start-Stop-Betrieb). Die theoretische Verallgemeinerung dieses Prinzips ist der universelle algorithmische Automat[11]. Die Funktionseinheiten werden zu Operationsautomaten, die ihre Mikrobefehle vom Steuerautomaten erhalten. Wir konzentrieren uns aber vor allem auf den Computer im Computer, denn das autonome Mikroprogrammsteuerwerk ist die anspruchsvollere Entwurfsaufgabe. Das Mikroprogrammsteuerwerk im Operationsautomaten – das von außen gestartet wird und anhält, wenn es seine Aufgabe erledigt hat – ist nur ein Spezialfall. Es ist nicht schwierig, ein autonomes Mikroprogrammsteuerwerk auf den Start-Stop-Betrieb umzubauen.

9. Verglichen mit üblichen Lösungen auf Grundlage fertiger Mikrocontroller oder RISC-Prozessoren. Wenn nichts zu rechnen ist, brauchen wir auch kein Rechenwerk. Wenn wir eines brauchen, können wir es so leistungsfähig und ggf. so spezialisiert auslegen, wie es jeweils zweckmäßig ist.
10. Vgl. die Abb. 3.4 und 3.5 (S. 114f).
11. Vgl. Abschnitt 2.4 (S. 86ff).

Mikrobefehlszyklen, Arbeitsspeicherzyklen, Befehlsphasen

Der Mikrobefehlszyklus ist der kürzeste der hier genannten Zyklen. Mikrobefehle steuern die Befehlsphasen[12] und die Arbeitsspeicherzugriffe. In neueren Architekturen gibt es in den meisten Befehlen höchstens zwei Speicherzugriffe, einer zum Lesen des Befehls selbst und einer zum Lesen eines Operanden oder zum Schreiben eines Ergebnisses (Abb. 4.13).



1 Befehlslesen (Instruction Fetch). Arbeitsspeicherzugriff
2 Befehlsdecodierung (Instruction Decode). Decodierschaltungen oder Funktionsverzweigung mit dem Operationscode
3 Arbeitsspeicherzugriff zum Lesen des Operanden (Operand Fetch)
4 Interner Registerzugriff zum Lesen des Operanden (z. B. aus dem Universalregistersatz)
5 Operationsausführung (Operate). Dauer hängt davon ab, wie kompliziert die Operation ist und wie die Operationswerke beschaffen sind
6 Ergebnis intern speichern (z. B. im Universalregistersatz)
7 Arbeitsspeicherzugriff zum Speichern des Ergebnisses (Result Store)

Abb. 4.13 Typische Befehle mit zwei Arbeitsspeicherzugriffen. a) Der Operand wird aus dem Arbeitsspeicher gelesen, das Ergebnis wird in einem Register gespeichert (z. B. Lade- oder Rechenbefehl). b) Der Operand kommt aus einem Register, das Ergebnis wird in den Arbeitsspeicher geschrieben(z. B. Speicherbefehl).

12. Vgl. auch Abb. 3.35, S. 156.

Die Frage ist, wie lange solche Befehlsphasen dauern, wieviele Mikrobefehlszyklen nötig sind, um sie zu steuern. Ein gleichsam klassischer Richtwert besagt, daß in einem Arbeitsspeicherzyklus 5 bis 10 Mikrobefehle ausgeführt werden müssen, wenn die Maschine etwas Vernünftiges leisten soll[13]. Also Zykluszeit des Mikroprogrammspeichers = $^1/_5$ bis $^1/_{10}$ der Zykluszeit des Arbeitsspeichers. Daraus folgt, daß extrem schnelle Mikroprogrammspeicher Voraussetzung sind, wenn man eine leistungsfähige Maschine mit Mikroprogrammsteuerung bauen möchte. Hätte man solche Speicher nicht, würde nur eine übermäßig aufwendige[14], aber langsame Maschine herauskommen. Eine sequentielle Steuerung wäre dann besser.
Diese Richtwerte betreffen aber nur eine bestimmte Art von Maschinenarchitekturen, namentlich Minicomputer mit komplexen Befehlssätzen (CISC). Die Befehlsabläufe sind kompliziert, der Aufwand soll aber gering bleiben. Deshalb steuert das Mikroprogramm alles. Keine aufwendigen Adreßrechenwerke, kein Pipelining, keine Caches usw. Wenn alles in 16-Bit-Abschnitten über eine einzige ALU laufen muß, erfordert allein schon die Adreßrechnung der Operandenzugriffe mehrere Mikrobefehle.
Gibt es aber keine solchen Einschränkungen, braucht man in den einzelnen Befehlsphasen viel weniger Mikrobefehle. Wenn man die Maschine entsprechend auslegt, kommt man womöglich mit einem einzigen aus: ein Mikrobefehlszyklus = ein Maschinentakt = eine Befehlsphase[15]. Der Mikrobefehl ist dann nur eine Art Steuerwort, das Register, Speicherzellen usw. auswählt und Signalwege schaltet. Der Mikrobefehl stellt die Signalwege nur ein, dann läuft alles sozusagen von selbst durch[16]. Es ist eine Art Datenflußprinzip im kleinen. Die ersten Befehlsphasen (Befehl lesen, Befehl decodieren, Operanden lesen) sind vergleichsweise einfache Abläufe. In den Pipeline-Stufen der Hochleistungsprozessoren werden sie üblicherweise sequentiell gesteuert. Das Mikroprogrammsteuerwerk kommt lediglich in der Ausführungsphase zur Wirkung.
Wenn aber bereits in den ersten Befehlsphasen kompliziertere Abläufe vorkommen[17], könnte man auch an Mikroprogrammsteuerwerke in den entsprechenden Pipelinestufen denken. Eine Sparlösung besteht darin, durch Befehlsdecodierung zwischen einfachen und komplizierten Abläufen zu unterscheiden. Die einfachen Abläufe werden dann in den Pipelinestufen sequentiell gesteuert, für die komplizierten wird das Pipelining ausgesetzt und das zentrale Mikroprogrammsteuerwerk zu Hilfe gerufen.

13. Quelle: [94].
14. Wegen der Grundaufwendungen eines Computers im Computer. Die Leiterplatten, Schaltkreise, Logikzellen usw. des Mikroprogrammsteuerwerks steuern die Zielmaschine nicht direkt, sondern sind nur eine Plattform zum Ausführen der Mikroprogramme.
15. Wenn alles glatt durchläuft (keine Wartezustände, keine Abhängigkeiten, keine Konflikte in der Pipeline).
16. Wirklich neu ist das nicht. Einige der ersten minimalen Maschinen, wie die Zuse Z22 und die Stantec Zebra, haben ebenso gearbeitet, nur bitseriell und viel langsamer. Vgl. [56] für eine überblicksmäßige Kurzbeschreibung.
17. Eine Frage der Prozessorarchitektur. Komplizierte Abläufe können sich beispielsweise dann ergeben, wenn man von den "reduzierten" Befehlswirkungen abgeht und Prinzipien der Objektorientierung von Grund auf in der Hardware unterstützt (als historisches Beispiel vgl. Intels iAPX432).

Die Mikroprogrammsteuerung eines universellen Prozessors

Um ein überschaubares Beispiel zu geben, darf der Prozessor nicht zu kompliziert sein. Abb. 4.14 zeigt einen Prozessorkern, der gegenüber dem von Abb. 4.1 erheblich vereinfacht ist. Nur ein einziger Zugriffsweg zum Lokalspeicher, nur zwei Bussysteme. Der Befehlszähler ist ein Register im Lokalspeicher. Abb 4.15 veranschaulicht ein (horizontales) Mikrobefehlsformat für diese Maschine[18].

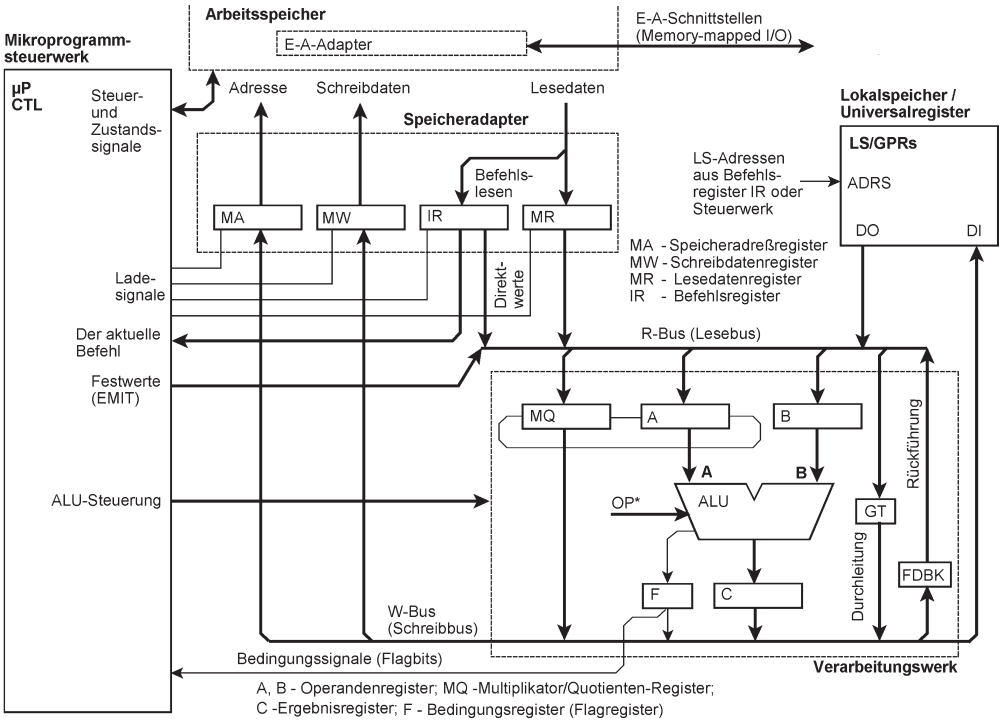


Abb. 4.14 Eine vereinfachte Maschine (Sparlösung) als Demonstrationsbeispiel. Wenn es nicht auf extremes Leistungsvermögen ankommt, ist es eine durchaus brauchbare Plattform für CISC-Architekturen.

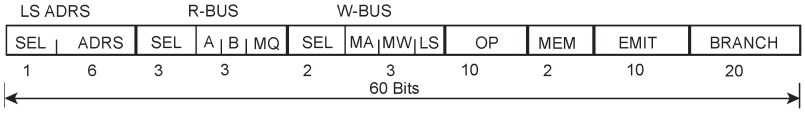


Abb. 4.15 Ein horizontales Mikrobefehlsformat. Es ergibt sich durch Aneinanderreihen von Bitfeldern, die zum Auswählen, Steuern, Verzweigen und als Direktwerte benötigt werden.

18. Abb. 4.15 soll nur dazu dienen, eine erste Vorstellung davon zu vermitteln, wie solche Mikrobefehlsformate aufgebaut sind. Näheres in Kapitel 6. Dort werden die Beispiele der Abb. 4.1 und 4.14 gründlicher diskutiert werden.

Direkte, sequentielle oder und Mikroprogrammsteuerung?
Vergleichende Betrachtungen müssen sich auf vergleichbare Größenordnungen der Aufwendungen beziehen:

1. Kostengünstige Prozessoren
Weil die Verarbeitungsschaltungen, Datenwege usw. sparsam ausgelegt sind, müssen viele Befehle in mehreren Taktzyklen ausgeführt werden. Das erfordert entsprechenden Aufwand in den Steuerschaltungen. Architekturen mit auch nur halbwegs komplizierten Wirkprinzipien (hierzu gehört u. a. bereits Intels 8086) kann man nur auf Grundlage der Mikroprogrammsteuerung kostengünstig implementieren.

2. Hochleistungsprozessoren
Möglichst alle Wirkungen eines Befehls sollen in einem einzigen Taktzyklus erbracht werden. Das Problem ist hier weniger der Aufwand, sondern die interne Kompliziertheit. Man bevorzugt die Mikroprogrammsteuerung vor allem deshalb, weil sie auf reguläre Strukturen führt und somit die Kompliziertheit beherrschbar macht.

Sequentielle oder Mikroprogrammsteuerung – Gatter oder Speicher?

- In einer direkten oder sequentiellen Steuerung werden die Steuersignale mit kombinatorischen Netzwerken gebildet, die aus Gattern aufgebaut sind (hart verdrahtete Steuerung). In einer Mikroprogrammsteuerung werden die Bitmuster der Steuersignale aus dem Steuerspeicher abgerufen.
- Im Speicher müssen alle Adreßbitkombinationen decodiert werden. n Adreßbits adressieren 2^n Speicherzellen. Die Schaltungstiefe der Adreßdecodierung liegt damit fest, unabhängig davon, was gespeichert wird. Die Schaltungstiefe eines Gatternetzwerks ist deutlich geringer, weil nur jene Implikanden decodiert werden, die in der jeweiligen Booleschen Funktion enthalten sind. Deshalb ist die Zugriffszeit einer Speicheranordnung typischerweise größer als die Verzögerungszeit eines Gatternetzwerks.
- Setzt man die gleiche Schaltkreistechnologie voraus, so könnte man eine direkt oder sequentiell gesteuerte Maschine mit einem schnelleren Takt betreiben als eine mikroprogrammgesteuerte.
- Eine Speicheranordnung hat einen höheren Integrationsgrad als ein Netzwerk aus Gattern und Flipflops.
- Speicherinhalte sind einfacher zu ändern als Schaltungen. Ersteres gelingt notfalls auch von Hand, letzteres nur über die Schaltungssynthese.
- Letzten Endes ist alles eine Frage der Kosten und des Entwicklungsaufwands.

Ein Befehlsablaufbeispiel
Abb. 4.16 zeigt ein typisches Befehlsformat einer herkömmlichen CISC-Architektur. Es ist ein elementarer Additionsbefehl. Ein Speicheroperand wird zu einem Registeroperanden addiert; der Registeroperand wird mit dem Ergebnis überschrieben. Abb. 4.17 veranschaulicht den Mikroprogrammablauf.

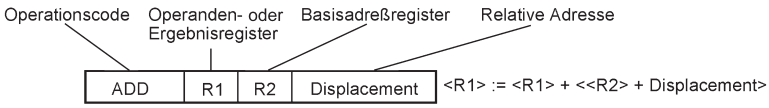
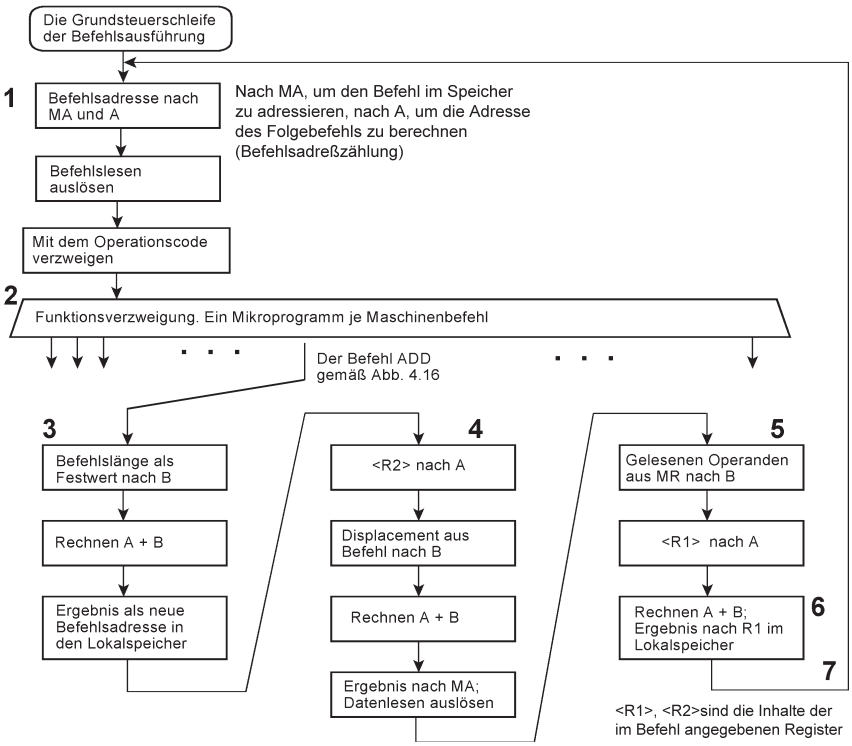


Abb. 4.16 Abb. Ein typischer Befehl einer CISC-Architektur.



1 Befehlslesen (Instruction Fetch)

2 Befehlsdecodierung (Instruction Decode) mittels Funktionsverzweigung. Der Operationscode fließt in eine Verzweigungsadresse ein. Jeder Befehl hat ein eigenes Mikroprogramm

3 Der Rest des Befehlslesens. Aufgrund der Verzweigung gemäß Operationscode ist bekannt, wie lang der Befehl ist. Die Befehlslänge wird zur Befehlsadresse addiert. Das ergibt den neuen Befehlszählerinhalt

4 Den Operanden lesen (Operand Fetch). Zunächst wird die Adreßrechnung Basisadresse + Displacement ausgeführt

5 Beide Operanden werden zunächst in die Register des Verarbeitungswerks geladen

6 Die Operation ausführen (Operate). Das Ergebnis wird im Lokalspeicher gespeichert

7 Der Befehl ist beendet. Fortsetzung mit dem Lesen des nächsten Befehls

Abb. 4.17 Die Ausführung des Befehls von Abb. 4.16 als Mikroprogrammablauf. Es ist eine pauschale Illustration unter Verzicht auf Einzelheiten und Spitzfindigkeiten. Zu den Phasen des Befehlsablaufs vgl. Abb. 4.13.

