6. Maschinen mit Mikroprogrammsteuerung

Nach Turing genügt, um alles zu berechnen, was überhaupt berechnet werden kann, ein Steuerautomat, ein ausreichend großer Speicher und ein Vorwärts-Rückwärts-Adreßzähler. Ein Rechenwerk im üblichen Sinne braucht die Maschine gar nicht. Wenn es nichts zu rechnen gibt, kommt man mit einer solchen Maschine auch tatsächlich aus. Im Bereich der Embedded Systems verhält es sich aber oftmals umgekehrt. Die meisten der Anwendungsaufgaben, die im Grunde reine Steuerungsprobleme sind, werden dadurch gelöst, daß man Universalrechner (Mikrocontroller) entsprechend programmiert. Im folgenden soll die Alternative untersucht werden, den speicherprogrammierbaren Steuerautomaten Schritt für Schritt zu entwickeln und bedarfsweise mit Recheneinrichtungen zu erweitern. Hierzu wollen wir auf den grundsätzlichen Erläuterungen der Kapitel 1 bis 5 aufbauen und Anregungen aus der Entwicklungsgeschichte einfließen lassen.

Zeitgemäße Maschinen dürften vorzugsweise mit programmierbaren Schaltkreisen (FPGAs) implementiert werden. Die Lösung des Anwendungsproblems ergibt Funktionseinheiten, die mittels Strukturbeschreibung oder zeichnerisch (Schaltplan) miteinander verbunden werden. Diese – anfänglich noch grobe – Struktur wird so lange verfeinert, bis man zu Modulen kommt, die man nur noch aufrufen muß (Instantiation) bzw. deren Funktionsweise so überschaubar ist, daß man sie ohne weiteres mittels Verhaltensbeschreibung erfassen kann.

Das typische Entwurfsziel dürfte vor allem die kleinere Maschine sein, die als Steuerautomat in einer anwendungsspezifischen Schaltungslösung zum Einsatz kommt oder als Akzelerator, Coprozessor, E-A-Prozessor usw. an einen Universalrechner angeschlossen wird. Weiterführende Überlegungen zu Universal- und Hochleistungsmaschinen ergeben sich aus dem grundsätzlichen Bestreben, den aktuellen Stand der Technik zu übertreffen.

Die nachfolgenden Erläuterungen stellen eine Art Baukasten dar. Er enthält Wirkprinzipien und Entwurfsgedanken, die aufeinander aufbauen und miteinander kombiniert werden können. Sie werden vor allem anhand von Blockschaltbildern veranschaulicht. Die Schaltungen beruhen auf Prinziplösungen, die in Kapitel 5 erläutert wurden.

Unsere Schaltungen arbeiten mit durchlaufenden Takten[1]. In den meisten Blockschaltbildern sind, wie auch bereits in Kapitel 5, diese Taktsignale (CLK usw.) nicht dargestellt. Auch sind nicht immer alle Erlaubnissignale eingezeichnet. In unseren Blockschaltbildern hängen alle Steuersignale, die vom Mikrobefehl erregt werden, an den Ausgängen des Mikrobefehlsregisters CSDR. Wenn man sich für den Einphasentakt entschieden hat, muß man manche dieser Signale an die Ausgänge des Steuerspeichers anschließen. Wir werden nur dann eigens darauf hinweisen, wenn es von besonderer Bedeutung ist. Zudem sei auf die Überlegung verwiesen, nicht unbedingt auf kürzeste Zykluszeiten hin zu entwerfen, sondern womöglich mit zwei oder mehr Taktphasen zu arbeiten und im einzelnen – etwas längeren – Mikrobefehlszyklus mehr zu leisten.

1. Vgl. die Abschnitte 5.1.4 (S. 218) und 8.6 (S. 479).

Wir erläutern Aufbau und Wirkungsweise der Mikroprogrammsteuerwerke zunächst an Beispielen einfacher Maschinen. Somit bleiben die Darstellungen überschaubar. Die typische Universalmaschine ist ein Verbund aus einem Mikroprogrammsteuerwerk (als Sequencer zum Steuern der Befehlsabläufe), einem Rechenwerk, dem Speicher, den E-A-Schaltungen usw. Auf den programmierbaren Sequencer kommt es an. Dem Mikroprogrammsteuerwerk ist es im Grunde gleichgültig, was es steuert, eine Universalmaschine, eine Spezialmaschine oder eine Einzweckschaltung zum Lösen der jeweiligen Anwendungsaufgabe. Optimierungen sind Sache des Feinentwurfs. Sie müssen sich auf die jeweiligen Technologien oder Bauelementefamilien beziehen. Im Gegensatz zu den Maschinen der Entwicklungsgeschichte (Kapitel 9) müssen wir aber nicht mit jedem Bit, Flipflop und Gatter sparen.

6.1 Zeitplansteuerungen (Sequencers)

Zeitplansteuerungen sind Automaten, die Folgen von Bitmustern ausgeben. Man kann sie u. a. als sequentielle Schaltwerke, als speicherbasierte Automaten oder mit Software implementieren. Es liegt nahe, einen Speicher, der die auszugebenden Bitmuster enthält, Wort für Wort auszulesen. Ob man solche Maschinen als speicherbasierte Zustandsautomaten oder als Mikroprogrammsteuerwerke bezeichnet, ist womöglich Auffassungssache[2]. Hier wollen wir sie als besonders einfache Mikroprogrammsteuerwerke auffassen und darstellen.
Abb. 6.1 veranschaulicht den grundsätzlichen Aufbau eines Mikroprogrammsteuerwerks, das als einfache Zeitplansteuerung (Sequencer) arbeitet. Es kann Mikrobefehle nacheinander aufrufen, den Mikrobefehlsaufruf anhalten, zum ersten Mikrobefehl springen und mit dem Aufrufen des Nachfolgers warten. Es kann aber nicht bedingt verzweigen. Übergeordnete Steuereinrichtungen können den Ablauf starten, ggf. Anfangsadressen laden und abfragen, ob der Ablauf beendet ist.

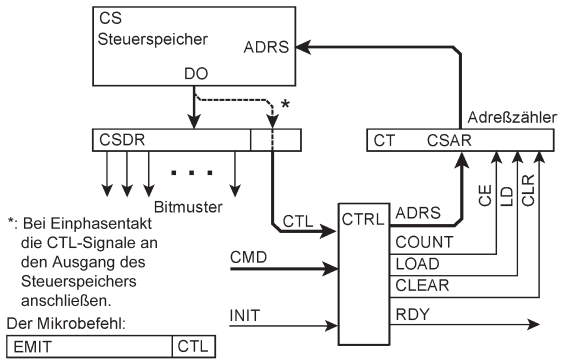


Abb. 6.1 Eine einfache Zeitplansteuerung (Sequencer) im Überblick.

2. Vgl. auch Abschnitt 2.2.2 (S. 65). Manche der folgenden Blockschaltbilder sind letzten Endes nur abgewandelte Darstellungen der Zustandsautomaten in den Abb. 2.41 bis 2.44.

Das Mikrobefehlsformat
Das Emit-Feld (EMIT) enthält die auszugebenden Bitmuster direkt oder es besteht aus Feldern, die das Setzen und Löschen von Flipflops, das Auswählen von Registern usw. anweisen. Zudem braucht man wenigstens eine Bitposition oder ein zu decodierendes Bitmuster für die Ablaufsteuerung (CTL). In der Entwurfspraxis wird man nur soviele Bitpositionen generieren, wie jeweils nötig sind, oder sich an die Datenformate der jeweils verfügbaren Speichermittel (z. B. Block-RAMs) anpassen.

Mikrobefehlsadressierung
Der Steuerspeicher wird Wort für Wort ausgelesen. Das Mikrobefehlsadreßregister CSAR ist ein Adreßzähler. Somit braucht man keine Adreßbits im Mikrobefehl; das gesamte Mikrobefehlswort steht für Steuerbits und -anweisungen zur Verfügung.

Mikroprogrammablaufsteuerung
Der Funktionsblock CTRL in Abb. 6.1 steht für alle nachfolgend beschriebenen Varianten der Ablaufsteuerung. Die Signale werden in den Tabellen 6.1 und 6.2 erläutert. In der Entwurfspraxis wird man nur die Steuerwirkungen generieren, die jeweils benötigt werden. Deshalb stellen wir hier auch keine Universalschaltung vor, sondern erläutern die Teilschaltungen einzeln. Die Verbindungen mit den übergeordneten Einrichtungen und der Anwendungsumgebung werden nicht näher beschrieben. Daß man bedarfsweise Synchronisierungsschaltungen, Pufferregister, Merkflipflops usw. vorsieht, gehört zu den Grundlagen des Schaltungsentwurfs und versteht sich von selbst.