Mikrobefehle in Hochleistungsprozessoren

Für höchste Verarbeitungsleistung braucht man horizontale Mikrobefehle (Abschnitt 4.5, S. 192) sowie ein Verarbeitungswerk mit parallel wirkenden Verknüpfungsschaltungen und entsprechend leistungsfähigen Datenwegen. Im Extremfall braucht ein Maschinenbefehl nur einen einzigen Mikrobefehl. Der Operationscode adressiert einen Mikrobefehl, der alle Signalwege so einstellt wie jeweils erforderlich. Alle Befehlswirkungen werden dann in einem einzigen Taktzyklus erbracht. So kann man – wenngleich mit erheblichem Aufwand – auch auf Grundlage einer traditionellen CISC-Architektur dem Ideal nahekommen, in jedem Taktzyklus einen Befehl auszuführen[19]. Die Steuerwerke der Superskalar-Hochleistungsprozessoren unterscheiden sich zudem durch einige Besonderheiten von der herkömmlichen Mikroprogrammsteuerung:

- Die Mikrobefehle werden an unabhängige Verarbeitungseinrichtungen ausgegeben und dort selbständig abgearbeitet.
- Viele Maschinenbefehle lassen sich in einen einzigen Mikrobefehl umsetzen. Darüber hinaus kommt man oftmals mit kurzen Mikroprogrammen aus (bis zu vier Mikrobefehle sind typisch), in denen es keine Verzweigungen gibt.
- Nur einfache Befehle werden in die Ablaufbeschleunigung und Parallelausführung einbezogen. Komplexe Befehle[20] und architekturseitige Funktionen[21] werden nach wie vor mit herkömmlichen Mikroprogrammen erledigt

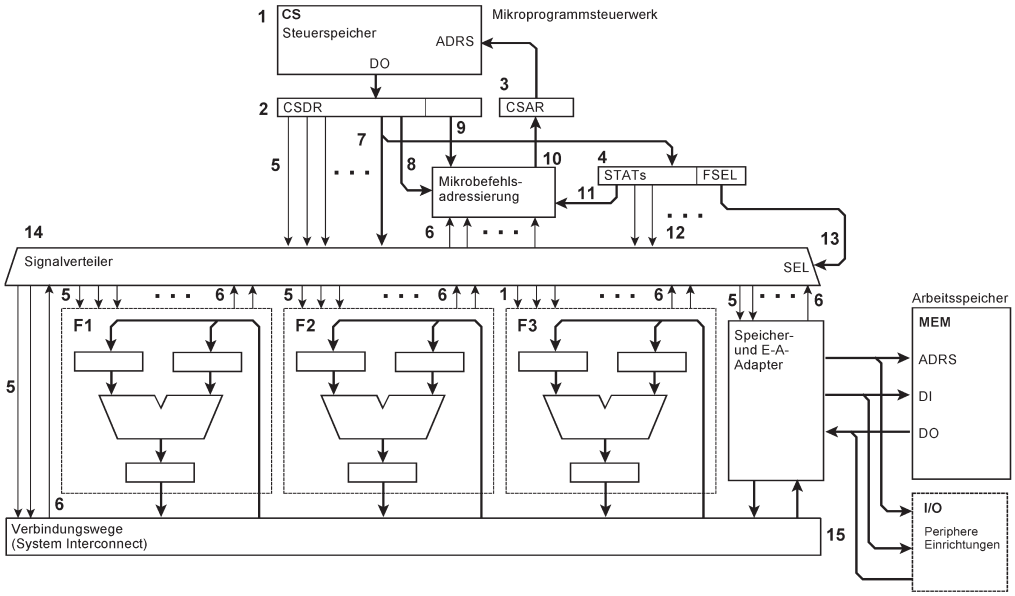
Das Mikroprogrammsteuerwerk als Dirigent im FPGA

Ein FPGA enthält anwendungsspezifische Funktionseinheiten. Um sie zu initialisieren, mit Parametern zu versorgen, deren Zusammenarbeit zu koordinieren, die Verwaltungsarbeiten zu erledigen und den Komfort bereitzustellen, den man heutzutage erwartet[22], schließt man alles an einen fertigen Prozessorkern (als IP Core) an. Das ist der übliche Stand der Technik.

Wir aber denken daran, den fertigen Prozessorkern durch ein Mikroprogrammsteuerwerk zu ersetzen. Was zu steuern ist, wird direkt angeschlossen; es ist dann gleichsam organisch eingebaut und ohne besonderen Overhead zu erreichen. Wir nehmen einen zwar großzügig dimensionierten, aber in seinen Wirkprinzipien einfachen "inneren" Computer als architekturseitige Grundlage, als Plattform. Das Mikroprogrammsteuerwerk ist eine Art Dirigent der Anwendungsschaltungen (Abb. 4.18). Es dient dazu, Komplexität beherrschbar zu machen. Die Anwendungsschaltungen sind nicht allzu komplex, so daß sie bequem mittels Verhaltensbe-

19. Mit anderen Worten: eine solche CISC-Maschine kann – was die MIPS/MHz angeht – ebenso viel leisten wie eine RISC-Maschine. Historisches Beispiel: Intels i486.
20. Deshalb empfehlen die Programmoptimierungsrichtlinien der Hersteller, die komplexen Befehle nicht zu verwenden und die jeweiligen Befehlswirkungen mit elementaren Befehlen auszuprogrammieren. Vgl. beispielsweise [95] bis [97]. Zu aktuellen Versionen s. das Internet.
21. Man denke an das Auslösen von Interrupts, an die Prozeßumschaltung (Context Switching / Swapping), an das Behandeln von Ausnahmebedingungen, an den virtuellen Speicher usw.
22. Vernetzung, automatische Updates, graphische Bedienoberflächen, kompatible Dateisysteme usw.

schreibung entworfen werden können. Deren Zusammenspiel wird vom Mikroprogrammsteuerwerk organisiert, das als optimierte Schaltungsstruktur entworfen und ggf. als harter IP Core implementiert wird. Auf diesem Wege kann man zu Superskalarmaschinen kommen, die direkt mit der Außenwelt zusammenwirken und als programmierte Emulatoren und Steuerautomaten außergewöhnlich viel leisten. Womöglich ergeben sich auch Alternativen der Funktionsaufteilung: mehr ins Mikroprogramm verlagern (ist vergleichsweise schnell geschrieben, erprobt und geändert), dafür werden die speziellen Schaltungen einfacher (so daß sie sich leichter entwerfen, simulieren und synthetisieren lassen). Auch könnte man komplexere Funktionseinheiten vorsehen, die als Operationsautomaten mit eigener Mikroprogrammsteuerung implementiert werden. Das übergeordnete Mikroprogrammsteuerwerk wird so zum Steuerautomaten im Sinne der Theorie der universellen algorithmischen Automaten. Es liegt dann nahe, es als Boolesche Maschine zu optimieren, die Ereignisse erkennen und Kommandos auslösen kann.



1	Steuerspeicher	9	Mikrobefehlsadresse im Mikrobefehl
2	Mikrobefehlsregister	10	Mikrobefehlsadresse
3	Mikrobefehlsadreßregister	11	Voreinstellung der Mikrobefehlsadressierung
4	Voreinstellflipflops (Staticizers, Bankregister)	12	Voreinstellsignale
5	Steuersignale	13	Funktionseinheitenauswahl
6	Bedingungssignale	14	Signalverteiler
7	Direktwerte	15	Verbindungswege der Funktionseinheiten
8	Steuerung der Mikrobefehlsadressierung		

Abb. 4.18 Das Mikroprogrammsteuerwerk als Plattform und Dirigent im FPGA. F1, F2, F3 sind Beispiele von Anwendungsschaltungen (Funktionseinheiten).

Mehrere Mikroprogramme gleichzeitig
Der Gedanke liegt nahe, den doch nicht unbeträchtlichen Aufwand eines leistungsfähigen Mikroprogrammsteuerwerks auszunutzen, um mehrere Programme oder Mikroprogramme gleichzeitig laufen zu lassen.

Herkömmliche Systemarchitektur, herkömmliche Maschinenbefehle
Die Rede ist von herkömmlichen Maschinen und Betriebssystemen. Traditionell wird der gesamte Mehrprogrammbetrieb – die Unterbrechungsannahme, das Retten und Wiedereinstellen, die Prozeßumschaltung usw. – mit (System-) Software erledigt. Das hat aber eine beträchtlichen Overhead und somit ziemlich lange Latenzzeiten zur Folge[23]. Um diese Verwaltungsfunktionen zu beschleunigen, kann man sie ins Mikroprogramm verlagern, zumindest die leistungsentscheidenden Abläufe (Microprogrammed Assists). Die Unterbrechungsannahme, das Retten, Wiedereinstellen usw. sind Standardfunktionen, die man immer wieder braucht. Wenn sie einmal funktionieren, werden sie nicht mehr grundsätzlich geändert. Unter diesen Bedingungen kann man sich den höheren Arbeitsaufwand der Mikroprogrammierung typischerweisen leisten.

Multitasking mit Mikroprogrammen
Was wir hier so bezeichnen, findet auschließlich auf der Ebene der Mikroprogramme statt; die Maschinenbefehle merken davon nichts. Es kann nur sein, daß sie zeitweilig langsamer laufen. Im Prinzip unterscheidet sich das Multitasking im Mikroprogrammsteuerwerk nicht vom Multitasking in der Software. Das Grundproblem ist immer das Retten und Wiedereinstellen des jeweiligen Maschinenzustands. Das kann gelöst werden, indem man Rettungsbereiche bereitstellt und die Informationstransporte mit Mikrobefehlen ausprogrammiert. Manchmal sieht man im Mikroprogrammsteuerwerk auch einen Hardwarestack vor.
Noch geringere Latenzzeiten ergeben sich, wenn man das Mikroprogrammsteuerwerk mit umschaltbaren Registern und Registersätzen ausrüstet. Dann kann man womöglich schon von einem Maschinentakt auf den anderen umschalten, ohne jeglichen Overhead. Es ist eine Frage des Aufwands[24]. Der Übergang zum Hardware-Multitasking ist fließend.
Die Mikroprogrammumschaltung mit Retten und Wiedereinstellen wird beispielsweise genutzt, um auf Fehlerbedingungen zu reagieren oder die Maschine mit Testmikroprogrammen zu überprüfen (periodisch oder dann, wenn ansonsten nichts zu tun ist). Solche Aufgaben auf der Ebene der Mikroprogramme zu erledigen hat den Vorteil, daß die Maschinenbefehle davon nichts mitbekommen. Die Fehlerbehandlung, Diagnose usw. ist vollkommen transparent. Auch kann das Mikroprogramm den Zustand der Befehlsausführung und den Programmkontext bis aufs Bit analysieren, zwecks späterer Auswertung retten usw. Werden hingegen die Fehler mit Software behandelt (Maschinenfehlerunterbrechung), wird allein schon das Umschalten auf die Fehlerbe-

23. In komplexen Systemumgebungen spricht man von mehreren bis vielen Millisekunden. Beispiel eines Richtwerts: Prozeßumschaltung in Windows etwa 20 ms, weitgehend unabhängig von der Prozessorleistung.
24. Näheres in Kapitel 5. Vgl. insbesondere die Seiten 299 bis 312.

handlung und das Ausführen der entsprechenden Befehle den Programmkontext verändern; sobald weitere Befehle ausgeführt werden, ist der Maschinenzustand verschwunden, der beim Auftreten des Fehlers vorgelegen hat[25].

Programm- und Mikroprogrammunterbrechungen
Geht es um Prozessoren mit Mikroprogrammsteuerung, so müssen wir auf die Unterschiede achten:

- Die Programmunterbrechung ist eine Unterbrechung des Befehlsablaufs. Sie gehört zum Programmiermodell der Maschinenarchitektur. Es gibt Anwendungsprogrammschnittstellen zur Unterbrechungsbehandlung. Der übliche Fachbegriff: Interrupt. Rückkehr zum unterbrochenen Programm: mit Befehl Return from Interrupt (RETI).
- Die Mikroprogrammunterbrechung gehört nicht zum Programmiermodell. Sie dient dazu, das Mikroprogrammsteuerwerk für weitere Aufgaben auszunutzen, beispielsweise zum Steuern von E-A-Abläufen. Die Maschinenbefehle laufen womöglich etwas langsamer, merken aber nichts davon. Ein traditioneller Fachbegriff[26]: Break-In. Rückkehr zum unterbrochenen Mikroprogramm: mit Mikroanweisung Break Out (BRKOUT). Ein Break-In ist eine einfache Unterbrechung (Lightweight Interrupt), die nach dem Schema Auslösung – Behandlung – Rückkehr abläuft. Die Unterbrechungsbehandlung erfordert zumeist nur wenige Mikrobefehle[27].

Hardware-Multitasking
Das Prinzip des zyklischen taktweisen Umschaltens zwischen Registersätzen – wie in Abschnitt 3.7 beschrieben – kann man auch im Mikroprogrammsteuerwerk implementieren. So könnten beispielsweise in einem Hautpzyklus (Major Cycle), der aus vier Maschinentakten (Minor Cycles) besteht, je ein Mikrobefehl von vier virtuellen Mikroprogrammsteuerwerken ausgeführt werden (Abb. 4.20). Für alles, was über den aktuellen Mikrobefehlszyklus (Minor Cycle) hinaus erhalten bleiben muß, also für den Maschinenzustand, braucht jede virtuelle Maschine eigene Register. Im Beispiel von Abb. 4.20 betrifft das die Mikrobefehlsadresse und die Voreinstellungen[28].

25. Das Problem wäre mit zusätzlichen Aufzeichnungsspeichern (History Buffers o. dergl.) zu lösen (gelegentlich ist es auch so gelöst worden). Das bedeutet aber einen nicht unerheblichen zusätzlichen Aufwand.
26. Er gehört zum Jargon im Umfeld des Systems /360. Das Prinzip wurde ursprünglich mit dem System /360 bekannt (als Beispiel vgl. [MD4]). Es diente dazu, die Schaltmittel des Prozessors auszunutzen, um Funktionen der E-A-Kanäle zu implementieren.
27. Im Beispiel [MD4] werden 6 Mikrobefehle genannt. Das war eine Entwurfsvorgabe; ein Break-In durfte nicht länger dauern.
28. Bedingungen müssen wir hier nicht retten, weil sie schon im aktuellen Mikrobefehl in die Bildung der Folgeadresse einfließen; Abfrage und Verzweigung bilden eine Einheit.

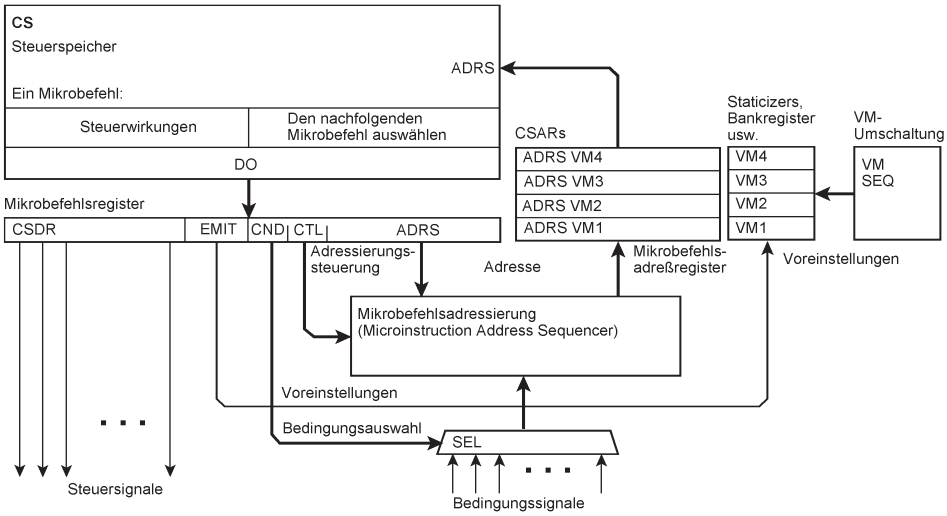


Abb. 4.19 Hardware-Multitasking im Mikroprogrammsteuerwerk. Hier werden vier virtuelle Maschinen unterstützt. Die Register können beispielsweise in einem Block-RAM untergebracht werden, der von einem Zähler adressiert wird (VM-Umschaltung). Vgl. auch Abschnitt 3.7.

4.3 Mikrobefehle im Operationsautomaten

Wir beziehen uns auf das Prinzip des universellen algorithmischen Automaten (Abschnitt 2.4, vgl. insbesondere Abb. 2.59). Der Steuerautomat führt Mikrobefehle aus. Diese Mikrobefehle haben Felder (in Abb. 2.59 mit Y bezeichnet), die Kommandos enthalten, die der Operationsautomat ausführen soll. Im Grunde bilden diese Felder einen Mikrobefehl, der den Operationsautomaten steuert. Es liegt nahe, auch den Operationsautomaten als mikroprogrammgesteuerte Einrichtung zu entwerfen und diesen Ansatz Schritt für Schritt weiterzubilden.

Der Operationsautomat wird aber kein autonomer Automat, sondern arbeitet nach dem Prinzip der Ereignis- oder Kommandosteuerung (Start-Stop-Prinzip).

Wenn wir das Schema von Abb. 2.59 1:1 implementieren, liefert der Steuerautomat die Operationsanweisungen des auszuführenden Mikrobefehls an den Operationsautomaten. Der braucht dann keinen Mikroprogrammspeicher. Es genügt ein Mikrobefehlsregister (Abb. 4.20). Der Mikrobefehl wird, falls erforderlich, mit Parametern ergänzt. Das sind vor allem Adressen und Direktwerte.

Mikroanweisungen für Operationsautomaten können aus vielen Bits bestehen. Deren Übertragung erfordert dann aufwendige, breite Signalwege. Abb. 4.21 veranschaulicht eine alternative Lösung. Wir bauen nun doch einen Mikroprogrammspeicher ein. Der Steuererautomat liefert keine Mikroanweisungen, sondern eine Mikrobefehlsadresse oder einen Kommandocode, der einen Mikrobefehl aus dem Mikroprogrammspeicher im Operationsautomaten auswählt (Funktionsverzweigung). Erforderlichenfalls werden zudem Parameter übergeben.

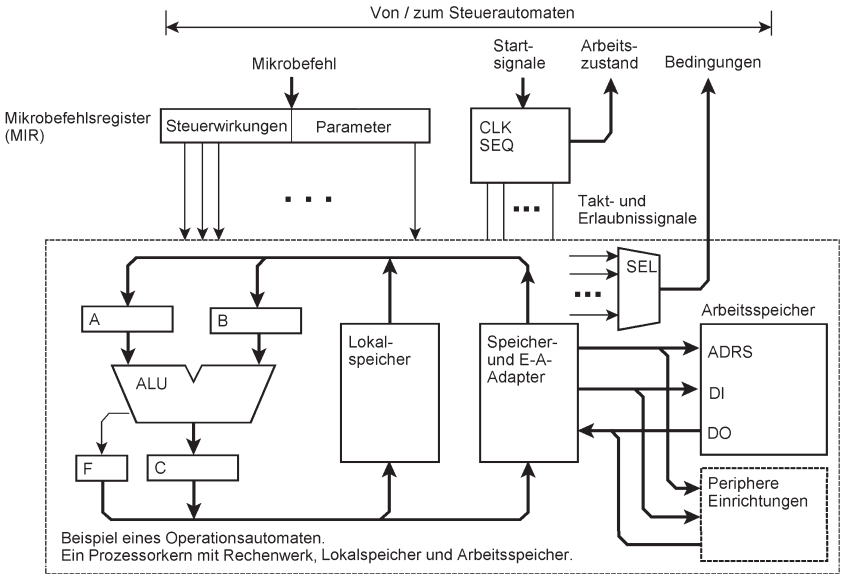


Abb. 4.20 Der Operationsautomat erhält seine Mikrobefehle vom Steuerautomaten.

Operationsautomaten müssen nicht auf einen einzigen Mikrobefehl beschränkt sein, dessen Ausführung von außen angewiesen wird. In letzter Konsequenz kann der Operationsautomat Abläufe beliebiger Komplexität ausführen. Dafür wird er mit einem kompletten Mikroprogrammsteuerwerk ausgerüstet (Abb. 4.22). Das wird aber nicht als autonomer Automat ausgelegt, sondern mit Kommandos gestartet (Start-Stop-Prinzip)[29]. Der Operationsautomat befindet sich zunächst im Ruhezustand. Der Steuerautomat schickt ein Kommando, worauf hin das Mikroprogrammsteuerwerk im Operationsautomaten zu arbeiten beginnt. Ist das Kommando ausgeführt worden, bewirkt der letzte Mikrobefehl die Rückkehr in den Ruhezustand. Das Mikroprogrammsteuerwerk von Abb. 4.22 arbeitet wie die anderen hier beschriebenen Mikroprogrammsteuerwerke im Sinne eines inneren Computers, nur daß es von außen gestartet wird und seine Arbeit auch wieder beendet, also nicht in eine Endlosschleife einläuft.

Mikrobefehle oder Kommandocodes?

Wenn der Steuerautomat die Mikrobefehle direkt zum Operationsautomaten schickt, kann er jede überhaupt mögliche Mikrobefehlswirkung auslösen. Ansonsten kann er nur einen der gespeicherten Mikrobefehle adressieren. Zu den Vorteilen der Adreß- oder Kommandoübertragung gehört aber, daß man damit einheitliche Formate implementieren kann. So könnte in einer Maschine gemäß Abb. 4.18 die eine Funktionseinheit 17 Mikrobefehlsbits benötigen, die andere

29. Der autonome Computer im Computer lädt solche Codes selbst (Funktionsverzweigung), der Operationsautomat bekommt sie von außen.

53 Bits usw. Nach dem Prinzip von Abb. 4.21 hingegen könnte man für alle Funktionseinheiten beispielsweise jeweils 12 Bits Adresse = Kommandocode und 32 Parameterbits festlegen.

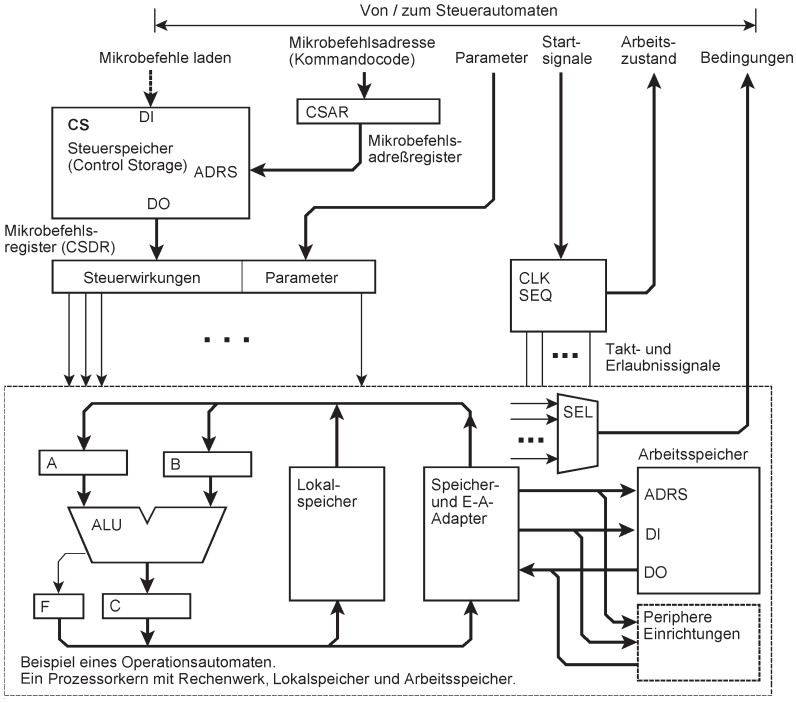


Abb. 4.21 Operationsautomat mit Mikroprogrammspeicher. Der Steuerautomat schickt die Adresse des auszuführenden Mikrobefehls, also praktisch einen Kommandocode. Es wird nur dieser eine Mikrobefehl ausgeführt.

Mikrobefehle laden

Wenn der Steuerautomat den Mikroprogrammspeicher nachladen kann, lassen sich die Beschränkungen der Kommandoübertragung umgehen. Der Steuerautomat ist dann in der Lage, Mikrobefehle sozusagen auf Vorrat zu laden.

Bedingungssignale

Die Theorie sagt, daß Bedingungssignale aus dem Operationsautomaten im Steuerautomaten ausgewertet werden. Oftmals ist es zweckmäßig, im Operationsautomaten eine Vorauswahl zu treffen. Das ist auch in den Abbildungen angedeutet. Im Operationsautomaten von Abb. 4.22 werden Bedingungssignale sowohl zum Steuerautomaten geschickt als auch intern zur Mikrobefehlsadressierung ausgewertet (Verzweigungen).

Grundfunktionen der Start-Stop-Steuerung

Der Steuerautomat kann den Zustand des Operationsautomaten abfragen (Ruhe – Arbeit, ggf. Fehlerzustände und Sonderbedingungen) und Kommandos auslösen. Es gibt normale Komman-

dos und Abbruchkommandos. Ein normales Kommando wird im Operationsautomaten nur dann wirksam, wenn er sich im Ruhezustand befindet, ein Abbruchkommando immer. Wie die Kommandos ausgelöst werden, ist Sache der Entwurfsoptimierung. Im einfachsten Fall löst der Steuerautomat ein Kommando nur dann aus, wenn sich der Operationsautomat im Ruhezustand befindet. Ein Abbruchkommando bewirkt, daß der Operationsautomat sofort in den Ruhezustand übergeht. Darüber hinaus könnte man u. a. an FIFO-Puffer denken, um eine Kommandowarteschlange zu halten. Wenn der Operationsautomat selbsttätig Mikrobefehle ausführt, müssen Mikroanweisungen vorgesehen werden, um den Ablauf anzuhalten, den Betriebszustand zu signalisieren und ggf. das Einspeisen neuer Kommandos zu erlauben.

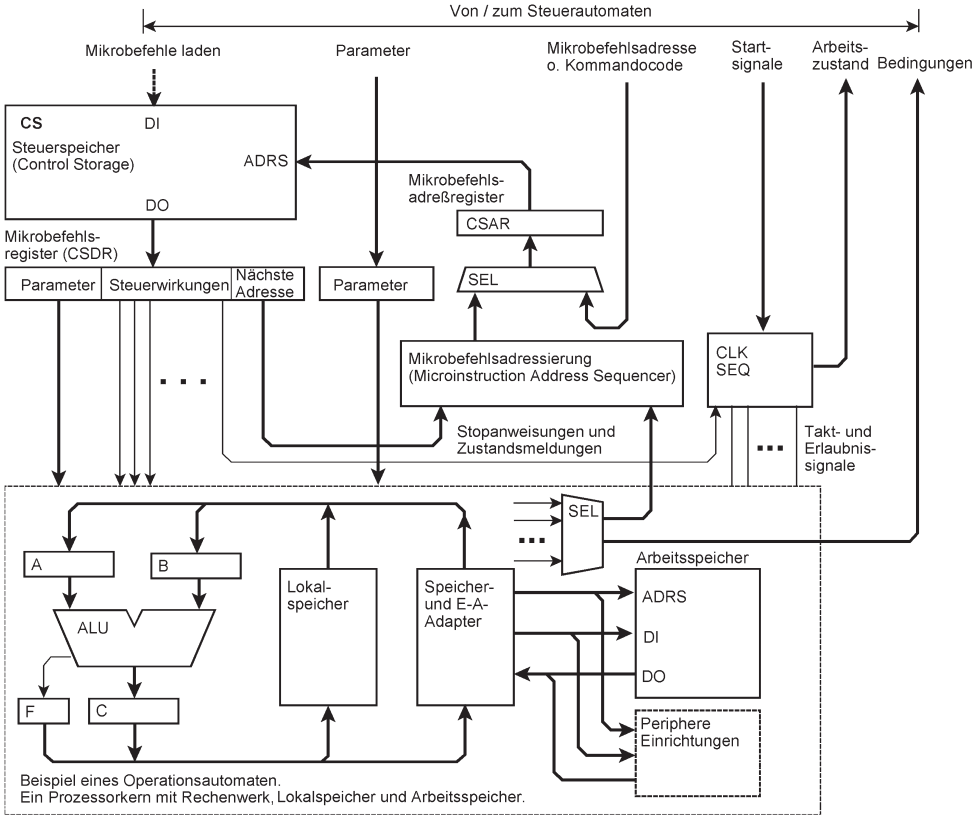


Abb. 4.22 Operationsautomat mit eigenem Mikroprogrammsteuerwerk.

Mikrobefehle auslagern

Das Prinzip ist in manchen Superskalarmaschinen mit komplexen Befehlslisten (CISC) üblich, kann aber auch als Anregung dienen, eigene Schaltungslösungen auszugestalten. Es ist im Grunde ein Implementierungsbeispiel der Entwurfgedanken, die in den Abb. 4.20 bis 4.22 dar-

gestellt sind. Das einzelne Verarbeitungswerk ist ein Operationsautomat, das zentrale Mikroprogrammsteuerwerk der Maschine ist der Steuerautomat.
Viele Befehle der CISC-Architekturen sind gar nicht besonders komplex. Um sie auszuführen, genügen nur wenige Maschinenzyklen. Es ist eine zeitstarre Folge, die man im Prinzip auch mit einfachen Sequencern steuern könnte (Steuerkette; vgl. S. 163ff). In jedem Zyklus sind aber viele Steuersignale zu erregen. Um sie zu bilden, würden man aufwendige Decoder benötigen. Deshalb entnimmt man die Signalbitmuster einem Speicher, der sequentiell adressiert wird. Das ergibt ein kleines, einfaches Mikroprogrammsteuerwerk. Wenn man mit wenigen Mikrobefehlen auskommt (beispielsweise mit 4 bis 16) , braucht man womöglich gar keinen richtigen Speicher. Es genügen Multiplexer, an deren Eingänge man die Mikrobefehlsbits anlegt, oder entsprechend geladene Schieberegister.
Wie aber entwirft man eine solche Funktionseinheit, wenn sie mehrere Befehle ausführen soll, beispielsweise 20 verschiedene? Dann würde das kleine Mikroprogrammsteuerwerk größer und komplizierter werden, mit einem größeren Mikroprogrammspeicher, mit Funktionsverzweigung zum Aufrufen der unterschiedlichen Mikroprogramme usw. Der Ausweg: man bleibt bei dem kleinen Mikroprogrammspeicher, den man auch mit Multiplexern, Schieberegistern usw. bauen kann. Das ergibt sehr kurze Zugriffszeiten. Dieser Speicher wird während der Befehlsdecodierung mit den Mikrobefehlen des auszuführenden Befehls geladen (aus einem Mikroprogrammspeicher des zentralen Steuerwerks oder der betreffenden Pipeline-Stufe). Mit entsprechendem Schaltungsaufwand kann man auch alle Mikrobefehle auf einmal liefern (paralleles Laden)[30]. So könnte man beispielsweise 4 Mikrobefehle zu 24 Bits in einem Steuerwort von 96 Bits unterbringen, das in einem einzigen Taktzyklus in den kleinen Mikroprogrammspeicher – im Grunde nur ein Registersatz – eingetragen wird. Abb. 4.23 veranschaulicht das Prinzip anhand eines einfachen Verarbeitungswerks. In einer Superskalarmaschine ist ein solches Werk nur eines von mehreren. Deshalb hat es in diesem Beispiel keinen eigenen Speicheradapter[31], sondern ist mit einem Lokalspeicher bzw. Registersatz und Kommunikationsschnittstellen verbunden.
Wenn ein Befehlsablauf nicht mit diesen lokalen Mikrobefehlen gesteuert werden kann, greift das zentrale Mikroprogrammsteuerwerk ein. Dann entfällt auch das gleichzeitige Ausführen verschiedener Befehle; die Superskalarmaschine wird zeitweilig zum herkömmlichen Einzelprozessor, in dem die Befehle nacheinander ausgeführt werden, einer zu einer Zeit.
Wieviele lokale Mikobefehle? – Naheliegenderweise nur soviele, wie man ohne zusätzliche Taktzyklen (des Pipelining-Taktrasters) laden kann. Wenn man zusätzliche Takte braucht, um die Mikrobefehle zu laden, lohnt es sich nicht. Wie legt man die lokale Ablaufsteuerung aus – als ganz einfachen Sequencer (1., 2., 3., 4. usw. Mikrobefehl als starre Folge) oder mit internen Wartezuständen und bedingten Zustandsübergängen, also letzten Endes so ähnlich wie die Steuerautomaten der Abb. 2.44 oder 6.17? Das ist Sache der Optimierung. Mit lokalen Wartezuständen und bedingten Verzweigungen wird die Maschine zwar flexibler, diese Funktionen

30. Vgl. Einzelheit ** in Abb. 4.23
31. Das Schreiben in den Arbeitsspeicher ist Sache anderer Pipelinestufen und spezieller Funktionseinheiten.

erfordern aber Bits im Mikrobefehl, Schaltungsaufwand und Durchlaufzeit. Deshalb bleibt man zumeist beim einfachsten Prinzip, bei der zeitstarren Ausführung von beispielsweise vier Mikrobefehlen, und ruft immer dann, wenn es komplizierter wird, das zentrale Mikroprogrammsteuerwerk zu Hilfe.

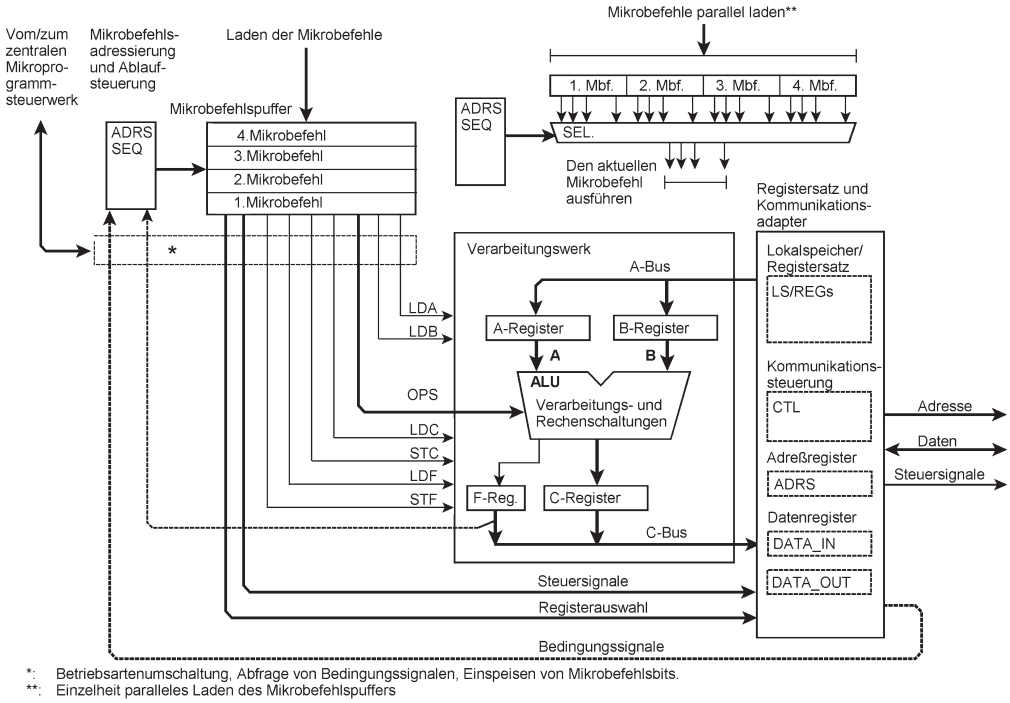


Abb. 4.23 Eine Verarbeitungseinheit, die mit kurzen lokalen Mikroprogrammen gesteuert wird. Ist ein Befehl auszuführen, wird der Mikrobefehlspuffer mit den Mikrobefehlen geladen, die dazu nötig sind. Ist es ein komplizierter Befehlsabauf, kommen die Mikrobefehle vom zentralen Mikroprogrammsteuerwerk. Es speist die Mikrobefehlsbits ein und fragt die Bedingungssignale ab.

4.4 Leistungsprobleme der Mikroprogrammsteuerung

Eine Maschine, die für jede Funktion eigene hart verdrahtete Schaltungen aufweist, kann sozusagen alles auf einmal erledigen, was sich überhaupt erledigen läßt. Eine Programmsteuerung hingegen braucht oftmals mehrere Schritte. Das gilt auch für die Mikroprogrammsteuerung. Daraus ergeben sich gelegentlich spürbare Leistungsgrenzen.

4.4.1 Verzweigungen

Der einzelne Mikrobefehl kann nur zwischen wenigen Folgezuständen auswählen (zwei bei einfacher Verzweigung, vier bis beispielsweise 16 bei Mehrfach- oder Funktionsverzweigung).