Bitmuster einmalig ausgeben
Das INIT-Signal hält das Mikrobefehlsadreßregister anfänglich gelöscht (Anfangszustand; IDLE). Wird INIT inaktiv, beginnt die Adreßzählung, und die weiteren Mikrobefehle werden gelesen (Abb. 6.2). Die Adreßzählung endet, wenn die Mikroanweisung (oder das einzelne Mikrobefehlsbit) RDY gesetzt ist. Dann zeigt das Mikrobefehlsadreßregister ständig auf diesen letzten Mikrobefehl (Endzustand; READY). RDY dient zudem als Fertigmeldung, die von übergeordneten Einrichtungen ausgewertet werden kann. Wird INIT erneut aktiviert, gelangt der Sequencer wieder in den Anfangszustand, und die Bitmuster können erneut ausgegeben werden.

Der Anfangs- und der Endzustand
In der Schaltung von Abb. 6.2 können beide Zustände längere Zeit anliegen. Dabei wird der jeweilige Mikrobefehl immer wieder gelesen. Die Entwurfsabsicht ist, im Endzustand die Adresse des letzten Mikrobefehls – das ist der mit der RDY-Anweisung – beizubehalten, die Adresse also nicht nochmals zu erhöhen. Deshalb muß, wenn die Maschine mit einem Einphasentakt arbeitet, RDY an den Ausgängen des Steuerspeichers abgenommen werden. Ob man in den beiden Zuständen weitere Mikroanweisungen ausführt oder nicht, ist Sache der Entwurfsoptimierung und der Abstimmung mit der Anwendungsumgebung. Im einfachsten Fall sieht man in diesen Zuständen wirkungslose Mikrobefehle (NOPs) vor.

Signale	Steuereingang	Vorrang	Wirkung
CLEAR	CLR	1.	Das Mikrobefehlsadreßregister löschen (Mikrobefehlsadresse = 0)
LOAD	LD	2.	Laden der Mikrobefehlsadresse. Die Adresse (ADRS) wird von außen geliefert (Kommandoübertragung CMD)
COUNT ENABLE	CE	3.	Erhöhen der Mikrobefehlsadresse (Zählung +1)
Alle diese Signale inaktiv			Nichts tun. Mikrobefehlsadresse bleibt erhalten

Tabelle 6.1 So wird das Mikrobefehlsadreßregister CSAR gesteuert.

Signale	Wirkung
CTL	Steuerbits oder -anweisungen im Mikrobefehl
CMD	Laden der Mikrobefehlsadresse von außen (Kommandoübertragung). Die CMD-Signale umfassen die zu ladende Adresse und das zugehörige Steuersignal
INIT	Das Mikrobefehlsadreßregister löschen (INITIATE; Anfangszustand). Wird INIT inaktiv, beginnt die Adreßzählung
RDY	Fertigmeldung (READY)

Tabelle 6.2 Die weiteren Signale der Mikrobefehlsablaufsteuerung.

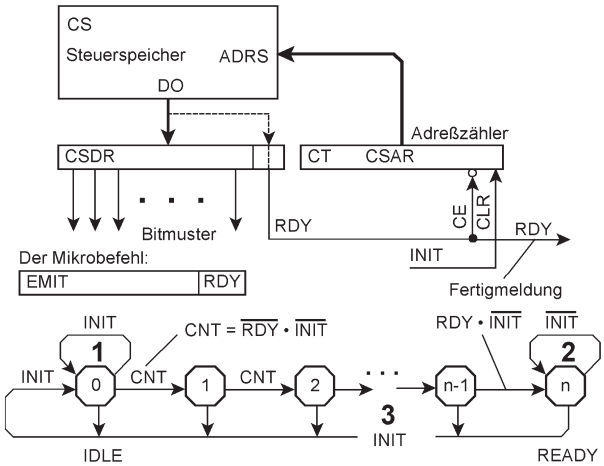
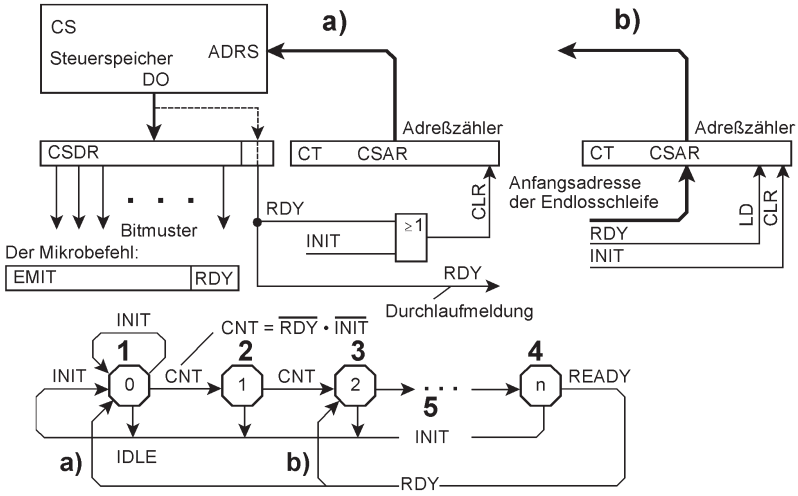


Abb. 6.2 Ein Mikroprogrammsteuerwerk (Sequencer) zum einmaligen Ausgeben von Bitmustern. 1 - Anfangszustand; 2 - Endzustand; 3 - INIT = Start- oder Rücksetzsignal von außen. INIT überführt die Maschine aus jedem beliebigen Zustand in den Anfangszustand.

Bitmuster zyklisch ausgeben

Das INIT-Signal hält das Mikrobefehlsadreßregister gelöscht (Anfangszustand; IDLE). Wird INIT inaktiv, beginnt die Adreßzählung, und die weiteren Mikrobefehle werden gelesen (Abb. 6.3). Ist im Mikrobefehl RDY gesetzt, beginnt die Adreßzählung von vorn. RDY dient zudem als Durchlaufmeldung, die von übergeordneten Einrichtungen ausgewertet werden kann. Wird INIT erneut aktiviert, gelangt der Sequencer wieder in den Anfangszustand.



1 Anfangszustand
2 Beispiel eines Zustands der Initialisierung
3 Der erste Zustand der Endlosschleife
4 INIT (von außen) überführt die Maschine aus jedem beliebigen Zustand in den Anfangszustand

Abb. 6.3 Ein Mikroprogrammsteuerwerk (Sequencer) zum zyklischen Ausgeben von Bitmustern. Hier sind zwei Varianten des zyklischen Umlaufs dargestellt. Im Fall a) führt RDY zum Anfangszustand, im Fall b) zum erste Zustand der Endlosschleife.

a) RDY bewirkt das Löschen des Mikrobefehlsadreßregisters und damit den Übergang in den Anfangszustand. Auf diesem Wege dauert der Anfangszustand nur einen Maschinenzyklus. Wird er aber mit INIT erzwungen, kann er längere Zeit anliegen. Es ist die Frage, ob das in der Anwendungsumgebung stört oder nicht (wenn ja, müßte der erste Mikrobefehl ein NOP sein).

b) RDY bewirkt, daß eine feste Anfangsadresse ins Mikrobefehlsadreßregister geladen wird. Damit ist die Endlosschleife der zyklischen Bitmusterausgabe vom Anfangszustand getrennt. Alle Zustände der Endlosschleife dauern dann jeweils nur einen einzigen Maschinenzyklus. Nach dem einschaten oder Rücksetzen /INITMan könnte ein Initialisierungsmikroprogramm vorsehen, das mit dem Anfangszustand (also an Adresse 0) beginnt und nach dem Einrichten der Maschine in die Endlosschleife einmündet.

Das zyklische Ausgeben anhalten

Abb. 6.4 zeigt eine Ergänzung, die es übergeordneten Einrichtungen ermöglicht, die zyklische Ausgabe nach dem Durchlauf anzuhalten und wieder zu starten (Start-Stop-Betrieb).

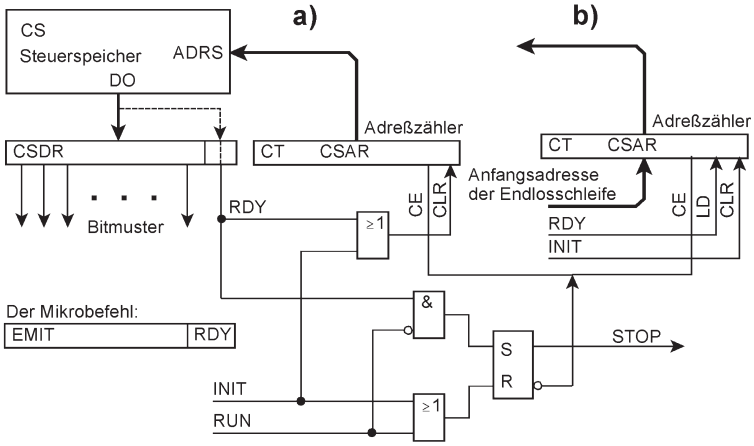


Abb. 6.4 Zyklisches Ausgeben mit Start-Stop-Betrieb. Die Varianten a) und b) entsprechen Abb. 6.3.