Stehen mehr Folgezustände zur Wahl als man mit einem Mikrobefehl auswählen kann, so müssen Zwischenzustände des Auswählens und Verzweigens eingefügt werden (Geschwindigkeitsverlust). Kombinatorische Gatternetzwerke kann man hingegen – zumindest dem Prinzip nach – so auslegen, daß ein einziger Taktzyklus genügt, jeden beliebigen Folgezustand einzunehmen. Das wird in Abb. 4.24 und Tabelle 4.2 anhand eines Beispiels veranschaulicht. Es ist ein Operand aus dem Speicher zu lesen. Welcher Zustand als nächster einzunehmen ist, hängt davon ab, was beim Lesen passiert. Es kann allerhand passieren. Im Beispiel gibt es sechs Folgezustände. Wenn die Mikroprogrammsteuerung nur zwischen zwei Richtungen wählen kann (bedingte Verzweigung), so müßte etwa so programmiert werden wie in Abb. 4.24c dargestellt. Jede dieser Entscheidungen kostet aber einen Mikrobefehl und damit einen Taktzyklus.
Würde man das NOR-Gatter (Abb. 4.24b) einsparen, so würde die o.k.-Bedingung nicht in der Hardware erkannt werden. Man könnte sie somit auch nicht am Anfang abfragen. Sie würde sich vielmehr erst ergeben, nachdem sämtliche Fehlerbedingungen abgefragt wurden. Der mikroprogrammierte Speicherzugriff würde dann *sehr* langsam werden ...
Demgegenüber kann eine hart verdrahtete Steuerung mit kombinatorischen Verknüpfungen in einem einzigen Taktzylus in jeden der Folgezustände übergehen (Abb. 4.24d).

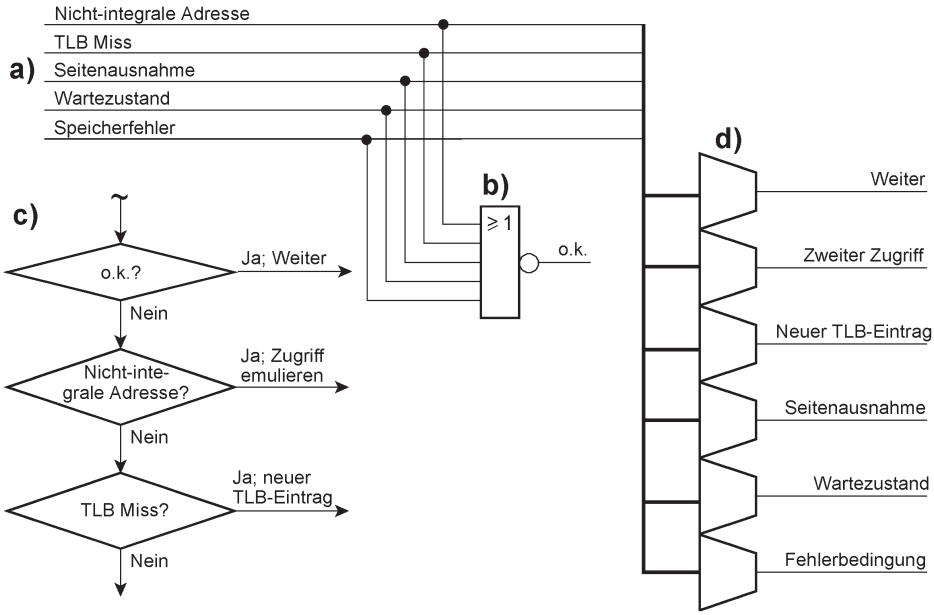


Abb. 4.24 Bedingungsauswertung beim Operandenlesen. a) Bedingungssignale (vom Speicheradapter); b) Bildung der o.k.-Bedingung (wird immer dann wirksam, wenn der Speicheradapter nichts Besonderes meldet); c) Bedingungsabfrage im Mikroprogramm; d) Bestimmung des Folgezustands mit kombinatorischen Verknüpfungen.

Mögliche Situationen beim Lesen eines Operanden (Auswahl)	Typische Reaktionen
Es ist alles in Ordnung (o.k.)	Befehlsablauf fortsetzen
Die Zugriffsadresse ist keine integrale Adresse	Fehlerbedingung auslösen oder Zugriff emulieren (zwei Zugriffe ausführen und den Operanden aus zwei Abschnitten zusammensetzen)
Ein TLB Miss	Einen neuen TLB-Eintrag aufbauen (Hardware/Mikroprogramm)
Eine Seitenausnahme	Ausnahmebedingung auslösen
Die Daten müssen erst geholt werden	Wartezustand
Ein Speicherfehler	Maschinenfehlerbehandlung auslösen

Tabelle 4.2 Mögliche Folgezustände beim Operandenlesen.

4.4.2 Mit Parametern arbeiten

Ein Mikroprogramm starten heißt im Grunde eine Funktion aufrufen. Die Frage ist, wie die Parameter übergeben werden. Wen man alles im Sinne einer Architektur von Grund auf entwirft (Micro-Architecture), kann man es passend einrichten. Beim Emulieren vorgegebener Architekturen muß man hingegen mit den Vorgaben zurechtkommen. Das betrifft vor allem die Formate der Befehle, Deskriptoren, Adreßzeiger, Kommandowörter, Zustandswörter usw. Abb. 4.25 zeigt drei übliche Befehlsformate. Die Parameter, die ein solcher Befehl ans Mikroprogramm übergibt, sind der Operationscode, die Registeradressen, das Adreß-Displacement und der Direktwert.

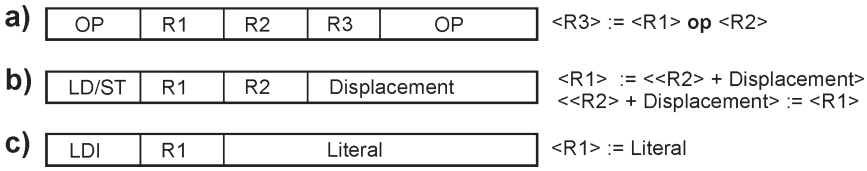


Abb. 4.25 Beispiele von Befehlsformaten. a) Operationsbefehl; b) Laden und Speichern; c) Laden eines Direktwerts. R1, R2, R3 sind Registeradressen.

Wie kann das Mikroprogramm mit diesen Parametern arbeiten? Betrachten wir die drei Beispiele:

a) Nach dem Befehlslesen will das Mikroprogramm zu der Routine verzweigen, die den betreffenden Befehlsablauf steuert. Dazu führt es eine Funktionsverzweigung mit einer Adresse aus, die aus den Bits des Operationscodes gebildet wird. Der Operationscode ist aber hier auf zwei Felder verteilt.

b) Um die Operandenadresse zu berechnen, braucht das Mikroprogramm die Registeradresse aus dem Feld R2 und das Displacement, das zum Addieren vorzeichengerecht auf die volle Wortlänge erweitert werden muß.
c) Auch der Direktwert muß vorzeichengerecht erweitert werden. Dieses Feld ist aber länger als das Displacement im Format b).

Wenn die Maschine nur diesen einen Befehlssatz unterstützen soll, kann man alles hart verdrahten (Abb 4.26 bis 4.28)[32].

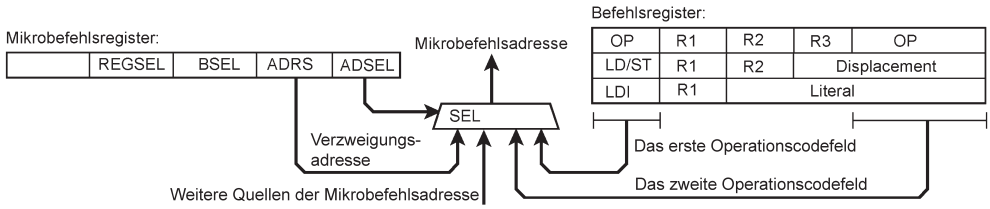


Abb. 4.26 Funktionsverzweigung mit dem Operationscode. Das ADSEL-Feld steuert, welche Signale in die Adreßbildung und Funktionsverzweigung einbezogen werden. Die Operationsbefehle von Abb. 4.25a erfordern zwei solcher Verzweigungen, zunächst mit dem ersten Operationscodefeld, dann mit dem zweiten.

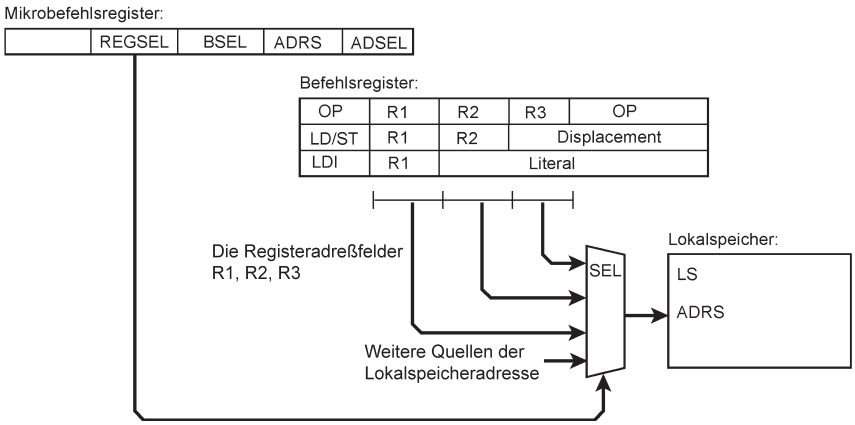


Abb. 4.27 Registeradressierung. Das REGSEL-Feld wählt das Registeradreßfeld im Befehl aus. Damit wird das betreffende Register im Lokalspeicher adressiert.

32. Solche Strukturen sind schon etwas unübersichtlich. Deshalb werden die Probleme a), b), c) getrennt in einzelnen Abbildungen dargestellt.

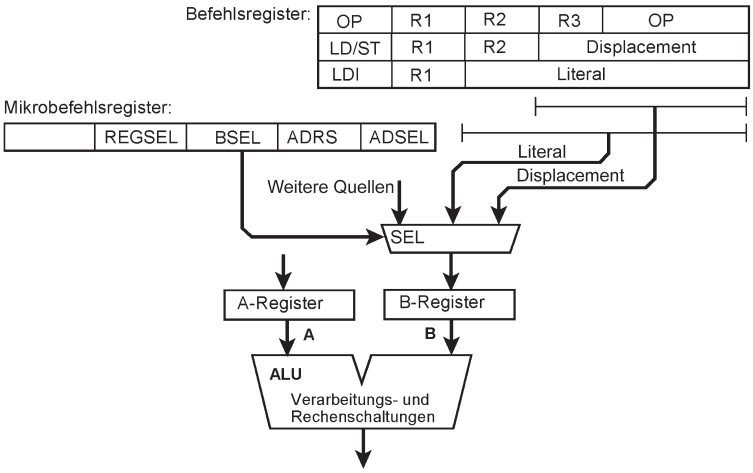


Abb. 4.28 Rechnen mit Direktwerten aus dem Befehl. Das BSEL-Feld wählt aus, welcher Operand am B-Eingang der ALU anliegt. Die Vorzeichenerweiterung auf die volle Wortlänge ist an den Eingängen der Auswahlschaltung hart verdrahtet.

Soll es aber ein universeller Emulator werden, eine Maschine, die beliebige Befehlssätze unterstützen kann, wird es schwierig. Um eine Verzweigungsadresse vorzubereiten, ein Register im Lokalspeicher zu adressieren oder einen Direktwert in voller Wortlänge bereitzustellen, muß das Mikroprogramm die jeweiligen Felder aus dem Maschinenbefehl herauslösen, jeweils passend aufbereiten[33] und in das jeweilige Adreß- oder Operandenregister laden. Wirklich kompliziert ist das nicht. Es kostet aber Zeit. Um ein Feld aus einem Befehl zu entnehmen, muß man den Inhalt des Befehlswortes verschieben, beispielsweise soweit, bis das niedrigstwertige Bit des Feldes in der niedrigstwertigen Bitposition des Maschinenwortes steht, und die überflüssigen Bits löschen. Maschinen, die als universelle Emulatoren entworfen wurden, haben eigens Mikrobefehle für solche Operationen. Beispiel: Burroughs B 1700 ([105], [106], [56]).

4.4.3 Wandeln und adaptieren

Wie kommen wir zu einem universellen Emulator, der schnell genug und trotzdem nicht allzu teuer ist? Eine Lösung wurde schon vor Jahrzehnten gefunden. Der Grundgedanke: Wir müssen zwischen den zu unterstützenden Befehlssätzen usw. nicht im laufenden Betrieb umschalten. Eigentlich brauchen wir nur eine Art grundsätzlicher Mikro-Architektur, eine Plattform, um auf dieser Grundlage je nach Bedarf spezifische Emulationsmaschinen anzubieten[34]. Alle speziellen Anpassungen werden mit Schaltungen erledigt, die in die Maschine eingefügt werden. Diese

33. Beispielsweise durch Verlängern, Hinzufügen von Festwerten oder Umcodieren.
34. Seinerzeit ging es u. a. darum, alte, bereits ausgestorbene Computer zu emulieren, um deren Software weiterverwenden zu können. Historisches Beispiel: das Transform Feature der CDC 5600 (S. 517ff).

Schaltungen sind im Grunde einfach. Ihre Aufgabe besteht vor allem darin, Signalwege bereitzustellen, Bedingungen zu codieren und Werte umzuwandeln. Es sind Aufgaben der Adaptierung. Früher hat man beispielsweise eigens Steckpositionen vorgesehen, in die Karten mit solchen Schaltungen eingesteckt werden konnten, und die Mikrobefehle hatten Anweisungsfelder, um die Schaltungen auf diesen Karten zu steuern. Heute kann man so etwas mit Funktionseinheiten auf dem FPGA erledigen, die einzeln synthetisiert und über Netzlisten in die Maschine eingebunden werden. Die Technologie ist fortgeschritten, das Prinzip bleibt. Die Abb. 4.29 und 4.30 zeigen zwei Beispiele. Abb. 4.29 löst das Problem von Abb. 4.24. Die decodierten Meldungen vom Speicher werden in einem Codierer (ENC = Encoder) in eine binäre Adresse umgewandelt (unter Berücksichtigung der Prioritäten usw.), die in die Funktionsverzweigung einfließt. Damit kann das Mikroprogramm im nächsten Maschinenzyklus auf alle Meldungen vom Speicher reagieren. Soll die Maschine mit einem anderen Speichersubsystem ausgerüstet werden, wird nur die Adapterschaltung geändert.

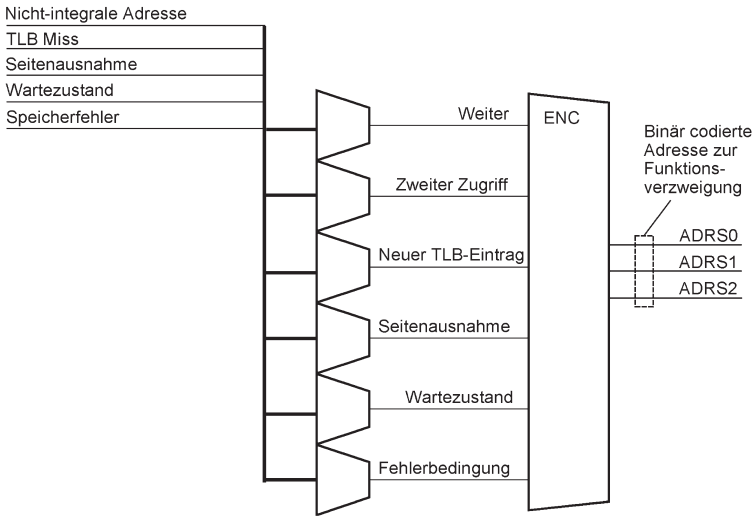


Abb. 4.29 Das Problem von Abb. 4.24 mit einer Adapterschaltung lösen. Hier ist es ein Codierer.

Abb. 4.30 zeigt, wie Registeradressen, die im Befehlsregister stehen, in Lokalspeicheradressen gewandelt werden. Hier sind drei Befehlsformate mit Registeradreßfeldern dargestellt[35]. Die Registeradressen können unterschiedlich lang sein, je nachdem welche Art von Registern adressiert wird (Universalregister, Adreßregister, Steuerregister usw.). Alle betreffenden Bitpositionen des Befehlsregisters sind an den Adapter angeschlossen. Die jeweilige Auswahl, Wandlung usw. wird vom Feld ADAP CTL gesteuert. Beispiel: Im Befehlsformat OP sind die Registeradressen 4 Bits lang, um eines von 16 Universalregistern auszuwählen, im Befehlsformat

35. Es sind fiktive Befehlsformate ähnlich dem Beispiel von Abb. 4.25.

LD/ST ist die Datenregisteradresse (R1) 5 Bits lang und die Adreßregisteradresse (R2) 3 Bits usw. Gesteuert vom Feld ADAP CTL wählt der Adapter jeweils eines der Registeradreßfelder aus und bildet daraus die Lokalspeicheradresse. Sie besteht aus den Adreßbits aus dem Befehl und aus Festwerten. Soll das Mikroprogrammsteuerwerk andere Befehlsformate unterstützen, wird nur der Adapter neu entworfen; die Lokalspeicheradressierung im Mikroprogrammsteuerwerk bleibt, ebenso die zugehörigen Felder in den Mikrobefehlen.

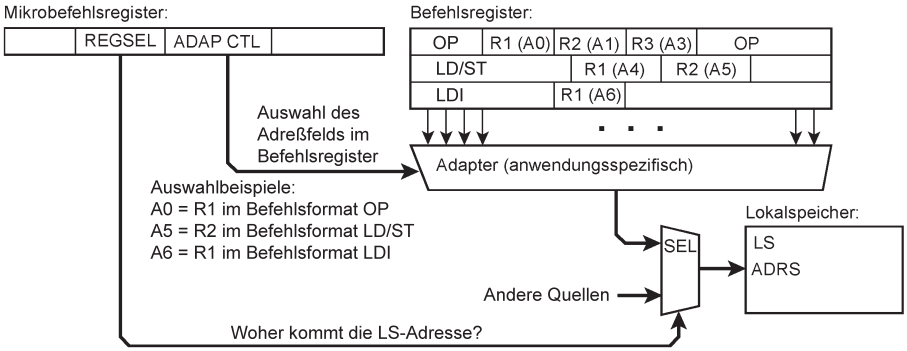


Abb. 4.30 Das Problem der Registeradressierung in den zu emulierenden Maschinenbefehlen mit einer Adapterschaltung lösen. Im Befehlsregister sind drei unterschiedliche Befehlsformate dargestellt. A0 bis A6 sind die Auswahlcodes der einzelnen Registeradressen im Feld ADAP CTL.

4.5 Mikrobefehlsformate

Ein Mikrobefehl enthält Bits und Bitfelder, die dazu dienen, Steuerwirkungen auszuüben und den nachfolgenden Mikrobefehl auszuwählen (Abb. 4.31a). Zudem kann der Mikrobefehl Direktwerte enthalten, die als Adressen, Konstanten usw. genutzt werden (Emit-Felder). Die grundsätzlichen Mikrobefehlsformate werden mit den bildhaften Begriffen “horizontal” und “vertikal” bezeichnet (Abb. 4.31b, c).

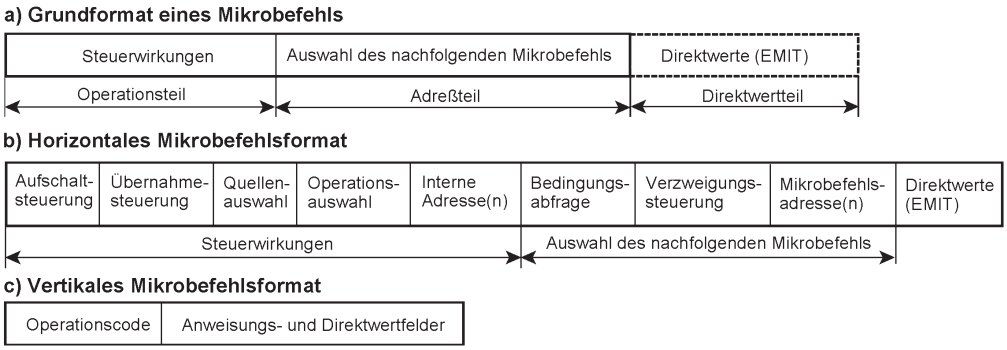


Abb. 4.31 Mikrobefehlsformate.

Horizontale Mikrobefehle

Ein Mikrobefehlsformat heißt horizontal, wenn alle Steuerwirkungen, die in einem Maschinenzyklus überhaupt ausgeführt werden können, in einem einzigen Mikrobefehl untergebracht sind (Abb. 4.31b). Solche Formate ergeben sich, wenn man alle Steuerbits, Bitfelder, Direktwerte, Adressen usw. aneinanderreiht. Derartige Mikrobefehle können hundert und mehr Bits lang sein. Es ist im Grunde der Entwurfsgedanke der Ressourcenvektormaschine. Welche Signale brauchen die zu steuernden Schaltungen, welche Bedingungssignale liefern sie zurück? Daraus ergibt sich, wieviele Bitpositionen anfänglich vorzusehen sind. Kürzen, sparen, optimieren kann man später.

Vertikale Mikrobefehle

Der einzelne Mikrobefehl ist kürzer (Richtwert: 12...32 Bits) und erbringt jeweils nur einige der insgesamt erforderlichen Steuerwirkungen (Abb. 4.31c). Eine typische Unterteilung ist die in Mikrobefehle zur Datenweg- und Operationssteuerung und solche zur Bedingungsabfrage und Verzweigung. Die extreme Auslegung: jeder Mikrobefehl hat jeweils nur ein Wirkung (Abfrage, Datenverknüpfung, Verzweigung usw.). Solche Mikrobefehle ähneln den Maschinenbefehlen der einfachen Mikrocontroller und RISC-Prozessoren (die Übergänge sind fließend). Die Theorie hat gezeigt, daß zwei Mikrobefehlstypen ausreichen: Operationsmikrobefehle mit nur einem Nachfolger und Verzweigungsmikrobefehle, die einen von zwei Nachfolgern auswählen[36].

Diagonale Mikrobefehle

Es ist eigentlich nur ein Sprachwitz (Pun intended), um Kompromißlösungen der Formatgestaltung zu bezeichnen. Sie bestehen in wenigen Formaten mittlerer Länge (Richtwert: 16...48 Bits)[37]. Es sind im Grunde verkürzte horizontale Mikrobefehle mit Steuerbits und Anweisungsfeldern, aber auf jeweils bestimmte Funktionen spezialisiert. So lieg es nahe, Formate zum Steuern, zum Rechnen mit Operanden und zum Verzweigen vorzusehen (Abb. 4.32).

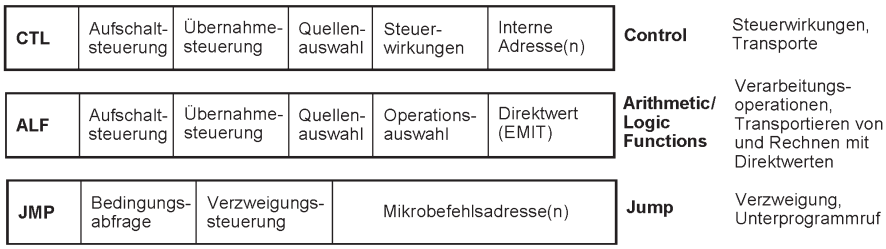


Abb. 4.32 Naheliegende vertikale oder – wenn man denn so will – diagonale Mikrobefehlsformate, vorgesehen für typische Aufgaben des Steuerns, Rechnens und Verzweigens. Die Operationsmikrobefehle wurden hier in zwei Formate aufgeteilt, eines zum Steuern (CTL) und eines für Operandenverknüpfungen (ALF).

36. Vgl. die sog. binären Mikrobefehle (Abschnitt 2.4, insbesondere S. 90f).
37. Solche Format passen gut zu den Zugriffsbreiten der typischen Block-RAMs in programmierbaren Schaltkreisen.

Horizontal oder vertikal?
Steuerwerk und zu steuernde Einrichtungen müssen zueinander passen; die Formatgestaltung sollte dem Leistungsvermögen der zu steuernden Einrichtungen entsprechen. Ein Mikrobefehlsformat sollte eine – möglichst brauchbare – Sammlung von Steuerwirkungen sein, die die Hardware jeweils in einem Maschinenzyklus erbringen kann.
Eine voll horizontale Auslegung lohnt sich nur dann, wenn tatsächlich alle Wirkungen, die der Mikrobefehl auslösen kann, im selben Maschinenzyklus erbracht werden können. Gelegentlich ist zu entscheiden, ob ein Mikroprogrammspeicher mit großer Aufrufbreite (und ein entsprechend breites Mikrobefehlsregister) ohne nennenswerten Decodieraufwand einem Speicher geringer Aufrufbreite, aber mit höherem Aufwand zur Decodierung der eigentlichen Steuersignale vorzuziehen ist oder nicht.

Sparen, koste es, was es wolle?
Das war einmal. Extreme Sparlösungen lohnen sich praktisch nicht mehr. Die Mikrobefehlsgestaltung sollte nicht zur übermäßigen Trickprogrammierung zwingen. Die Mikroprogrammsteuerung muß – im Hinblick auf die jeweilige Anwendung – mehr leisten als einer der üblichen Mikrocontroller oder RISC-Prozessoren, denn sonst kann man gleich einen der handelsüblichen Schaltkreise oder IP Cores verwenden. Da darf es auf ein paar Bits nicht ankommen[38]. Also zunächst so viele Bits vorsehen, wie zur Lösung der Entwurfsaufgabe erforderlich sind. Nur dann nachbessern, wenn das so erhaltene Format nicht in eine der jeweils verfügbaren Speicherstrukturen paßt (Beispiel: es haben sich 41 Bits ergeben, aber es stehen nur Block RAMs mit 36 Bits zur Verfügung)[39]. Die Wirtschaftlichkeit ergibt sich nicht durch Knausern im Kleinen (Ersparnis von Flipflops oder Gattern), sondern aus der genauen Anpassung an die Anwendungs- oder Entwicklungsaufgabe. Mikroprogramme für derart (also vergleichsweise großzügig) ausgelegte Steuerwerke sind im Grunde nicht wirklich schwieriger zu schreiben als übliche Maschinenprogramme. Während da die symbolischen Anweisungen – als Operationsocdes – einzeln untereinander stehen, gehören hier mehrere solche Anweisungen zusammen. Alles andere (symbolische Adressen, Direktwerte usw.) bleibt sich im Grunde gleich[40].

Voreinstellungen
Was bekommt der einzelne Mikrobefehl zu sehen – die gesamte Maschine mit all ihren Adreßräumen (Speicher, Register, E-A usw.) oder nur einen Ausschnitt, der nötig ist, um die jeweils aktuell auszuführende Funktion zu steuern? Wenn jeder Mikrobefehl in die gesamte Maschine hineinsehen und jedes Steuersignal erregen soll, muß er Adressen und Steuerbitfelder in voller Länge enthalten. Die Mikrobefehle dürfen aber nicht übermäßig lang werden, auch dann nicht, wenn beliebig viel Speicherkapazität zur Verfügung steht. Speicherinhalte zu lesen dauert um so

38. Also: Klotzen statt Kleckern (Heinz Guderian). Wenn das selbst entworfene Mikroprogrammsteuerwerk nicht mehr oder Besseres leistet als ein fertiger Prozessorkern, lohnt sich die Mühe gar nicht.
39. Der einfachste Ansatz: codierte Anweisungsfelder anstelle von Einzelbits. Hierzu kann man Anweisungen, die nie gleichzeitig ausgelöst werden, in codierten Feldern zusammenfassen.
40. Vgl. auch Anhang 1 (S. 537f).

länger, je mehr Bits zu holen sind. Ist die Zugriffsbreite gering, braucht man mehrere Speicherzyklen, ist sie groß, längere Zykluszeiten. Sie ergeben sich infolge der längeren Signalwege, der Takttoleranzen (Clock Skew), der gegenseitigen Störbeeinflussung (Crosstalk) und der Stromaufnahme beim Schalten der vielen Signale (di/dt).
Der Grundgedanke: der Mikrobefehl muß gar nicht alles auf einmal adressieren und steuern. Ein Mikroprogramm steuert die Ausführung eines Befehls oder eines Ablaufs der Unterbrechungsbehandlung, der Fehlerbehandlung usw. Wenn Gleitkommazahlen multipliziert werden, befaßt sich das Mikroprogramm nicht mit der Ein- und Ausgabe oder der Priorität von Unterbrechungsanforderungen. Um einen solchen Ablauf zu steuern, wird das Mikroprogramm mit nur wenigen Registern, Adressen, Steuersignalen und Bedingungen arbeiten (Lokalität der Zugriffe). Also muß der Mikrobefehl nur die Ausschnitte aus den Adreßräumen und nur die Steuersignale zu sehen bekommen, die er im aktuellen Ablauf braucht. Das wird in Registern und Flipflops voreingestellt (Abb. 4.33). Die Voreinstellung entspricht im Prinzip einem Maschinenzustand, der sich während des Mikroprogrammablaufs nicht ändert[41].

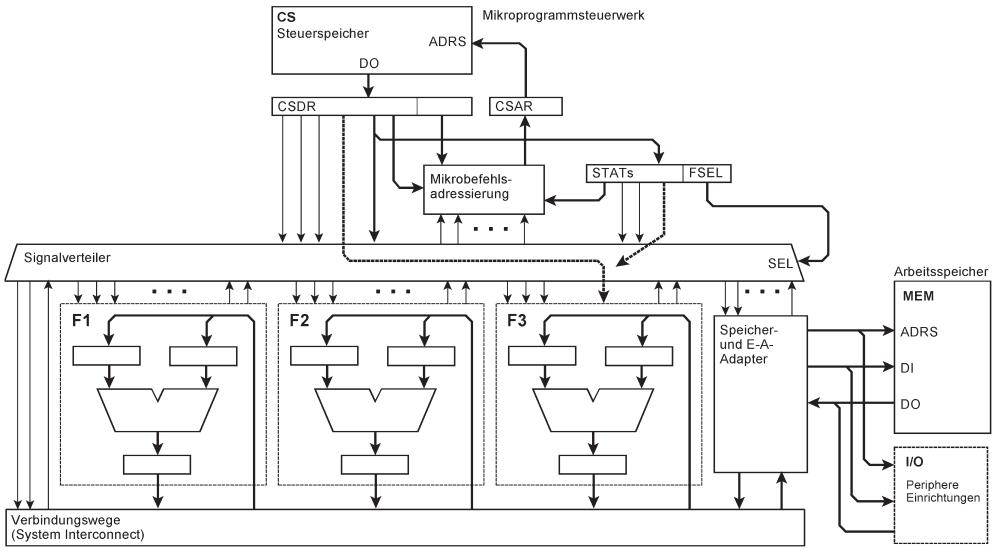


Abb. 4.33 Ein Beispiel der Voreinstellung. Es ist die Maschine von Abb. 4.18. Das Mikroprogramm beschäftigt sich hier mit der Funktionseinheit F3. Dazu muß der Mikrobefehl nur die Bits und Anweisungen enthalten, die erforderlich sind, um F3 zu steuern. Das wird hier mit den Staticizer-Flipflops (STATs) voreingestellt.

41. Er kann somit als statischer Zustand bezeichet werden. Deshalb die Bezeichung solcher Register und Flipflops als Staticizers oder STATs. Da wir für für die Funktionseinheiten der Voreinstellung einen kennzeichnenden Ausdruck brauchen, greifen wir den Slang aus alten S/360-Zeiten auf, ungeachtet der seinerzeitigen Bedeutung, die von unserer etwas abweicht. [MD1] ist ein Beispiel einer Maschinendokumentation, die solche Begriffe erklärt und entsprechende Schaltungen zeigt.

Auf diese Weise ergeben sich Mikrobefehle, deren Felder und Steuerbits unterschiedliche Wirkungen haben, je nachdem, wie die Maschine voreingestellt ist oder in welchem Zustand sie arbeitet. Das kann soweit gehen, daß man die komplette Formatierung umschaltet. Ein Gleitkomma-Mikrobefehl sieht dann ganz anders aus als ein Mikrobefehl der Unterbrechungssteuerung usw. Im Beispiel von Abb. 4.33 können die Mikrobefehle für die Funktionseinheiten F1, F2, F3 usw. jeweils unterschiedlich formatiert sein. Wenn man das Hardware-Multitasking implementiert, könnte jede virtuelle Maschine ein anderes Mikrobefehlsformat haben. Beispielsweise könnte die virtuelle Maschine VM1 die Funktionseinheit F1 mit Mikrobefehlen steuern, die für eigens F1 formatiert sind, die virtuelle Maschine VM2 die Funktionseinheit F2 usw. (vgl. Abschnitt 3.7, insbesondere Abb. 3.37, S. 157).

Abb. 4.34 veranschaulicht den grundsätzlichen Zweck der Voreinstellung:

a) zeigt einen Mikrobefehl, der aus mehreren Feldern besteht. Die Frage ist, wie lang sie sind.
b) Hier wird nicht gespart. Alle Felder sind so lang, daß jeweils alles untergebracht werden kann: die Steuersignale und Bedingungsauswertung für die gesamte Maschine, Adressen für alle Adreßräume, Konstanten bzw. Direktwerte in voller Verarbeitungsbreite usw. In der Praxis werden aber solche extrem langen Mikrobefehle zumeist nur als fiktive Befehlsformate im Entwurfsablauf[42] in Betracht kommen; in der Hardware müssen sie kürzer sein[43].
c) zeigt ein Mikrobefehlsformat mit kurzen Feldern. Sie reichen nun aber nicht mehr für alles. Also muß man sie bestimmten Funktionseinheiten, bestimmten Abläufen usw. zuordnen.

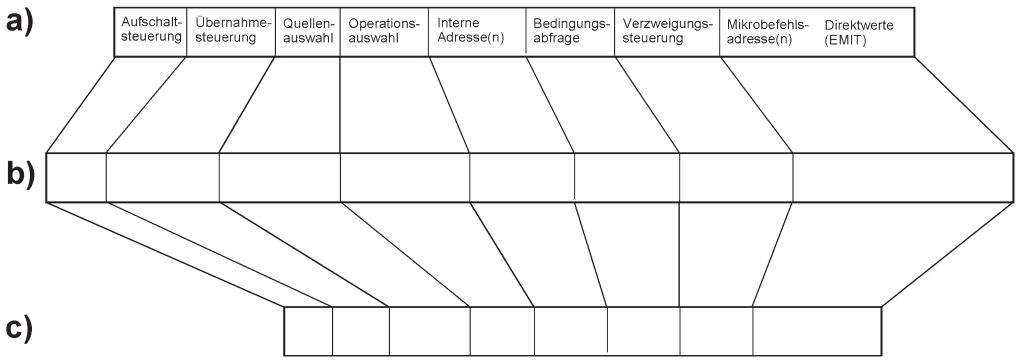


Abb. 4.34 Die Voreinstellung dient dazu, die Mikrobefehle kurz zu halten.

Wie aber kann man mit kurzen Feldern im Mikrobefehl auskommen? Abb. 4.35 zeigt typische Lösungen.

42. Prinzip der Ressourcenvektormaschine.
43. Und womöglich auch an vorgegebene Wortlängen, Zugriffsbreiten usw. angepaßt.