Dieses Mikroprogrammsteuerwerk hat einen zusätzlichen Steuereingang RUN und ein Stop-Flipflop STOP. Es verhindert, daß die Mikrobefehlsadresse weiterzählt. Ist RUN aktiv, kann STOP nicht einschalten. RDY bewirkt dann einen zyklischen Umlauf gemäß Abb. 6.3. Ist RUN inaktiv, bewirkt RDY, daß STOP einschaltet. Der jeweilige erste Zustand des zyklischen Umlaufs wird dann noch eingeleitet (je nach Variante a) oder b)), die Mikrobefehlsadresse zählt dann aber nicht weiter. Somit liegt der jeweils erste Mikrobefehl solange an, wie STOP aktiv ist. STOP kann in den Schaltungen der Anwendungsumgebung verwendet werden, um dort Mikrobefehlswirkungen zu verhindern, während sich die Maschine im Stopzustand befindet[3]. Wird RUN wieder aktiv, startet der Ablauf von neuem. Steuerung der Betriebsarten (von außen):

1. Zyklisch (umlaufend). RUN aktiv halten.
2. Start-Stop (Anhalten nach jedem Durchlauf): RUN ausschalten. Wenn STOP aktiv ist, einen RUN-Impuls geben, der kürzer ist als ein Durchlauf dauert. Ein einziger Taktzyklus genügt.

Bitmuster auf Kommando ausgeben

Oftmals müssen Sequencer nicht nur einen einzigen Ablauf steuern, sondern jeweils einen von mehreren Teilabläufen, deren Ausführung von außen angewiesen wird. Abb. 6.5 veranschaulicht eine einfache Lösung. Was wir hier als Kommando bezeichnen, ist ein einmaliger Durchlauf, wie im Zustandsdiagramm von Abb. 6.2 gezeigt. Jedes dieser Kommando hat eine eigene

3. Womöglich genügt auch die Einfachlösung, als ersten Mikrobefehl einen NOP zu programmieren.

Startadresse, die ins Mikrobefehlsadreßregister geladen wird (CMD_ADRS). Die übergeordnete steuernde Einrichtung legt diese Adresse an und aktiviert das Steuersignal CMD_LOAD, um die Startadresse zu laden. Wenn CMD_LOAD länger als einen Zyklus aktiv ist, wird der erste Mikrobefehl des Kommandos immer wieder gelesen[4].
Das Auslösen der Kommandos, deren wiederholte Ausführung usw. obliegt der steuernden Einrichtung. Die Schaltung von Abb. 6.5 erlaubt es, ein neues Kommando zu laden, während ein anderes Kommando ausgeführt wird. Die Ausführung des laufenden Kommandos wird damit abgebrochen. Das ist nicht selten eine nützliche Funktion, u. a. zur Fehlerbehebung und zum Eingreifen in Notfällen. Wenn aber ein neues Kommando erst geladen werden darf, nachdem das bisherige vollständig ausgeführt wurde, muß die steuernde Einrichtung die Fertigmeldung (RDY) abfragen oder die Ausführungszeit auf andere Weise berücksichtigen[5].

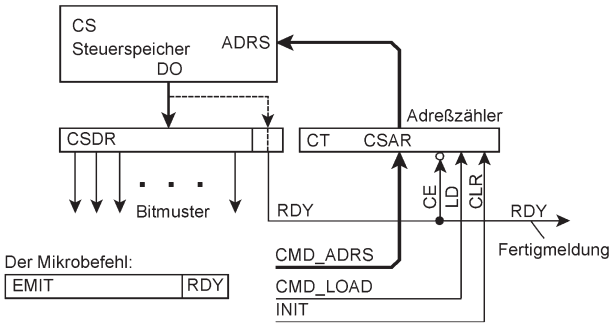


Abb. 6.5 Dieses Mikroprogrammsteuerwerk kann auf Kommando unterschiedliche Bitmuster ausgeben. Jedes Bitmuster hat eine Anfangsadresse, die als Kommandoadresse geladen wird.

Im Mikroprogramm warten

Die Schaltungen der Abb. 6.2 bis 6.5 liefern in jedem Maschinenzyklus eine neues Bitmuster. Die Taktzyklen bestimmen das Realzeitraster der Maschine. Um längere Zeiten darzustellen, muß man entsprechend viele Mikrobefehle nacheinander ausführen. Manche werden nur aufgerufen, um Zeit zu verbrauchen. Sonst haben sie keine Wirkung (No Operation, NOP). Geht es um längere Zeiten (Millisekunden, Sekunden oder noch mehr[6]) ist dieses Prinzip praktisch unbrauchbar. Um einen einfachen Sequencer in einem Wartezustand zu halten, kann man das Zählen der Mikrobefehlsadresse verhindern. Abb. 6.6 zeigt eine Prinzipschaltung. Das Warten

4. Wenn das unerwünscht ist: als ersten Mikrobefehl stets einen NOP programmieren oder RUN so bilden, daß es nur in einem einzigen Zyklus aktiv ist (Single-Shot).
5. Man könnte z. B. an Pufferregister denken, die dem Mikrobefehlsadreßregister CSAR vorgeschaltet werden.
6. Sie kommen in der Praxis recht häufig vor. Solche Anforderungen sind u. a. überall dort zu erfüllen, wo man früher Relaisketten, elektromechanische Schaltwerke, Magnetbänder usw. eingesetzt hat. Die Flüssigkeitsrakete – ein etwas exotisches, aber einleuchtendes Beispiel – wurde bereits erwähnt (S. 36). Um etwas Naheliegenderes, Einfacheres zu haben, denke man beispielsweise an das sog. Power Sequencing, das schrittweise Ein- und Ausschalten elektrischer Verbraucher mit hoher Stromaufnahme.

hängt von der Wartebedingung ab. Der Wartemikrobefehl (WCTL) sieht nur ein einziges Bedingungssignal: warten oder nicht warten, also den Ablauf fortsetzen (PROCEED). Es wird von einer Wartesteuerschaltung (WAIT CTL) geliefert. Welche Wartebedingung auf das Bedingungssignal wirkt, muß eingestellt werden, bevor der Wartemikrobefehl ausgeführt wird. Wartebedingungen können Zeitgebersignale sein, einzelne Signale aus der Anwendungsumgebung oder anwendungsspezifische Signalverknüpfungen[7]. Der Wartemikrobefehl kann warten, bis eine bestimmte Zeit abgelaufen ist oder bis bestimmte Signale vorgegebene Werte angenommen haben.

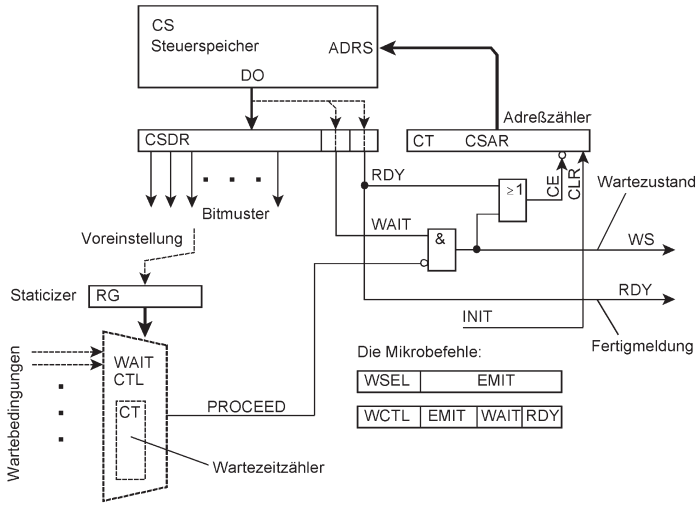


Abb. 6.6 Ein einfacher Sequencer, der auch warten kann – auf Bedingungen oder so lange, wie als Wartezeit vorgegeben wird. Getrennte Mikrobefehle (vertikale Mikroprogrammierung): WSEL zum Einstellen der Wartebedingung, WCTL zum Warten.