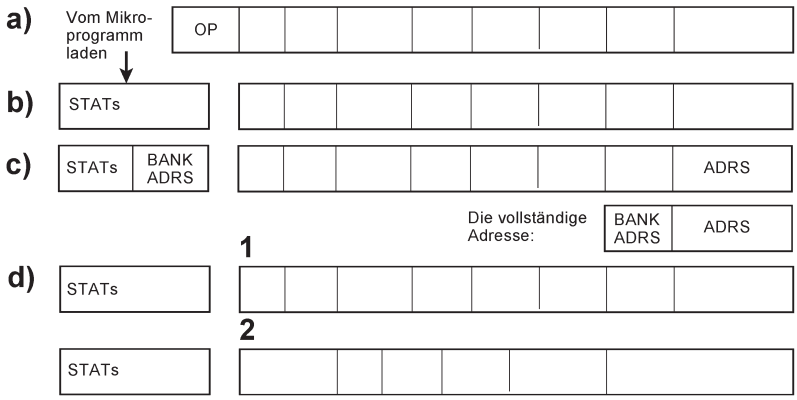


Abb. 4.35 Mit kurzen Feldern im Mikrobefehl auskommen. Typische Lösungen.

a) Man kann eine Operationscode (OP) vorsetzen. Dort ist codiert, daß es sich um einen Mikrobefehl für die Funktionseinheit F1 handelt oder um einen Mikrobefehl für Speicherzugriffe, zur Unterbrechungsbehandlung usw. Der Operationscode ist aber in jedem einzelnen Mikrobefehl erforderlich. Auch muß er in jedem Mikrobefehlszyklus von neuem decodiert werden. Das erhöht die Schaltungstiefe und verlängert die Durchlaufzeit.

b) Die Alternative: Voreinstellungen in Registern und Flipflops. Sie gelten nicht nur für den einzelnen Mikrobefehl, sondern für einen ganzen Mikroprogrammablauf. Die Voreinstellungen liegen in den nachfolgenden Mikrobefehlszyklen statisch an. Im wesentlichen handelt es sich nur um ein Durchschalten und Auswählen von Signalwegen[44], so daß die Durchlaufzeiten vergleichsweise kurz bleiben. In jeder Voreinstellung muß zumindest ein Mikrobefehlsformat vorgesehen sein, mit dem man die Register und Flipflops der Voreinstellung neu laden kann.

c) Adreßfelder werden mit dem Inhalt sog. Bankregister verlängert. So kann man mit einem kurzen Adreßfeld im Mikrobefehl in einen größeren Adreßraum hineinsehen[45].

d) Im Format b) bleibt die Feldaufteilung in allen Voreinstellungen gleich, nur die dort untergebrachten Funktionen, Adressen, Direktwerte usw. sind verschieden. Hier hingegen soll gezeigt werden, daß die Voreinstellungen auch das gesamte Mikrobefehlsformat betreffen können. Die Mikrobefehle der Formate 1 und 2 haben unterschiedlich viele und unterschiedlich lange Felder.

In den mikroprogrammgesteuerten Maschinen der Vergangenheit wurde vom Voreinstellen ausgiebig und nicht selten trickreich Gebrauch gemacht. Jedes Mikrobefehlsbit hat gezählt. Es war

44. Wofür man – wenn es auf kürzeste Durchlaufzeiten ankommt – beispielsweise Transfer-Gates anstelle der üblichen Logikbaustufen verwenden könnte. Womöglich passen die Verknüpfungen zum Durchschalten und Auswählen sogar in Logikzellen, die ohnehin zu durchlaufen sind.

45. Die Adreßverlängerung mit Bankregistern ist ein altbewährtes Prinzip in der Rechnerarchitektur (vgl. u. a. [56]).

entweder eine Vorgabe seitens der besonderen Speichertechnologie[46] oder der Zwang, sich an bestimmte Formate zu halten[47]. Wenn solche Entwurfsvorgaben sozusagen eingefroren sind, dann aber die Aufgabe steht, Fehler zu beseitigen oder neue Funktionen zu implementieren, bleibt gar nichts anderes übrig, als auf Biegen und Brechen zu tricksen[48].
Wir streben hier eine Art Mikro-Architektur an, großzügig dimensionierte Lösungen, die überschaubar bleiben und in denen nicht jedes Bit dreimal umgedreht werden muß. Sie müssen aber auch kostengünstig sein. Extrem lange Mikrobefehle und Mikroprogrammspeicher mit extremen Zugriffsbreiten kann man sich auch beim aktuellen Stand der Technologie nicht leisten. In Einsatzfällen, wie sie in den Abb. 4.18 und 4.33 skizziert sind, liegt es nahe, das Prinzip der Voreinstellung aufzugreifen.

4.6 Nanoprogrammsteuerung und Nanobefehle

Was soll ein Mikrobefehl steuern? Ist alles in einem Taktzyklus – womöglich mit einer einzigen Taktflanke – zu schaffen? Wenn man den Mikrobefehl kurz hält, muß man mehr decodieren. Wie wirkt sich das auf die Schaltungstiefe aus? Solche Überlegungen haben auf den Gedanken geführt, auch die Ausführung der Mikrobefehle mit einem speicherprogrammierbaren Steuerwerk zu steuern (Abb. 4.36). Zu den Begriffen Nanoprogramm, Nanobefehl und Nanoprogrammsteuerung ist es dann nicht weit.
Zwischen dem Maschinenbefehl und der Hardware liegt eine weitere Schicht der speicherprogrammierten Steuerung (Abb. 4.37 und 4.38). Der Ablauf des aktuellen Maschinenbefehls wird von einem Mikroprogramm gesteuert, die Ausführung des einzelnen Mikrobefehls von einem Nanoprogramm. Die typischen Nanobefehle sind extrem breite horizontale Mikrobefehle. Deren Bits wirken direkt auf die zu steuernden Funktionseinheiten, ohne daß dazu Code- und Anweisungsfelder aufwendig decodiert werden müssen. Es liegt nahe, Nanobefehle so auszulegen, daß sie tatsächlich alles das steuern, was jeweils mit einer einzigen Taktflanke zu erledigen ist. Auch die Adressierung ist auf kürzeste Verzögerungszeiten ausgelegt. Eine Einfachlösung besteht darin, daß die Adresse aus dem Nanobefehl direkt zum Steuerspeicher geführt wird, wobei Bedingungssignale in ausgewählte Adreßbitpositionen einfließen. Nanoprogramme sind nicht unterbrechbar; Mikroprogrammunterbrechungen bleiben anhängig, bis das jeweils laufende Nanoprogramm zu Ende gekommen ist.
Die Nanoprogrammsteuerung arbeitet typischerweise nicht als autonomer Automat, sondern nach dem Start-Stop-Prinzip. Der Mikrobefehl wählt das zugehörige Nanoprogramm aus und wartet, bis es abgelaufen ist.

46. Z. B. eines Transformatorspeichers (TROS = Transformer Read-Only Memory). Ist so ein Apparat durchkonstruiert, kann man nicht schnell mal ein paar Bits hinzufügen ...
47. Beispielsweise dann, wenn die Mikroprogramme im Arbeitsspeicher untergebracht waren.
48. Wer selbst vor solchen Aufgaben gestanden hat, weiß, wovon er redet. Er meint das nicht als Kritik, sondern als Anerkennung ...

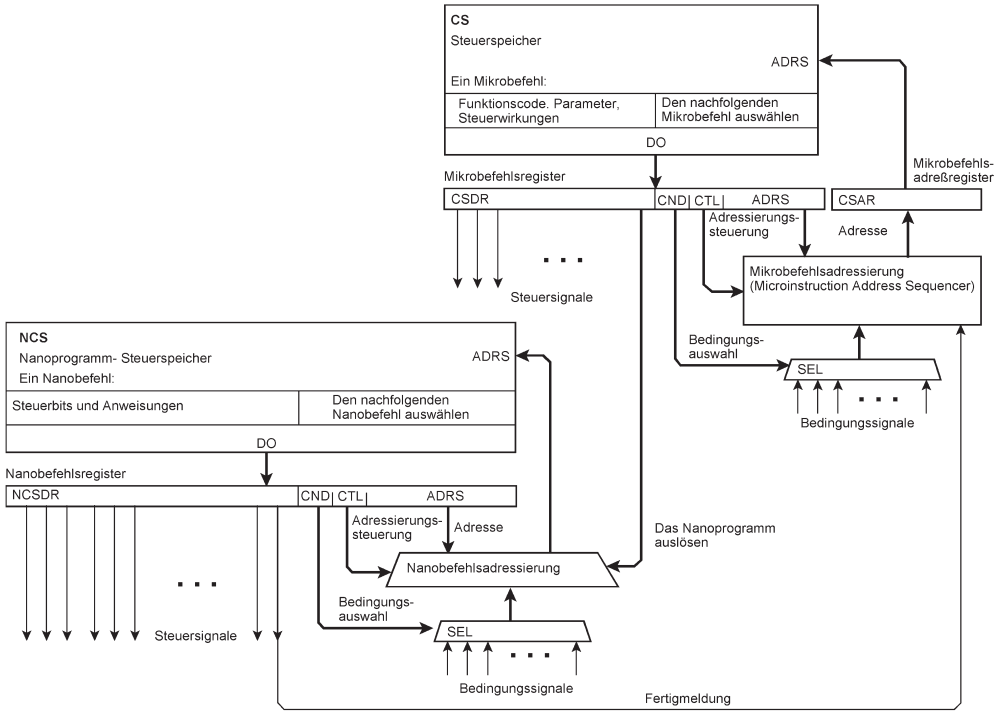


Abb. 4.36 Ein Mikroprogrammsteuerwerk mit nachgeschalteter Nanoprogrammsteuerung. Funktionscodes im Mikrobefehl lösen das zugehörige Nanoprogramm aus. Der Mikrobefehl verharrt solange in einem Wartezustand, bis das Nanoprogramm abgelaufen ist (Fertigmeldung).

Nanoprogrammierung zur Aufwandsverminderung

Die Mikrobefehle werden kompakt codiert. Das ergibt ein kostengünstiges Mikroprogrammsteuerwerk. Die Decodierung kann aber vergleichsweise aufwendig sein, vor allem dann, wenn eine umfangreiche Schaltung zu steuern ist, die viele Steuersignale benötigt. Deshalb schalten wir ein Nanoprogrammsteuerwerk nach. Der Mikrobefehl adressiert den jeweils ersten Nanobefehl im Nanoprogrammspeicher. Die Nanobefehle liefern die Steuersignale 1:1 an die zu steuernden Schaltungen. Damit nutzen wir praktisch den Adreßdecoder des Nanoprogrammspeichers als Mikrobefehlsdecoder.

Nanoprogrammierung und Mikrobefehlsarchitektur

Der Maschinenbefehl ist ein Funktionsaufruf. Dessen Parameter sind in der Maschinenarchitektur spezifiziert. Das Mikroprogramm führt diese Funktion mit den aktuellen Parametern aus. In Weiterführung dieses Gedankens – des Mikroprogramms als Implementierung einer Funktion – liegt es nahe, maschinenunabhängige Mikrobefehlsarchitekturen zu entwickeln, die für universelle Emulationszwecke und zur Anwendungsprogrammierung geeignet sind. Die Mikrobefehle

sind kompakt codiert, aber nicht genau auf die zu steuernden Schaltungen abgestimmt. Beispielsweise können sie ein Arithmetik-Logik-Einheit (ALU) anweisen, Funktionen ADD, SUB, AND, OR usw. auszuführen. Sie enthalten dafür ein Funktionscodefeld, aber keine Steuerbits und Anweisungen, die die Signale der ALU in der jeweiligen Schaltung erregen. Das ist vielmehr Sache der Nanobefehle. Die Mikrobefehle sind für alle Maschinentypen der Architektur gleich, die Nanobefehle sind an die Schaltungen des jeweiligen Maschinentyps angepaßt.

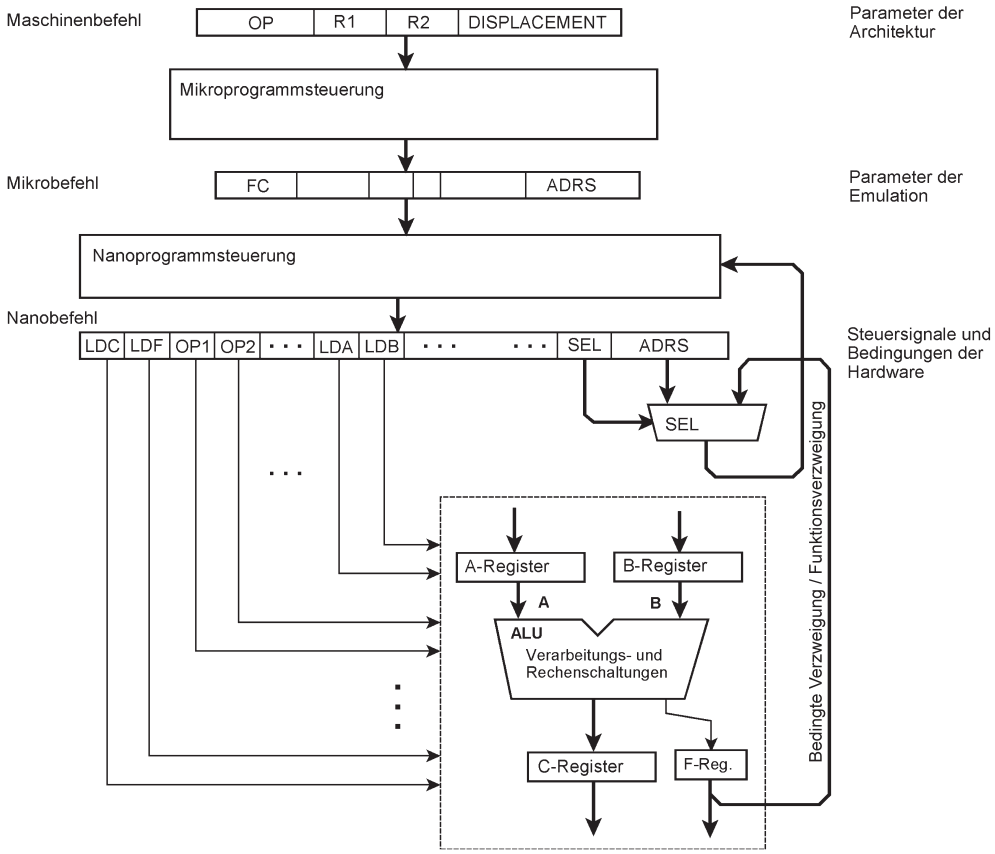


Abb. 4.37 Ein zweistufiges Schichtenmodell der speicherprogrammierbaren Steuerung einer Universal- oder Spezialmaschine.

Die zu steuernde Hardware braucht Steuersignale, die in jedem einzelnen Taktzyklus sagen, was zu tun ist. Die Frage ist, wie man diese mit annehmbaren Aufwendungen gewinnen kann. Das jeweilige Steuerwort – Maschinenbefehl, Mikrobefehl, Nanobefehl – weist die zu erbringenden Wirkungen an. Ein Signal, das eine Steuerwirkung in einem einzelnen Taktzyklus ausüben soll, muß direkt aus einem Flipflop kommen oder durch kombinatorische Decodierung gewonnen werden. Wenn das in einer bestimmten Schicht nicht möglich ist, fügen wir eine weitere hinzu.

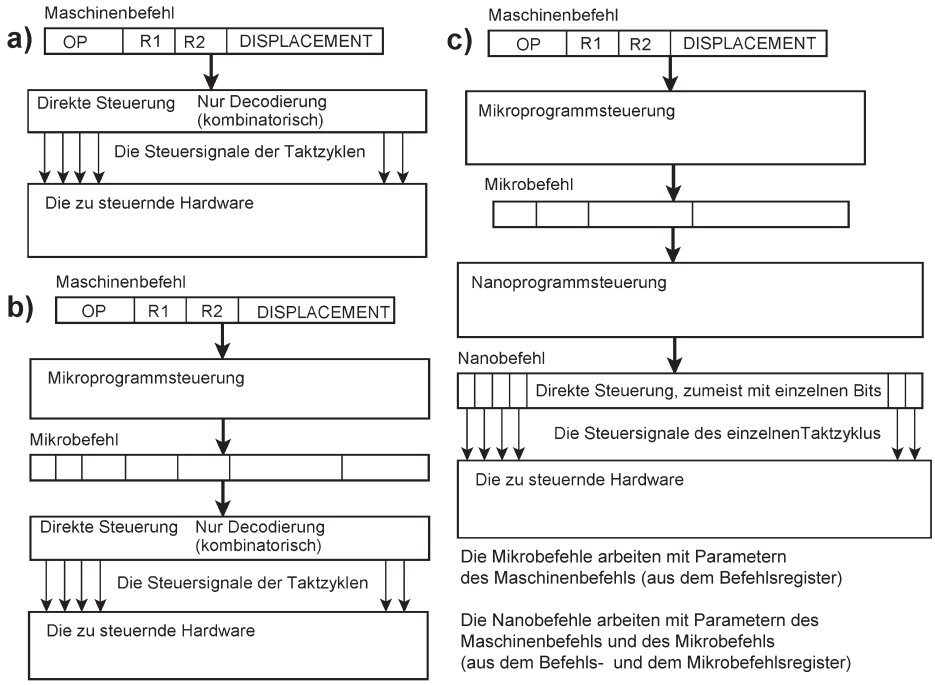


Abb. 4.38 Schichtenmodelle der Befehlsablaufsteuerung[49].

a) Die Maschinenbefehle selbst können direkt gesteuert werden. Das betrifft beispielsweise RISC-Befehle in Zusammenhang mit einer entsprechend ausgelegten Hardware (eigene Signalwege für alle Informationsflüsse, entsprechend viele Speicherzugriffswege usw.; vgl. beispielsweise Abb. 4.1).
b) Die Maschinenbefehle können nicht in einem einzigen Zyklus erledigt werden[50]. Deshalb wird ein Mikroprogrammsteuerwerk nachgesetzt.
c) Die Ausführung der Mikrobefehle wird mit Nanobefehlen gesteuert. Eine solche Lösung kann zweckmäßig sein, um den Aufwand zu vermindern oder um eine hardware-unabhängige Mikrobefehlsarchitektur zu unterstützen.

4.7 Maschinenbefehle und Mikrobefehle

Der typische Operationsbefehl eines Universalrechners bildet aus zwei Operanden ein Ergebnis sowie einige Bedingungsbits (Flagbits oder Bedingungscodes). Operandenverknüpfungen, die nicht in den Maschinenbefehlen vorgesehen sind, müssen mit mehreren Befehlen ausprogram-

49. Im großen und ganzen ist das vergleichbar zu den Schichtenmodellen der Kommunikationstechnik.
50. Beispielsweise weil es nur einen Zugriffsweg, ein Bussystem o. dergl. gibt. Vgl. auch Abb. 4.14.

miert werden. Befehlsformate und Befehlswirkungen sind im Rahmen der Maschinenarchitektur definiert. Beim Architekturentwurf strebt man an, eine nicht nur brauchbare, sondern vorteilhafte Anwendungsprogrammschnittstelle (API) anzubieten. Sie soll universell und erweiterungsfähig sein. Tabelle 4.3 gibt einen Überblick über die sozusagen klassischen Anforderungen. Die Architektur soll irgendwie elegant erscheinen, Zukunftsaussichten bieten und sich bequem programmieren lassen. In der ersten Zeit der Entwicklungsgeschichte ging es um die Bequemlichkeit der Maschinenprogrammierung (Assembler), später darum, effektive Compiler schreiben zu können. Auf die Eleganz der Maschinenbefehle, Orthogonalität, Symmetrie usw. kam es dann dann nicht mehr an. Die Maschinenprogramme sollten ja vom Compiler erzeugt werden. Dem konnte man schon einiges zumuten. Später gingen die Anforderungen dahin, die Maschinen mit extrem hohen Taktfrequenzen zu betreiben und effektive Befehlspipelines zu implementieren (RISC-Architekturen).