Zum Warten sind zwei Mikrobefehlstypen vorgesehen:

- WSEL (Wait Condition Select) zum Einstellen der Wartesteuerschaltung und zum Auswählen der Wartebedingung. Das Emit-Feld enthält die zugehörigen Mikroanweisungen und Direktwerte.
- WCTL (Wait and Control) als Wartemikrobefehl. Im Beispiel ist ein Emit-Feld dargestellt, um zu veranschaulichen, daß der Wartemikrobefehl auch Steuerwirkungen ausüben kann. Im einfachsten Fall gibt es gar keine Steuerwirkungen. Ein solcher Wartemikrobefehl kann keine Signale schalten oder Direktwerte laden, sondern nur warten. Was vor oder nach dem Warten zu erledigen ist, wäre mit weiteren Mikrobefehlen auszuprogrammieren.

7. Die beispielweise von einer anwendungsspezifischen Adapterschaltung geliefert werden. Vgl. S. 190f.

Was wollen wir unter einer Wartebedingung verstehen?
Es geht um eine sprachliche Spitzfindigkeit. Soll so lange gewartet werden, wie die Bedingung X anliegt ist oder soll der Wartezustand verlassen werden, wenn die Bedingung X wirksam geworden ist? Die eine Bedingung ist offensichtlich die Negation der anderen. Beides ist also logisch gleichwertig. In der Praxis spricht man aber wohl zumeist vom Warten auf etwas (warten, bis ein Antwortsignal kommt, bis die Zeit abgelaufen ist usw.). Wir wollen im folgenden bei dieser Gepflogenheit bleiben[8]. Dementsprechend ist die Wartebedingung eine Bedingung, die das Verlassen des Wartezustands bewirkt, also eigentlich die Bedingung zum Fortsetzen des Mikroprogramms (Proceed). Warten heißt, das Weiterzählen der Mikrobefehlsadresse verhindern und den Wartemikrobefehl immer wieder aufrufen. Ist die WAIT-Anweisung gesetzt und die Wartebedingung nicht erfüllt (PROCEED = 0), wird das Zählen der Mikrobefehlsadresse verhindert und der Wartemikrobefehl in jedem Zyklus neu geladen. Ist die Wartebedingung erfüllt, wird die Mikrobefehlsadresse weitergezählt. Der Wartemikrobefehl liegt in diesem Maschinenzyklus noch an; dann wird der nachfolgende Mikrobefehl ausgeführt. Ist die Wartebedingung schon am Anfang erfüllt, dauert der Wartemikrobefehl nur einen einzigen Zyklus. Übergeordnete Einrichtungen und Schaltungen der Anwendungsumgebung können den Wartezustand (Wait State WS = WAIT • $\overline{\text{PROCEED}}$) abfragen.

Steuerwirkungen im Wartemikrobefehl
Die Frage ist, wann: sofort nach dem Aufruf, während des Wartens immer wieder, oder nach dem Verlassen des Wartezustands? Diese Mikrobefehlswirkungen sind anwendungsspezifisch festzulegen (Sache des Feinentwurfs). Ein typisches Beispiel ist die Steuerung von Handshaking-Abläufen an einem Interface. Ein WCTL-Mikrobefehl kann z. B. auf ein Antwortsignal vom Interface warten und, wenn die Antwort eingetroffen ist, sofort reagieren, also Daten vom Bus in ein Register laden und die zugehörigen Bestätigungssignale des Interfaces erregen[9]. Obwohl ein solcher Mikrobefehl mehrere Taktphasen erfordert, laufen die Signalspiele am Interface schneller ab als wenn man sie mit einfacheren Mikrobefehlen ausprogrammiert.

In einem einzigen Mikrobefehl die Wartebedingung auswählen und warten
Der Wartemikrobefehl enthält auch die Auswahl der Wartebedingung. Ein solcher Mikrobefehl steuert seine Nachfolge aufgrund von Bedingungen, die von Mikroanweisungen ausgewählt werden, die er selbst mitbringt. Mit einer einzigen Taktflanke ist das nicht zu bewältigen. Die Abb. 6.7 und 6.8 veranschaulichen eine Lösung. Hier bleiben wir zwar bei einem einzigen Takt (CLK), verwenden aber beide Taktflanken. Die Mikrobefehlszugriffe (Adreßzählung, Adressieren des Steuerspeichers, Laden des Mikrobefehlsregisters) unterscheiden sich nicht von den Abläufen, die vorstehend beschrieben wurden. Die zweite Taktflanke betrifft nur die Wartesteuerung. Sie schaltet lediglich das WAIT-Flipflop.

8. Abweichend vom Sprachgebrauch in den vorhergehenden Kapiteln. Dort war es ein Warten solange X, hier ist es ein Warten auf X = solange $\overline{X}$.
9. Beispiele s. [MD27].

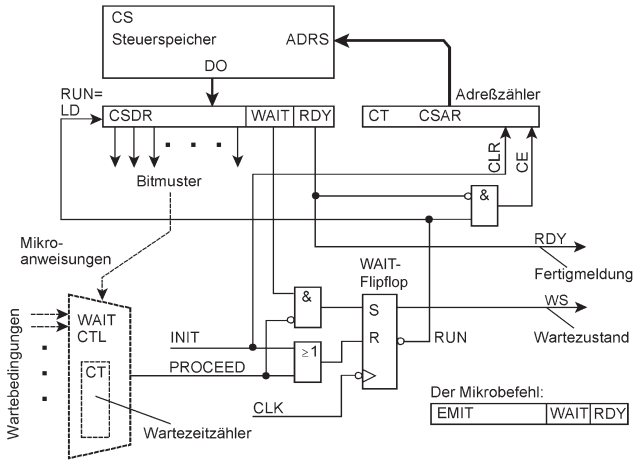


Abb. 6.7 Ein einziger Mikrobefehl zum Einstellen der Wartebedingung und zum Warten (horizontale Mikroprogrammierung). Das WAIT-Flipflop schaltet mit der zweiten Taktflanke.

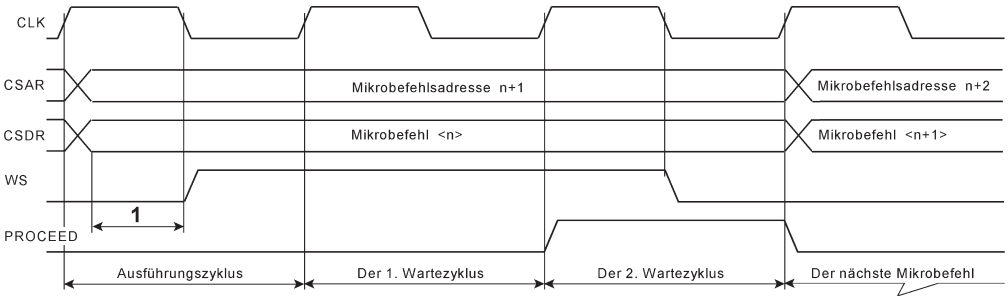


Abb. 6.8 Warten in der Schaltung von Abb. 6.7. Das WAIT-Flipflop wird mit der zweiten Taktflanke geschaltet. 1 - diese Zeit steht zur Verfügung, um zu entscheiden, ob gewartet wird oder nicht.

Wenn der Wartemikrobefehl ins Mikrobefehlsregister übernommen wird, zählt auch das Mikrobefehlsadreßregister um 1 weiter. Damit wird jetzt schon der nachfolgende Mikrobefehl im Steuerspeicher adressiert. Der Wartemikrobefehl soll aber so lange stehenbleiben, bis die Wartebedingung (PROCEED) erfüllt ist. Mit der zweiten Taktflanke, also etwa in der Mitte des Zyklus, wird die Wartebedingung ausgewertet. Ist sie nicht erfüllt (PROCEED = 0), schaltet das WAIT-Flipflop ein (Wartezustand). Das verhindert sowohl das Zählen der Mikrobefehlsadresse als auch das Laden des Mikrobefehlsadreßregisters. Der Wartemikrobefehl verbleibt im Mikrobefehlsregister, und das Mikrobefehlsadreßregister adressiert den nächsten Mikrobefehl, der am Ausgang des Steuerspeichers bereitsteht. Bei erfüllter Wartebedingung (PROCEED = 1) schaltet das WAIT-Flipflop aus. Daraufhin wird zu Beginn des nächsten Zyklus der nachfolgende Mikrobefehl ins Mikrobefehlsregister übernommen und die Mikrobefehlsadresse weitergezählt.

Zeit abwarten

Diese Aufgabe ist ziemlich oft zu lösen. Abb. 6.9 zeigt einen Sequencer, der Zeitintervalle abwarten kann. Hier werden sie im Wartemikrobefehl als Direktwerte vorgegeben. Es ist im Prinzip die Schaltung von Abb. 6.7, nur beschränkt auf eine einzige Wartebedingung. Sie kommt hier vom Wartezeitzähler.

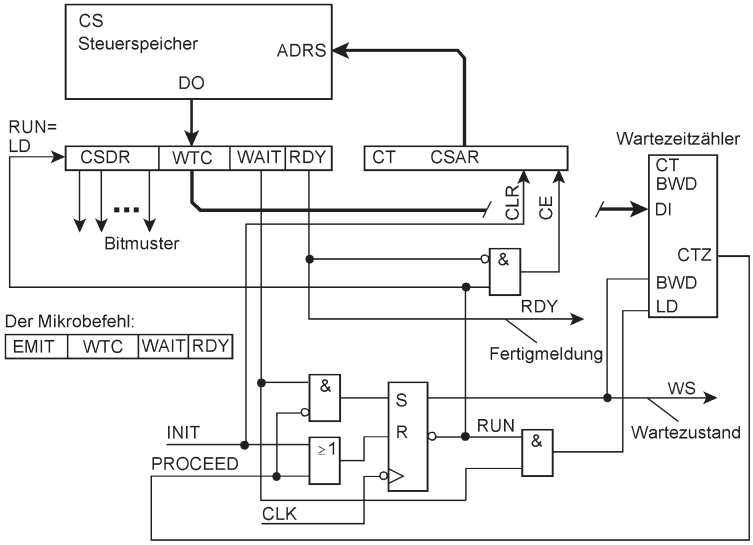


Abb. 6.9 Ein Sequencer, der Zeitintervalle abwarten kann.

Der Wartezeitzähler ist ein Rückwärtszähler. Der Mikrobefehl enthält ein Direktwertfeld mit der Wartezeit (WTC = Wait Count). Der Wartezeitzähler schaltet mit der zweiten Taktflanke. Er wird geladen, wenn die Ausführung des Wartemikrobefehls beginnt (WAIT-Anweisung im Mikrobefehl und WAIT-Flipflop noch ausgeschaltet). Anschließend wird sein Inhalt in jedem weiteren Zyklus des Wartemikrobefehls um 1 vermindert. Ist der Zählerstand = 0 (CTZ = Count Zero), wird die Wartebedingung aktiv (PROCEED = 1) und das Warten beendet (Abb. 6.10).

Erweiterte Zeitzählung

Gelegentlich steht die Aufgabe, ein Realzeitraster exakt einzuhalten. Es sind Zeitintervalle vorgegeben. In jedem Zeitintervall ist aber unterschiedlich viel zu tun (Abb. 6.11). Nachdem die jeweilige Mikrobefehlsfolge ausgeführt wurde, muß so lange gewartet werden, bis das Zeitintervall zu Ende ist. Nur in ganz einfachen Fällen kann man auszählen, wieviele Takte nach dem letzen Mikrobefehl noch abzuwarten sind, und demgemäß NOPs einfügen. Das gelingt aber schon dann nicht mehr, wenn die Mikroprogrammstücke Wartemikrobefehle enthalten, die auf Signale der Anwendungsumgebung warten. In komplizierteren Steuerwerken mit Verzweigungen, Break-Ins usw. wird es überhaupt nicht funktionieren. Wir brauchen also einen Intervallzeitzähler, der unabhängig von den Mikrobefehlen die Zeit sozusagen nebenher auszählt.

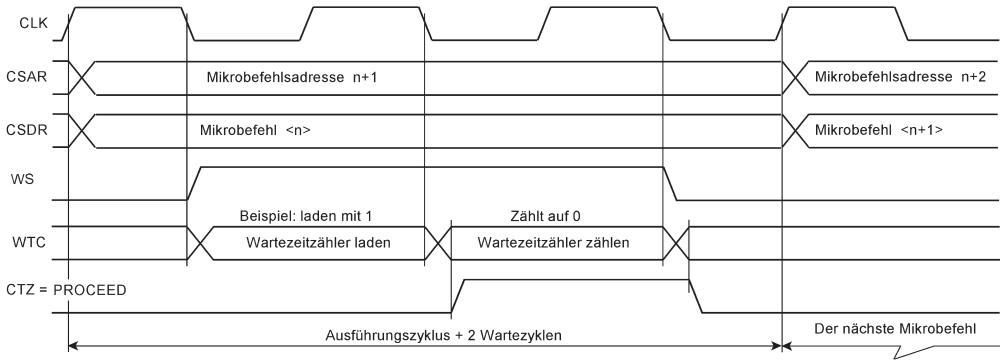


Abb. 6.10 Warten und Wartezeitzählung in der Schaltung von Abb. 6.9.

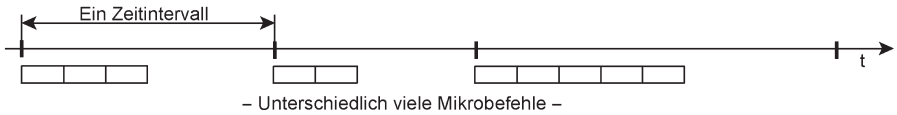


Abb. 6.11 Ein Realzeitraster aus vorgegebenen Zeitintervallen ist exakt einzuhalten. Die Mikroprogramme laufen aber unterschiedlich lange.

Abb. 6.12 zeigt eine Abwandlung der Schaltung von Abb. 6.9. Es gibt einen Einstellmikrobefehl LDW (Load Wait Count), der den Rückwärtszähler lädt und die Zählerlaubnis einschaltet. Der Wartemikrobefehl kann nur warten. Der erste Mikrobefehl im Zeitintervall ist dann ein LDW, der letzte Mikrobefehl ein WAIT (Abb. 6.13). Der Wartezeitzähler zählt die gesamte Länge des Zeitintervalls ab, der WAIT-Mikrobefehl wartet so lange, bis das Zeitintervall zu Ende ist. Das Wartebedingungssignal (PROCEED) schaltet auch die Zählerlaubnis aus.

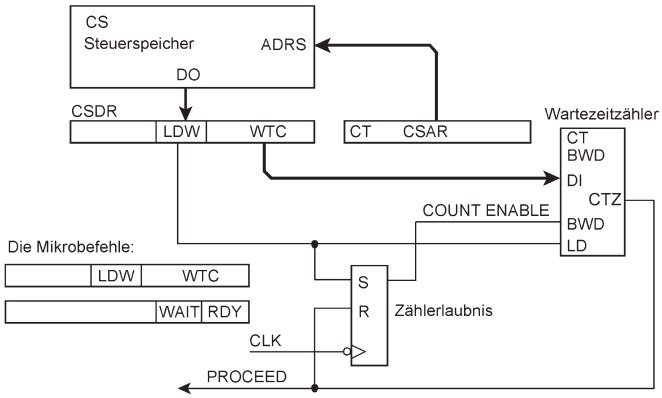


Abb. 6.12 Der Sequencer von Abb. 6.9 wird so abgewandelt, daß er warten kann, bis ein Zeitintervall abgelaufen ist. Die Änderung betrifft nur den Wartezeitzähler. Schaltung sonst wie Abb. 6.9.

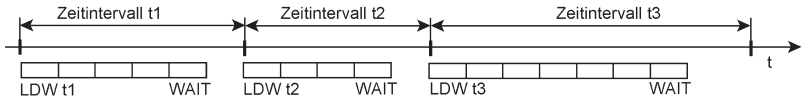


Abb. 6.13 So werden Zeitintervalle ausprogrammiert.

Abb. 6.14 zeigt eine weitere Abwandlung. Der Wartezeitzähler ist hier ein Vorwärtszähler, der anfänglich gelöscht wird (Mikroanweisung WINIT). Die Wartezeit ist ein Direktwert (WTC = Wait Count), der mit dem Inhalt des Wartezeitzählers verglichen wird. Es wird so lange gewartet, bis Zählwert und Wartezeitwert gleich sind. Der Vorteil: wenn eine Folge von Zeitintervallen auszuzählen ist, muß man den Zähler nur einmal (ganz am Anfang) löschen (Abb. 6.15).