Anforderung	Erläuterungen und Anmerkungen
Konzeptionelle Einheitlichkeit	Eine einheitliche, klar erkennbare Philosophie für alles. Gegenbeispiel: die einen Funktionen mit Stack, andere mit Universalregistern, wieder andere mit Einzweckregistern
Orthogonalität	Die einzelnen Funktionsmerkmale sind unabhängig voneinander. Alles ist mit allem kombinierbar (alle Adressierungsarten für alle Datentypen). Gegenbeispiel: manche Adressierungsarten gibt es nur für ganze Binärzahlen, nicht aber für Gleitkommazahlen
Symmetrie	Gleichartige Funktionen für alle Datentypen, die dafür in Betracht kommen. Gegenbeispiel: Multiplikation nur für 16 Bits, nicht für 32 Bits, obwohl in der Architektur 32-Bit-Wörter definiert sind
Angepaßtheit an die Anforderungen der Nutzer	Versteht sich von selbst. Sonst handelt man sich schlechte Ratings ein und verkauft weniger
Optimierung der technischen Mittel gemäß der Nutzungshäufigkeit	Für Funktionen, die oft durchlaufen werden (wie die Binäraddition und die bedingte Verzweigung) werden zusätzliche Schaltmittel eingesetzt, um sie zu beschleunigen. Funktionen, die nur selten aufgerufen werden, haben Zeit (brauchen also keinen Zusatzaufwand). Frage von Kosten und Nutzen
Vorkehrungen für Erweiterungen	Nichts ist für die Ewigkeit, es kommt immer wieder Neues. Wenn man es sich aber grundsätzlich verbaut hat, ans Bewährte anzuknüpfen, wird es ärgerlich. Also: Reserven vorsehen
Unabhängigkeit von Implementierung und Technologie	Um programmkompatible Baureihen, Familien usw. anbieten und neue Fertigungsverfahren, Technologien usw. nutzen zu können

Tabelle 4.3 Traditionelle Anforderungen an Rechnerarchitekturen.

Der einzelne Maschinenbefehl ist seiner grundsätzlichen Wirkung nach überschaubar und universell einsetzbar. Im Idealfall kann er die gesamte Maschine ansprechen (alle Register, den gesamten Adreßraum usw.). Der typische Maschinenbefehl weist eine Operationscode und eine Adresse oder einen Direktwert auf.

Der Mikrobefehl kann nur die zu steuernden Schaltungen ansprechen. Was der Mikrobefehl letzten Endes bewirkt – ob weniger als ein üblicher Maschinenbefehl oder mehr – hängt davon ab, welche Schaltungen er steuert. Typische Mikrobefehle bestehen aus Bitfeldern und einzelnen Steuerbits, die teils im Verbund, teils unabhängig voneinander wirken. Somit kann ein einziger Mikrobefehl vielfältige Kombinationen von Steuerwirkungen gleichzeitig auslösen (analytischer Befehlscode).
Traditionell wurde gespart. Die typische Aufgabe eines Mikroprogramms ist die Steuerung der Ausführung eines Maschinenbefehls. Solche Mikroprogramme sind vergleichsweise kurz. Sie nutzen nur selten die gesamte Maschine, sondern vor allem die Funktionseinheiten, die zur Ausführung des jeweiligen Maschinenbefehls erforderlich sind. Deshalb können die Mikrobefehle womöglich mehreres gleichzeitig auslösen, aber nicht alles.

Reserven lassen
Es ist ein Gebot der elementaren Vernunft, nicht alles bis zum letzten Bit oder Bitmuster auszunutzen. Die Ausgestaltung ist eine Frage der Optimierung und Feinabstimmung. Nicht selten spricht das Marketing ein gewichtiges Wort mit[51]. Wir wollen uns hier auf einige kurze Anmerkungen beschränken:

- Eine naheliegende Lösung: Großzügig dimensionieren. Das betrifft die Parameter, die ein Befehl mitbringt, und die Anweisungsfelder der Mikrobefehle. Wenn sich beispielsweise aus dem Feinentwurf der zu steuernden Schaltungen 7 Steuerbits und 13 auszuwählende Register ergeben, ist es sicherlich nicht falsch, 12 Steuerbits vorzusehen und 5 Bits zur Registerauswahl.
- Eine Alternative: Grundsätzlich zwischen Alt und Neu umschalten. Beispiel einer Einfachlösung: das Escape-Bit. Ist das Bit = 0, so ist es die herkömmliche Architektur, ist es = 1, so ist es etwas Neues.
- Eine erste historische Faustregel (IBM): Formate für Binärzahlen – also Adressen, Anzahlen, Auswahlcodes usw. – so dimensionieren, daß eine Erweiterung auf das Vierfache möglich ist, also zwei Bits Reserve.
- Manchmal werden aber auch die vorsorglich eingebauten Reserven nicht genutzt. Statt dessen werden die Erweiterungen von Grund auf neu eingeführt[52].
- Eine zweite historische Faustregel ([94]): Eine gute Architektur übersteht wenigstens eine grundsätzliche Erweiterung.
- Die Praxis zeigt, daß sich eine noch weitergehende Vorsorge – über die nächste Generation hinaus – nur selten lohnt. Erstens kommt es anders, zweitens als man denkt ... Es ändern sich die Anforderungen, die Erwartungen der Nutzer, die Märkte, die Technologien usw. – und es wird auch immer wieder Neues erfunden. Also ist es manchmal doch besser, von frischem anzufangen (Clean Slate).

51. Vor allem bei der Frage kompatibel, koste es, was es wolle, oder doch besser etwas ganz Neues?
52. Historisches Beispiel: Das Bildschirmsystem IBM 3270.

Anwendungsprogrammierung mit Mikrobefehlen
Implementiert man Programmabläufe der Anwendungsaufgaben nicht mit den Maschinenbefehlen der Zielmaschine, sondern mit deren Mikrobefehlen, so erhöht sich die Arbeitsgeschwindigkeit (Speedup). Es ergeben sich aber zwei kritische Fragen:

1. Der Arbeitsaufwand. Das Schreiben von Anwendungsprogrammen für Mikroprogrammsteuerwerke ist Maschinenprogrammierung mit hohem Schwierigkeitsgrad. Handelt es sich um eine große, komplexe Maschine, so kann ein erfahrener Programmierer kaum mehr als einige hundert Mikrobefehle im Jahr bewältigen[53]. In der üblichen Anwendungsprogrammierung mit ihren harten Termin- und Kostenvorgaben[54] dürfte das kaum in Betracht kommen .
2. Die Zukunftssicherheit. Was geschieht beim Übergang auf einen anderen Maschinentyp?

Die Vorteile der Mikrobefehle gegenüber den üblichen Maschinenbefehlen:

- Man hat die Hardware in jedem Maschinentakt unter Kontrolle und kann somit das Leistungsvermögen der Maschine voll ausnutzen.
- Man kann auf Bedingungen, die sich in der Hardware ergeben, besonders schnell reagieren (Verzweigungen, Unterprogrammaufruf usw.). Wenn man mit üblichen Maschinenbefehlen programmiert, kann man solche Bedingungen nur erkennen, indem man die jeweiligen Bitmuster in die Verarbeitungseinheit (ALU) transportiert und dort mit Verarbeitungsbefehlen auswertet. Dann kann man auch nur auf die wenigen Bedingungen verzweigen, die die ALU erkennt und als Flagbits oder Bedingungscodes bereitstellt.
- Man kann Mikrobefehlsfunktionen implementieren, die über die Wirkungen der üblichen Maschinenbefehle hinausgehen, wie direkte Steuerwirkungen in den peripheren Schnittstellen, Verzweigungen in mehrere Richtungen und Unterprogrammaufrufe, die keine zusätzlichen Maschinenzyklen (Overhead) kosten.

Mikroprogramme sind schneller als Maschinenprogramme (Speedup)
Um wieviel, hängt davon ab, was man miteinander vergleicht und was man ins Mikroprogramm verlagert. Einige Richtwerte[55]:

- Für kurze Befehlsfolgen: 10:1 bis 100:1.
- Für größere Programmteile (Mittelwert über alles): 2:1 bis 5:1.
- Leistungskritsche Betriebssystemfunktionen einer CISC-Architektur (MP Assists): 2:1 bis 40:1.

53. Das betrifft einsatzfähige Mikroprogramme einschließlich Test und Dokumentation.
54. Im Unterschied zu Forschungsprojekten und zur Systemprogrammierung.
55. Erfahrungswerte (die oftmals nur in internen Forschungsberichten oder gar nicht publiziert wurden). [77] ist eine der wenigen offen zugänglichen Quellen.

RISC-Befehle und Mikrobefehle

Auch die universellen RISC-Prozessoren können die meisten Befehle in einem einzigen Maschinenzyklus ausführen. Aufgrund ihrer überschaubaren, regulären Struktur sind sie bequemere Zielmaschinen für Compiler. Verglichen mit Mikroprogrammsteuerwerken weisen herkömmliche RISC-Prozessoren aber typische Leistungsschwächen auf:

- RISC-Befehle leisten auch nicht mehr als die Befehle der herkömmlichen Einadreß- und Universalregistermaschinen; eine einzige Operandenverknüpfung, ein einziger Datentransport, eine einzige Verzweigung usw.
- Informationstransporte und Verarbeitungsfunktionen sind mit getrennten Befehle auszuprogrammieren (Load-Store-Architektur). Erst die Operanden in Prozessorregister laden, dann die Registerinhalte miteinander verknüpfen, dann die Ergebnisse speichern oder ausgeben. Das Prinzip ergibt einfache, schnell laufende Maschinen, hat aber zur Folge, daß man deutlich mehr Befehle braucht.
- Verzweigungen kosten Zeit. Sie sind weniger effektiv als höher entwickelte Mikrobefehlsverzweigungsverfahren (Funktionsverzweigung, Spätverzweigung usw.).
- Die E-A-Einrichtungen sind vom Prozessorkern aus nur über E-A-Befehle erreichbar. Im Vergleich zu einem Mikroprogrammsteuerwerk, das die E-A-Schaltungen direkt beeinflussen und abfragen kann, sind die Latenz- und Reaktionszeiten länger.

Befehls- und Datencaches mit parallelen, unabhängigen Zugriffswegen machen aus der v. Neumann-Maschine eine Art Harvard-Maschine (Abb. 4.39). Die RISC-Befehle entsprechen dann den vertikalen Mikrobefehlen, der Befehls-Cache entspricht dem Mikroprogrammspeicher.

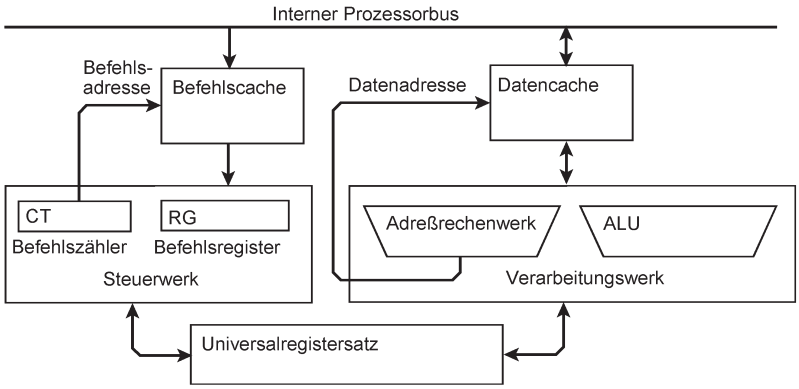


Abb. 4.39 Ein RISC-Prozessor mit Befehls- und Datencaches.

Sowohl RISC-Befehle als auch Mikrobefehle weisen Operationen, Transporte usw. an, die im aktuellen Maschinentakt ausgeführt werden. RISC-Operationen sind Operationen mit Registerinhalten. Die Ein- und Ausgabe erfordert zusätzliche Befehle. Beschleunigungsmaßnahmen (Caches, mehrere Verarbeitungswerke, spekulative Ausführung, Sprungzielvorhersage usw.)

wirken nur im Sinne von Erwartungswerten. Sie sind damit programmabhängig, manche auch datenabhängig. Das Realzeitverhalten ist keineswegs bis auf den Maschinentakt genau vorhersagbar. Daß man solche Prozessoren in industriellen Steuerungsanwendungen erfolgreich einsetzen kann, liegt vor allem an den extremen Taktfrequenzen. Damit ergeben sich genügend Zeitreserven, um auch harte Realzeitanforderungen der Anwendungsumgebungen zu erfüllen. Manche RISC-Befehle sind den vertikalen Mikrobefehlen sehr ähnlich. Manchmal ist es nur eine Bezeichnungsfrage. Womöglich kann man durch systematisches Weiterentwickeln vom Mikrobefehl aus zu effektiven Strukturen von Maschinen kommen, die sich u. a. als E-A-Prozessoren eignen. Beispiele dazu in Kapitel 6.

4.8 Mikrobefehle als Grundlage der Maschinenarchitektur

Dieser Gedanke wurde im Laufe der Entwicklungsgeschichte immer wieder aufgegriffen. Die Mikrobefehle werden der Anwendungsprogrammierung zur Verfügung gestellt, die Mikroprogrammsteuerung gehört zur Anwendungsprogrammschnittstelle der Architektur. Es gibt zwei grundsätzliche Möglichkeiten, Mikrobefehle in der Anwendungsprogrammierung auszunutzen:

1) als Unterstützungsfunktionen im Rahmen einer gegebenen Architektur (Microprogrammed Assists),
2) als eigenständige Maschinenarchitektur (Micro-Architecture). Die Mikrobefehle sind die grundsätzliche Anwendungsprogrammschnittstelle (API) der Maschine.

Mit dem Aufkommen der RISC- und der Superskalarmaschinen sind diese Entwicklungsrichtungen aus der Mode gekommen:

- RISC-Maschinen haben gar keine Mikroprogrammsteuerung.
- Die Mikrobefehle der CISC-Superskalarprozessoren dienen vor allem dazu, die Ausführung einfacher Befehle in mehreren Verarbeitungswerken zu steuern[56]. Sie sind somit als frei nutzbare Anwendungsprogrammschnittstelle (API) weder vorgesehen noch geeignet.
- Universelle RISC-Prozessoren mit Caches und mehreren Verarbeitungswerken (Superskalarmaschinen) sind praktisch ebenso schnell wie herkömmliche Mikroprogrammsteuerwerke, die zum freien Programmieren geeignet sind. Zudem sind es angenehmere Ziele für Compiler.

Leistungsvergleiche sind nur sinnvoll, wenn man gleiche Technologien, Gatterverzögerungszeiten usw. annimmt. Auch gibt es bei den Mikroprogrammsteuerwerken beträchtliche Unterschiede in Hinsicht auf Leistung und Aufwand. Die kleineren Maschinen der Vergangenheit hatten vergleichsweise einfache Mikroprogrammarchitekturen, zumeist mit vertikalen Mikrobefehlen. Ein typischer Mikrobefehl, den man zum Schreiben von Anwendungsprogrammen bequem nutzen könnte, hat kaum mehr geleistet als ein typischer RISC-Befehl.

56. Ein solcher Befehl braucht (Richtwert) maximal vier Mikrobefehle. Viele Befehle kommen mit einem einzigen aus. Der Mikrobefehl steuert nur die Datenwege und die Schaltnetze der Operandenverknüpfungen.

Mikroprogramme zur Unterstützung und Beschleunigung (Microprogrammed Assists)
Die architekturseitige Anwendungsprogrammschnittstelle (API) beruht weiterhin auf den üblichen Maschinenbefehlen. Es ist aber möglich, Programmabläufe nicht mit Maschinenbefehlen, sondern mit Mikrobefehlen zu implementieren. Um dies zu unterstützen, muß die gegebene API um Maschinenbefehle erweitert werden, die die anwendungsspezifischen Mikroprogrammabläufe mit Parametern versorgen und starten. Zudem ist eine entsprechende Mikroprogrammspeicherkapazität bereitzustellen und zu adressieren[57].

Mikrobefehle als Anwendungsprogrammschnittstelle (Micro-Architecture)
Die Maschine ermöglicht es den Anwendern, eigene Mikroprogramme auszuführen. So kann man eigene Befehlslisten implementieren, Anwendungsprogramme und Interpreter für höhere Programmiersprachen als Mikroprogramme schreiben usw. Das wird mit entsprechenden Entwicklungsystemen unterstützt. Womöglich kann man sogar ganz auf Maschinenbefehle verzichten und Anwendungsprogramme direkt in Mikroprogramme übersetzen.

Mikrobefehle in der Maschinenarchitektur
Die mikroprogrammierten Unterstützungsfunktionen (Assists) müssen hier nicht weiter diskutiert werden, da sie nur in CISC-Architekuren in Betracht kommen, und auch dann nur, wenn die Maschine von Hause aus ein Mikroprogrammsteuerwerk aufweist. Das ist ein seit langem bekannter Stand der Technik.
Hier hingegen soll der Gedanke verfolgt werden, Prinzipien der Mikroprogrammsteuerung aufzugreifen, um effektivere Anwendungsprogrammschnittstellen und neuartige Maschinenarchitekturen zu schaffen. Letzten Endes findet die Informationsverarbeitung immer in der Hardware statt. Deshalb kann es zweckmäßig sein, die Programmierabsicht der Anwendungsprogrammierung direkt in Steueranweisungen der Hardware umzusetzen.

Maschinen- und Mikrobefehle in herkömmlichen Architekturen
Es ist zweckmäßig, sich die grundsätzlichen Unterschiede klarzumachen (Tabelle 4.4). Daraus ergeben sich Anregungen, Mikrobefehle und Mikroprogrammsteuerwerke in Richtung auf echte, praxisbrauchbare Maschinenarchitekturen und Anwendungsprogrammschnittstellen weiterzuentwickeln.