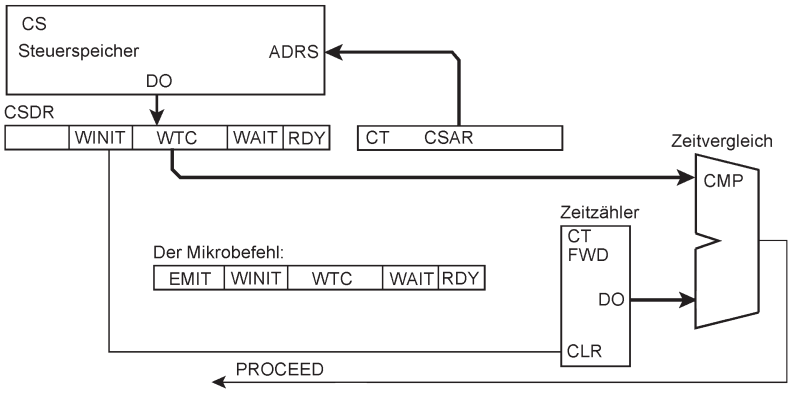


Abb. 6.14 Zeitzählung mit Wartezeitvergleich. In diesem Beispiel werden alle Mikroanweisungen in einem einzigen Mikrobefehlsformat untergebracht (horizontale Mikroprogrammierung). Die Änderung betrifft nur den Wartezeitzähler mit Vergleicher. Der Zähler zählt ständig und wird vom Mikroprogramm nur gelöscht[10]. Schaltung sonst wie Abb. 6.9.

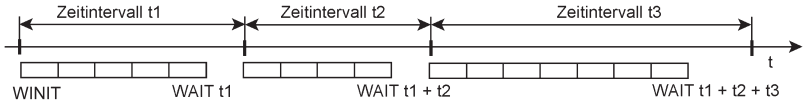


Abb. 6.15 Zeitintervalle mittels Zeitvergleich ausprogrammieren.

In den obigen Beispielen sind die Zähl- und Zeitparameter Direktwerte im Mikrobefehl. Es versteht sich von selbst, daß man solche Werte auch in programmseitig ladbaren Registern übergeben kann. Praxistip: Die Zeitzählschaltungen großzügig dimensionieren (Zählweite nicht zu knapp). Zu den Zeitintervallen s. weiterhin Abschnitt 8.4 (S. 470).

10. Mikroanweisungen zum Ein- und Ausschalten des Zählers (z. B. zum Stromsparen) sind Sache der Entwurfsoptimierung. Schaltungslösung mit Zählerlaubnisflipflop ähnlich Abb. 6.12.

6.2 Universelle Steuerautomaten (Branch Sequencers)

Wir beschränken uns zunächst auf Steuerwerke, die binäre Eingangssignale abfragen und binäre Ausgangssignale erregen[11]. Das Speichern von Daten, das Ausführen von Rechenoperationen usw. wird nicht berücksichtigt. Auch wollen wir mit wenig Aufwand auskommen. Das typische Entwurfsziel ist das kleine Mikroprogrammsteuerwerk als Alternative oder Ergänzung zum Mikrocontroller. Es soll nur wenige Ressourcen verbrauchen und sich einfach programmieren lassen[12].

Zustandsübergänge

Das Mikroprogrammsteuerwerk soll elementare Zustandsübergänge des Weiterschaltens, Wartens und Verzweigens unterstützen (Abb. 6.16). Das ist die Aufgabe der Mikrobefehlsadressierung. Die Mikrobefehlsformate müssen so gestaltet werden, daß die zugehörigen Anweisungen und Parameter hineinpassen.

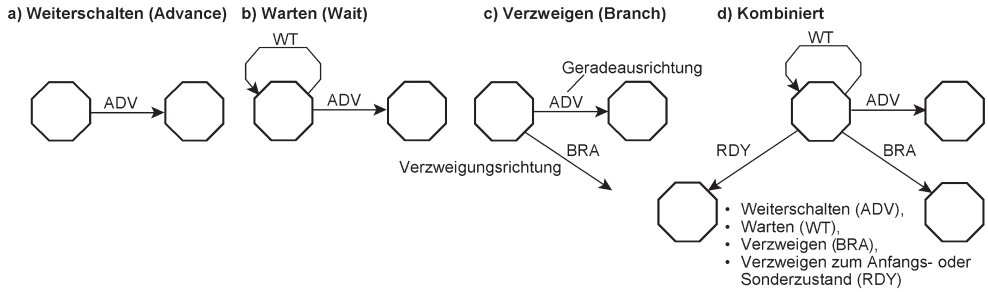


Abb. 6.16 Zustandsübergänge. Diese Übergangsmuster dienen als Vorlage zum Implementieren der Mikrobefehlsadressierung.

a) Weiterschalten (ADV = Advance). Das Weiterschalten entspricht dem unbedingten Übergang von einem Mikrobefehl zu seinem einzigen Nachfolger. Da wir einfache, leicht zu programmierende Maschinen anstreben, implementieren wir es durch Adreßzählung. Aufeinander folgende Mikrobefehle werden nacheinander angeordnet, wie beim üblichen Universalprozessor. Die Adreßzählung wurde bereits in Abschnitt 6.1 beschrieben. Hier wird sie mit anderen Arten der Zustandsübergänge kombiniert.

b) Warten (WT = Wait). Warten heißt, daß der Mikrobefehl erhalten bleibt, also nicht überladen wird. Es wird so lange gewartet, bis die zugehörige Wartebedingung erfüllt ist. Das Warten wurde bereits in Abschnitt 6.1 beschrieben. Hier wird es mit anderen Arten der Zustandsübergänge kombiniert.

11. Zur grundsätzlichen Entwurfsaufgabe vgl. auch Abb. 5.1.
12. Vergleichbar der Assemblerprogrammierung typischer Mikrocontroller. Mikroprogrammsteuerwerke mit Mehrfach- und Spätverzweigung, Break-Ins usw. scheiden deshalb hier aus. Zur Nutzung typischer Makroassembler vgl. Anhang 1 (S.537).

c) Bedingt verzweigen (BRA = Branch). Bedingt verzweigen heißt, eine Mikrobefehlsadresse (Verzweigungsadresse) ins Mikrobefehlsadreßregister zu laden, wenn die zugehörige Bedingung erfüllt ist (Mikroprogrammfortsetzung in Verzweigungsrichtung). Ist die Bedingung nicht erfüllt, wird der nächste Mikrobefehl auf andere Weise adressiert, in den meisten Fällen durch Weiterschalten (Adreßzählung, Mikroprogrammfortsetzung in Geradeausrichtung). Die Verzweigungsadresse kann als Direktwert aus dem Mikrobefehl oder aus Registern entnommen werden, sie kann eine Festadresse sein oder aus Signalen der Anwendungsumgebung gebildet werden (Funktionsverzweigung). Nähere Einzelheiten und alternative Lösungen wurden bereits in Kapitel 5 erläutert. Wir beschränken uns hier auf die einfachste Implementierung, bei der die Verzweigungsadresse ein Direktwert im Mikrobefehl ist.

d) Zu einer Anfangs- oder Festadresse verzweigen (RDY = Ready). Dieses universelle Übergangsmuster wurde bereits in Kapitel 2 erläutert[13]. Die Mikrobefehle können warten und bedingt zu einer beliebigen Adresse oder zu einer Festadresse verzweigen. Das Verzweigen zur Festadresse ist typischerweise vorgesehen, um Fehlerbedingungen zu behandeln, um anzuhalten oder um am Ende des Mikroprogrammablaufs wieder in die Hauptsteuerschleife einzutreten. Die einfachste Lösung besteht im Löschen der Mikrobefehlsadresse; es ist praktisch ein programmseitig erzwungenes Rücksetzen. Das Rücksetzmikroprogramm muß die Ursache des Rücksetzens abfragen und zur jeweiligen Behandlungsroutine verzweigen. Alternativ dazu kann man beliebige feste Adressen ins Mikrobefehlsadreßregister laden. Hierzu müssen an dessen Dateneingängen Auswahlschaltungen (z. B. Multiplexer) vorgesehen werden.

Der universelle Steuerautomat als Mikroprogrammsteuerwerk

Es ist eine Einfachlösung in folgerichtiger Weiterentwicklung der Entwurfsgedanken in den Abb. 2.20 und 2.44. Die Maschine in Abb. 2.20 wurde um einen Steuerzähler herumgebaut. Der wird hier zum Mikrobefehlsadreßregister CSAR. Es ist ein Binärzähler, dessen Funktionen – Zählen, Laden und Löschen – voll ausgenutzt werden; das Zählen zum Holen des nachfolgenden Mikrobefehls, das Anhalten des Zählens zum Warten, das Laden zum Verzweigen und das Löschen zum Rücksetzen bzw. zum Übergang in einen Anfangszustand. Die Zählwerte des Steuerzählers sind binär codierte Zustände. Hier werden sie zu Mikrobefehlsadressen. Im Grunde nutzen wir den Adreßdecoder des Steuerspeichers aus, um die Zustände zu decodieren. Diese 1:1-Entsprechung – ein Zählwert = ein Maschinenzustand = eine Mikrobefehlsadresse – führt auf Maschinen mit horizontalen Mikobefehlen (Abb. 6.17 und 6.18). Eine so einfache Maschine kann vollsynchron entworfen werden (mit einer einzigen Taktflanke für alles (Einphasentakt)), vorausgesetzt, die Bedingungssignale werden entsprechend synchronisiert und im richtigen Zeitraster bereitgestellt (Sache der Anwendungsschaltungen).

13. Vgl. die Abschnitte 2.1.2 und 2.1.3, insbesondere Abb. 2.4 und 2.5 (S. 29f).

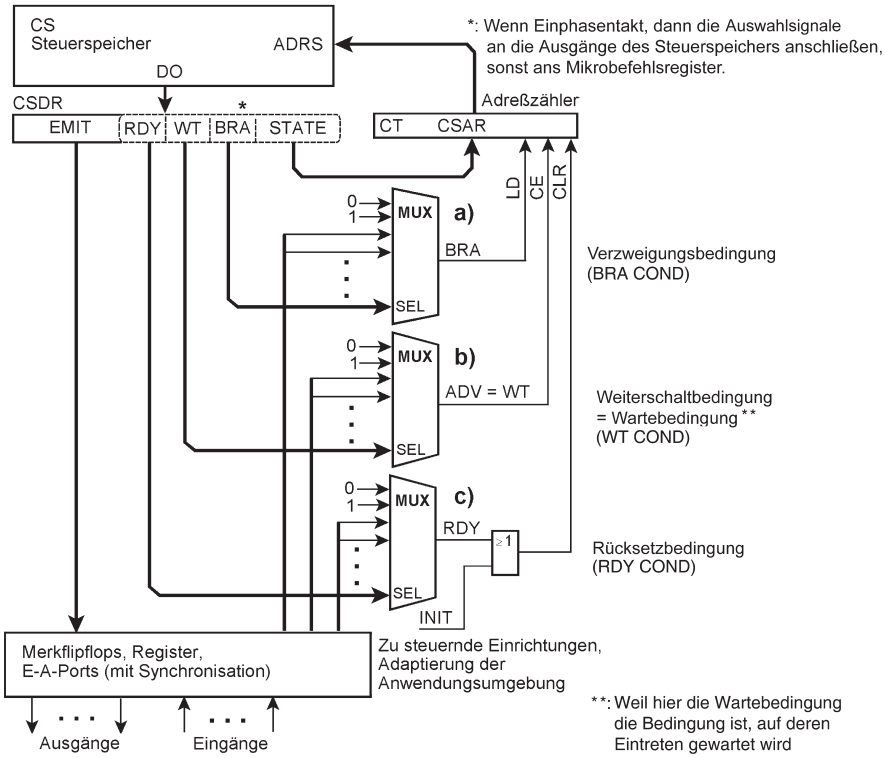


Abb. 6.17 Ein universeller Branch Sequencer als Mikroprogrammsteuerwerk. Jede Mikrobefehlsadresse entspricht einem Maschinenzustand (State). Die Festwerte (0, 1) an den Auswahlschaltungen dienen zum Unterdrücken oder zum unbedingten Auslösen des jeweiligen Zustandsübergangs.

Abb. 6.18 zeigt das Muster der Zustandsübergänge und das horizontale Mikrobefehlsformat:

a) Verzweigen (Branch; BRA). Die Verzweigungsadresse (STATE = neuer Zustand) wird geladen, wenn die Verzweigungsbedingung (Branch Condition; BRA COND) erfüllt ist. Soll unbedingt verzweigt werden, den Festwert 1 auswählen, soll nicht verzweigt werden, den Festwert 0.

b) Warten (Wait; WT). Ist die Wartebedingung (Wait Condition; WT COND) nicht erfüllt, wird die Adreßzählung angehalten; der aktuelle Mikrobefehl verbleibt im Mikrobefehlsregister. Ist die Wartebedingung erfüllt, wird der nächste Mikrobefehl ausgeführt (ADV = Advance = Weiterschalten). Soll nicht gewartet werden, den Festwert 1 auswählen, soll immer gewartet werden, den Festwert 0. Immer warten = Anhalten. Dabei sind aber die anderen Zustandsübergänge (BRA, RDY) noch möglich. Anhalten mit Rücksetzen: nicht verzweigen und Rücksetzbedingung auswählen, hartes Anhalten (Endzustand): weder verzweigen noch zurücksetzen (jeweils Festwert 0 auswählen).

c) Rücksetzen (Ready; RDY). Ist die Rücksetzbedingung (Ready Condition; RDY COND) erfüllt, wird das Mikrobefehlsadreßregister gelöscht und der nächste Mikrobefehl von Adresse 0 gelesen. Soll unbedingt zurücksetzt werden, den Festwert 1 auswählen, soll nicht zurückgesetzt werden, den Festwert 0.

Prioritäten:
1. Rücksetzen (CLR = RDY), 2. Verzweigen = Laden (LD = BRA) 3. Zählen (ADV = WT = CE), 4. nichts tun (weder CLR noch BRA noch WT = $\overline{\text{CLR V BRA V WT}}$). Die meisten der typischen Binärzähler (als Funktionselemente in den Bibliotheken der Entwicklungssysteme oder als Schaltkreise) sind so ausgelegt.

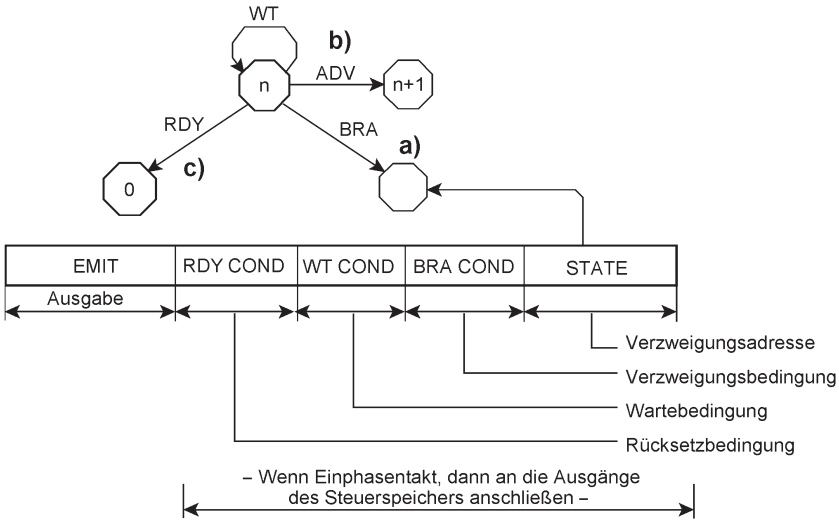


Abb. 6.18 Die Zustandsübergänge und das horizontale Mikrobefehlsformat.

Die Abb. 6.19 und 6.20 veranschaulichen einen solchen Sequencer mit einigen praxisgerechten Erweiterungen. Hier nehmen wir an, daß es zweckmäßiger ist, mit zwei Taktflanken oder mit Taktphasen zu arbeiten. Deshalb sind alle Steuer-, Direktwert- und Auswahlsignale ans Mikrobefehlsregister angeschlossen.Die Schaltung ist um einen Zeitgeber mit Zeitvergleich erweitert. Die Zeitvergleichsbedingung kann als Verzweigungsbedingung, Wartebedingung oder Rücksetzbedingung ausgewählt werden.

Bedingungen invertieren
Die Auswahlfelder enthalten ein zusätzliches Bit, das steuert, ob die Bedingung direkt oder invertiert wirksam wird (CPL = Complement). Deshalb genügt auch der Wert 0 als Festwert.