Entwicklungsziele
Wofür, zu welchem Zweck soll ein Mikroprogrammsteuerwerk entworfen werden? Danach richtet sich die Ausgestaltung der Einzelheiten. Typische Einsatzfälle von Mikroprogrammsteuerwerken – und damit Entwurfsaufgaben – im Überblick:

- Steuerung einer gegebenen Schaltungslösung,
- Emulation einer gegebenen Maschinenarchitektur,

57. Für die erweiterte Speicherkapazität braucht man eine hinreichend lange Adresse, die auch im Mikroprogrammsteuerwerk unterstützt werden muß (Adreßfelder im Mikrobefehl, Segmentregister usw.).

- Emulation beliebiger Maschinenarchitekturen,
- Emulation abstrakter Sprachumgebungen (Interpreter),
- Anwendungsprogrammierung.

Ein erster Eindruck von den Anforderungen ergibt sich, wenn man vom Mikroprogrammsteuerwerk auf die zu steuernden Schaltungen und zu unterstützenden Adreßräume blickt (Abb. 4.40). Der Begriff des Blickwinkels ergibt eine bildhafte Vorstellung davon, wie ein Mikroprogrammsteuerwerk auszulegen ist. Ein weiter Blickwinkel bedeutet viele Signale, die zu erregen und abzufragen sind, lange Adressen usw. Wenn hingegen ein beschränkter Blickwinkel genügt, sind es nur wenige Signale, und die Adressen dürfen kurz sein (nicht länger als nötig, um das zu adressieren, worauf das aktuelle Mikroprogramm zugreifen muß). Alles weitere kann mit Voreinstellungen erledigt werden.

Maschinenbefehle	Mikrobefehle
• Maschinenbefehle sind Architekturmerkmale. Sie betreffen eine hardware-unabhängige Programmschnittstelle. • Befehle sind die programmseitig nutzbaren Funktionsaufrufe der Schnittstelle zwischen Software und Hardware (API). • Wieviele Maschinenzyklen die Ausführung eines Befehls dauert, hängt von der Art des Befehlsablaufs und der Implementierung der Hardware ab. • Der Maschinenbefehl sieht Funktionen und Einrichtungen der Architektur.	• Die Mikroprogrammsteuerung ist kein Architekturmerkmal, sondern Mittel zum Zweck. • Mikrobefehle betreffen eine schaltungsabhängige Programmschnittstelle, beschränkt auf die vergleichsweise überschaubare Aufgabe, Befehle und andere Funktionsabläufe zu steuern. • Der einzelne Mikrobefehl kommandiert zumeist einen einzigen Maschinenzyklus. • Der Mikrobefehl sieht Funktionen und Einrichtungen der Schaltung (Hardware).

Tabelle 4.4 Maschinen- und Mikrobefehle in herkömmlichen Architekturen.

Steuerung einer gegebenen Schaltungslösung

Die Mikrobefehle sind so zu formatieren und zu dimensionieren, daß die jeweilige Schaltung gesteuert werden kann. Man kann herangehen wie an die Ressourcenvektormaschine. Alle Steuer- und Bedingungssignale der zu steuernden Schaltungen werden ans Mikroprogrammsteuerwerk angeschlossen. Dann wird optimiert. Der Mikrobefehl muß steuern, was im einzelnen Maschinenzyklus abläuft. Alle hierfür nötigen Steuersignale müssen gleichzeitig erregt werden. Im Idealfall muß das Mikroprogrammsteuerwerk alle Bedingungen, die sich im aktuellen Maschinenzyklus ergeben, sofort auswerten und demgemäß den nächsten Mikrobefehl auswählen. Der Blickwinkel muß dies alles umfassen. Er ist somit anwendungsspezifisch. Was zu implementieren ist, ergibt sich aus den Anforderungen der Anwendungsumgebung. Wenn die zeitlichen Anforderungen nicht wirklich extrem sind, kann man oftmals Aufwand gegen Zeit tauschen, also schmale Mikrobefehle vorsehen, die einen entsprechend kleinen Blickwinkel haben, und die Voreinstellung entsprechend oft ändern.

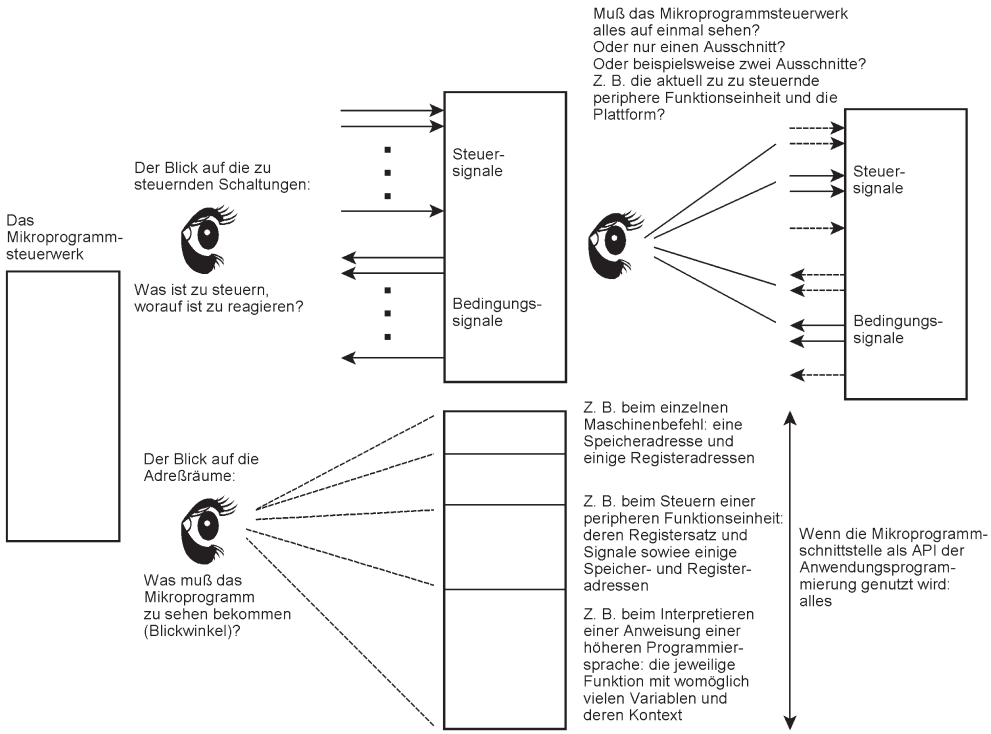


Abb. 4.40 Was muß ein Mikroprogrammsteuerwerk zu sehen bekommen, damit es seine Arbeit tun kann? Der Blickwinkel ist eine anschauliche Modellvorstellung.

Emulation einer gegebenen Maschinenarchitektur

Die Informationsstrukturen (Befehlsformate, Datentypen, Deskriptoren, Kommandowörter, Seitenumsetzungstabellen usw.) der Zielarchitektur müssen unterstützt werden. Viele Parameter der Mikroprogramme können somit Festwerte sein[58]. Das einzelne Mikroprogramm führt einen einzigen Befehl aus oder einen anderen in der Architektur vorgesehenen Ablauf, wie beispielsweise eine Unterbrechungsauslösung, eine Fehlerbehandlung oder eine Kontextumschaltung. Der Blickwinkel ist sehr beschränkt (Lokalität). Er umfaßt nur die wenigen aktuellen Parameter.

Emulation beliebiger Maschinenarchitekturen

Die Maschine muß die Informationsstrukturen beliebiger Zielarchitekturen unterstützen. Im Extemfall muß alles bis aufs Bit auflösbar sein (Bitadressierung, variable Bitfelder)[59]. Wenn die Zielarchitektur nicht allzu sehr vom Gewöhnlichen abweicht, muß das einzelne Mikropro-

58. Das Maschinenwort ist beispielsweise nicht beliebige x Bits, sondern stets 32 Bits lang, die Registeradressen stehen nicht irgendwo im Befehl, sondern auschließlich an bestimmten Bitpositionen usw.
59. Historisches Beispiel: Burroughs B1700 ([105], [106]; überblicksmäßige Kurzbeschreibung in [56]).

gramm nur jeweils einen einzigen Befehl oder eine anderen in der Architektur vorgesehenen Ablauf ausführen. Dann ist der Blickwinkel beschränkt (Lokalität) und umfaßt nur die wenigen aktuellen Parameter.

Emulation abstrakter Sprachumgebungen (Interpreter)
Wenn das Mikroprogramm eine herkömmliche Befehlsliste emuliert, muß es seine Parameter als Bits oder Bitfelder aus dem Befehlsformat herauslösen. Soll das nicht allzu viele Maschinenzyklen kosten, muß die Maschine variable Bitfelder usw. unterstützen. Hier hingegen führen die Mikroprogramme Anweisungen aus, die in einer höheren Programmiersprache formuliert wurden. Das können auch komplexere Funktionen sein. Hierzu braucht es womöglich viele Parameter und einen entsprechend weiten Blickwinkel. Die binäre Codierung, die Parameterübergabe usw. kann direkt auf die Mikroprogramm-Architektur abgestimmt werden. Der Compiler erzeugt keine herkömmlichen Maschinenbefehle, sondern Funktionsaufrufe[60].

Anwendungsprogrammierung
Die Mikroprogramm-Architektur ist die API der Anwendungsprogrammierung. Es sind also womöglich sehr große Mikroprogramme zu erstellen und auszuführen. Hierzu wird man sich einen entsprechenden Programmierkomfort wünschen, auch seitens der Compiler-Autoren[61]. Das Mikroprogramm soll das gesamte Anwendungsprogramm überblicken und mit den vielen Parametern zurechtkommen. Abb. 4.41 veranschaulicht, welche Vorteile man von solchen Architekturprinzipien erwarten kann. Viele Anwendungslösungen beruhen auf virtuellen bzw. fiktiven Maschinen[62]. Was von außen – beispielsweise übers Internet – in die eigentliche ausführende Maschine (Host-Maschine) geladen wird, sind keine Maschinenprogramme der Host-Architektur, sondern Programme fiktiver Maschinen, die typischerweise Industriestandards sind. Sie werden von Emulatorprogrammen interpretiert oder in Maschinenprogramme übersetzt – oftmals während des Ladens (Just-in-Time Compilation). Die Abläufe der Maschinenbefehle werden ihrerseits von Mikroprogrammen gesteuert[63]. Der Grundgedanke: wenn es ohnehin nur ums Interpretieren oder Compilieren geht, kann man die Ebene der Maschinenbefehle überspringen und die fiktive Maschine mit Mikroprogrammen direkt interpretieren oder deren Programme in Mikroprogramme übersetzen. Man hat eine Ebene gespart und kann zudem die Ressourcenausstattung so einrichten, wie man es als zweckmäßig ansieht.

60. Wobei er die Parameter in Registern oder einem Stack Frame ans Mikroprogramm übergibt, so daß sie das Mikroprogramm ohne Bitfeldtransporte usw. bequem nutzen kann (vgl. die Parameterübergabe beim Funktionsaufruf in höheren Programmiersprachen). Beispiel einer solchen Mikro-Architektur: MAX 2. Überblicksmäßige Kurzbeschreibung in [56], als (unvollständige) Dokumentation vgl. [103].
61. Man möchte den Quelltext zügig umsetzen und dabei nicht durch allzu kleinliche Einschränkungen gehindert werden. Ständig Bankregister und Staticizers umladen und auf die Eigentümlichkeiten der eingestellten Betriebsarten achten zu müssen, ist gelegentlich *sehr* lästig ...
62. "Virtuelle Maschine" ist eine verbreitete Bezeichnung. Der Begriff "fiktive Maschine" dient dazu, solche Architekturen von den virtuellen Maschinen zu unterscheiden, die mit den Befehlen der Host-Architektur arbeiten. Vgl. auch [56].
63. Es sei denn, es sind RISC-Befehle, die kein Mikroprogramm brauchen.

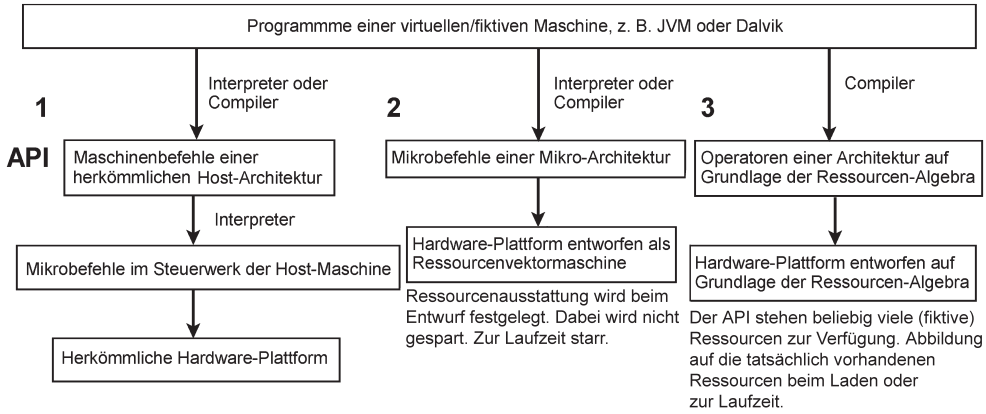


Abb. 4.41 Architekturprinzipien im Vergleich. 1 - herkömmlich; 2 - auf Grundlage einer Mikro-Architektur; 3 - auf Grundlage der Ressourcen-Algebra[64]. Wir befassen uns hier mit Prinzip 2.

Muß das Mikroprogrammsteuerwerk Turing-vollständig sein?

Es kommt darauf an, was es leisten soll. Wenn es Befehlsabläufe steuert oder Konstrukte höherer Programmiersprachen interpretiert, entspricht es dem Steuerautomaten der universellen Turingmaschine, der – für sich gesehen – nicht Turing-vollständig ist, weil er keinen frei nutzbaren Arbeitsspeicher hat. Die Turing-Vollständigkeit der gesamten Maschine ergibt sich erst aus dem Verbund mit dem Rechenwerk, den Adreßrechenschaltungen und dem Speicher. Als Steuerwerk eines universellen Prozessors muß es nicht Turing-vollständig sein. Praxisbrauchbar Turing-vollständig wird die Maschine nur durch die architekturseitigen Maschinenbefehle, die vom Mikroprogramm emuliert werden und – vor allem – durch den Arbeitsspeicher. Er entspricht dem Band der Turingmaschine, das sowohl das Anwendungsprogramm als auch die Daten enthält. Das Mikroprogrammsteuerwerk ist nur der Steuerautomat. Soll die Mikro-Architektur jedoch die eigentliche universelle Programmschnittstelle (API) darstellen, muß die Maschine anwendungsbrauchbar Turing-vollständig sein. Auch eine solche Maschine hat einen Arbeitsspeicher, Rechenschaltungen für die Operandenverknüpfungen und Adressierungsschaltungen, die die Adreßrechnung unterstützen. Die Mikrobefehle müssen direkten Zugriff auf alles haben und nicht erst über den Umweg der – zu emulierenden – Maschinenbefehle. Daß es möglich ist, auf dieser Grundlage wirklich universelle Maschinen zu bauen, kann mit Überlegungen zur Ausgestaltung fiktiver Turingmaschinen nachgewiesen werden[65]. Damit das Prinzip praxisbrauchbar wird, müssen die Mikrobefehle einen alles[66] umfassenden Blickwinkel aufweisen und entsprechend großzügig ausgelegt werden. Also nicht mit jedem Bit knausern ...

64. Vgl. [42] bis [45] sowie [I1].
65. Einzelheiten in [56].
66. Soll heißen: alle Adreßräume und zu steuernden Funktionseinheiten.

Einfache Mikroprogrammsteuerwerke oder Mikrocontroller?
Es liegt nahe, Zustandsautomaten nicht selbst zu entwerfen, sondern einen handelsüblichen Mikrocontroller zu verwenden und die Zustandsdiagramme auszuprogrammieren. Das Mikroprogrammsteuerwerk hat aber auch etwas für sich:

- Es weist unter allen Umständen ein exaktes Realzeitverhalten auf, bis auf den Maschinentakt genau.
- Übliche Programme können abstürzen oder – in entsprechenden Einsatzumgebungen (mit Vernetzung usw.) – von außen beeinflußt werden, einfache, starre Mikroprogramme nicht.
- Das Mikroprogrammsteuerwerk braucht weniger Schaltungsressourcen. Das ist dann von Bedeutung, wenn die gesamte Anwendungslösung in einem einzigen Schaltkreis untergebracht werden soll (SoC).
- Mikroprogramme laufen schneller als Maschinenprogramme (Speedup). Zumeist sind die Anforderungen der Anwendungsumgebung vorgegeben. Mit Ausnahme von Änderungs- und Erweiterungsreserven gibt es keinen Grund, diese Vorgaben zu überbieten. Der Umkehrschluß: für die gleiche Verarbeitungsleistung – um das geforderte Realzeitverhalten darzustellen – braucht man weniger Ressourcen, käme also, um das Anwendungsproblem zu lösen, mit einem kleineren FPGA aus, würde weniger Strom aufnehmen usw.
- Aufgrund der innigen Verbindung mit der Peripherie – da man alles aus einem Guß entwerfen kann – können die peripheren Schaltungen einfacher sein. Die peripheren Funktionen werden teilweise ins Mikroprogramm verlagert. Die Peripherie wird mehr soft, ist leichter zu debuggen, zu ändern usw.
- Man könnte mit niedrigeren Taktfrequenzen arbeiten, käme also womöglich mit FPGAs aus, die keine besonders hochentwickelte Taktverwaltung (Clock Management, Clock Distribution) aufweisen. Ein zentraler Taktgenerator und eine simple Taktverteilung dürften oftmals genügen. Auch wäre –da weniger Schaltvorgänge auftreten – die Stromaufnahme von Grund auf[67] geringer.
- Wenn nötig, werden Taktzyklen mit zwei oder mehr Taktphasen vorgesehen. Prinzip: Längere Zyklen, aber im einzelnen Zyklus mehr leisten.
- Ein sinnfälliges Entwickungsziel: Ein Mikrobefehl sollte – bei vergleichbaren Aufwendungen und bezogen auf die Anwendungsprogramierung – wenigstens das Doppelte eines typischen RISC-Befehls leisten[68].
- Das selbst entworfene Steuerwerk kostet keine Lizenzgebühren und darf beliebig geändert, portiert und weiterentwickelt werden.

67. Also ohne besondere Stromsparmaßnahmen.
68. Denn sonst brauchten wir gar nicht erst anzufangen ...