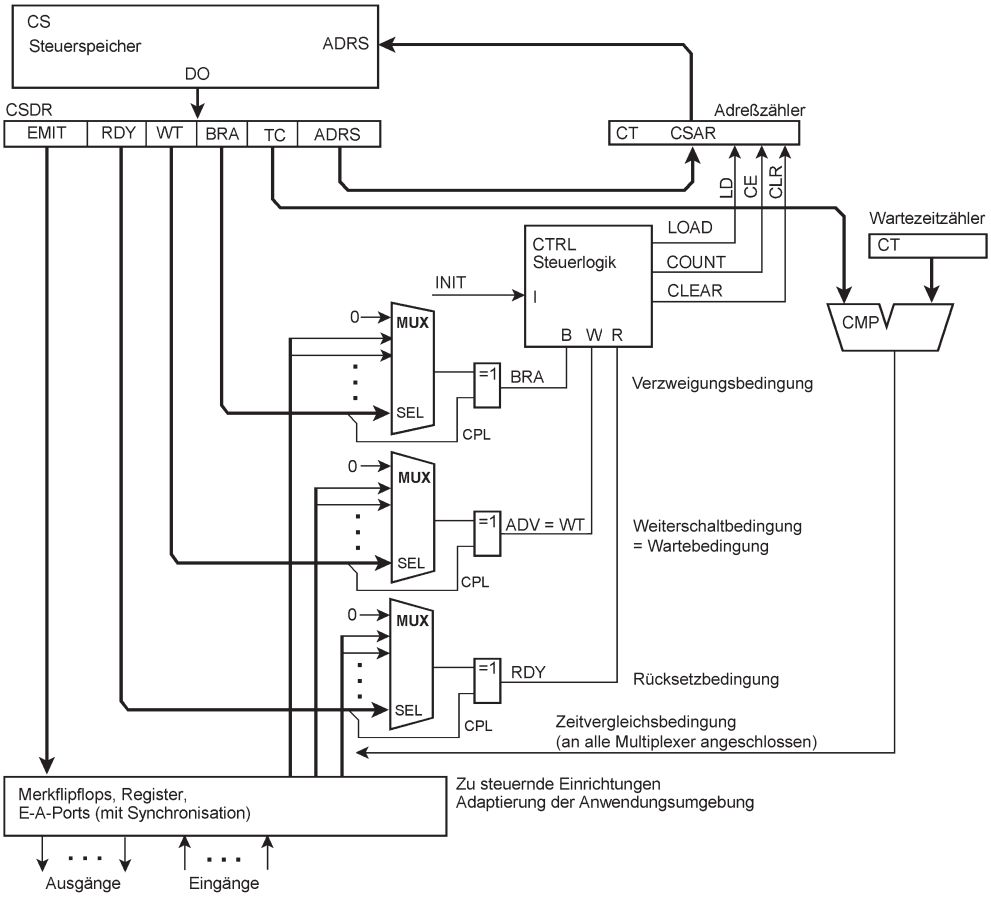


Abb. 6.19 Ein erweiterter universeller Branch Sequencer. Die Steuerlogik ist als Funktionsblock dargestellt, der mittels Verhaltensbeschreibung erfaßt werden kann. Die XOR-Gatter dienen zum Invertieren der ausgewählten Bedingungen.

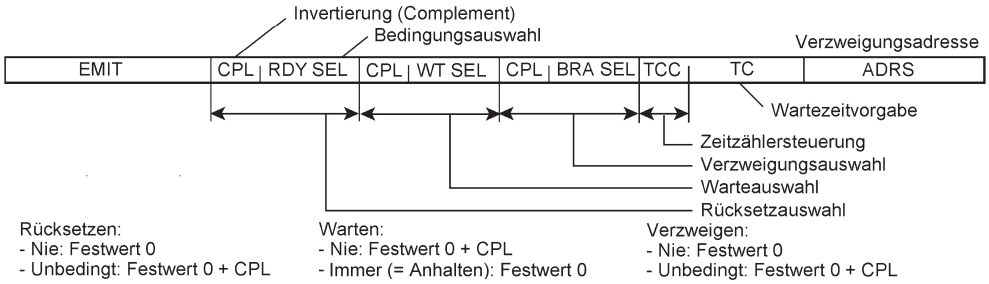


Abb. 6.20 Das horizontale Mikrobefehlsformat.

Bedingungen kombinieren
Die im Mikrobefehl von Abb. 6.20 ausgewählten Bedingungen wirken unabhängig voneinander gemäß der Priorität der zugehörigen Steuersignale am Mikrobefehlsadreßregister CSAR. Beispiele:

- Warten auf Bedingung A, Verzweigen, wenn Bedingung B, Rücksetzen, wenn Bedingung C. Die Maschine wartet, wenn weder A noch B noch C aktiv sind (Guaranteed Coverage). Wird A aktiv, so wird der Wartezustand verlassen und der nachfolgende Mikrobefehl aufgerufen.Wird B während des Wartens aktiv, so wird die Verzweigung ausgeführt, wird C aktiv, wird das Mikroprogramm zurückgesetzt und neu gestartet.
- Warten auf zwei Antwortsignale A, B (z. B. an einem Interface), Rücksetzen, wenn es zu lange dauert (Timeout): Zeitzählung gemäß Timeout-Intervall einstellen. Warten auf Signal A, Verzweigen, wenn Signal B, Rücksetzen, wenn Zeit abgelaufen.

Um jeden Preis sparen?
Nein. Die Minimal- und Sparlösungen der Vergangenheit sind nicht mehr von Bedeutung. Wir entwerfen einfach und dimensionieren großzügig. An die typischen elementaren Datenstrukturen (Bytes, 32-Bit-Wörter usw.) müssen wir uns nicht unbedingt halten[14]. Es ist ja gerade der grundsätzliche Vorteil des als Schaltung entworfenen Mikroprogrammsteuerwerks, daß man Adressen, Datenstrukturen, Anweisungsfelder usw. jeweils so lang auslegen kann, wie es zweckmäßig ist. Damit umgehen wir die Einschränkungen der kleinen Mikrocontroller und vermeiden den unnötigen Ressourcenverbrauch[15] der großen.

Einfache Mikrobefehlsformate
Ein horizontales Mikrobefehlsformat ist nur selten eine geeignete Grundlage für eine kleine programmierbare Maschine, die beispielsweise als IP Core irgendwo im FPGA sitzt, um einen Akzelerator oder eine Schnittstelle zu steuern. Wenn nur wenige Zustandsübergänge vorkommen, die so komplex sind wie in den Abb. 6.18 gezeigt, wird zuviel Speicherkapazität verschwendet.
Für kleine, kostengünstige Maschinen liegt es nahe, vertikale Mikrobefehlsformate zu bevorzugen. Wenn es besondere Mikrobefehle zum Ausgeben von Bitmustern, zum Warten, zum Rücksetzen und zum Verzweigen gibt, kann man jeden Zustandsübergang mit so vielen Mikrobefehlen ausprogrammieren, wie jeweils nötig sind. Das Warten kann durch ein Verzweigen auf sich selbst ersetzt werden, das Rücksetzen durch ein Verzweigen auf eine Anfangsadresse (z. B. Adresse 0).
Zwei Grundformate genügen (Abb. 6.21), eines zum Ausüben von Steuerwirkungen und zum Laden von Direktwerten (CONTROL) und eines zum Verzweigen (BRANCH). Ein Bit genügt, um die Formate voneinander zu unterscheiden (Formatcode).

14. Und wenn, dann runden wir großzügig auf. Typische Wortlängen, die man ausnutzen kann, ergeben sich u. a. aus den Zugriffsbreiten der Block-RAMs im FPGA (beispielsweise 18, 36, 48 und 72 Bits).
15. Der sich dann ergibt, wenn man deren Leistungsvermögen nur zu einem geringen Teil ausnutzt.

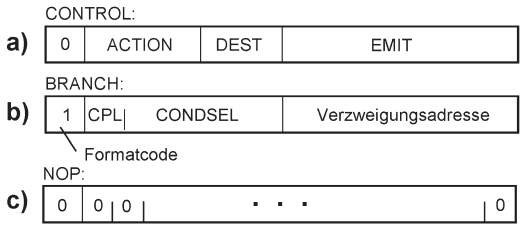


Abb. 6.21 Einfache Mikrobefehlsformate zum Ausführen von Anweisungen, zum Verzweigen und zum Nichtstun. Vgl. auch [MD27].

a) Anweisungen ausführen (Operationsmikrobefehl; CONTROL). Dieses Format kann beliebige Steuerbits und Anweisungsfelder aufnehmen. Die Adresse des nachfolgenden Mikrobefehls wird durch Adreßzählung gewonnen. Hier ist ein typisches Formatbeispiel dargestellt, das für viele Anwendungsfälle geeignet ist. Im ACTION-Feld ist die auszuführende Funktion codiert, es ist eine Art Operationscode (Opcode). Das DEST-Feld (Destination, Ziel) wählt das Signal, das Register usw. aus, dem das jeweilige Ergebnis zugewiesen werden soll. Das Emit-Feld enthält einen Direktwert oder – abhängig vom ACTION-Feld – weitere Mikroanweisungen oder Steuerbits. Alles, was über den aktuellen Mikrobefehlszyklus hinaus zu halten ist, muß in Register geladen werden.
b) Verzweigen (Verzweigungsmikrobefehl; BRANCH). Der Mikrobefehl enthält die Verzweigungsadresse. Das CONDSEL-Feld (Condition Select) wählt die jeweilige Verzweigungsbedingung aus. Das CPL-Bit (Complement) steuert, ob bei erfüllter oder bei nicht erfüllter Bedingung verzweigt wird.
c) Nichts tun (NOP). Manchmal braucht man Mikrobefehle, die nur Zeit verbrauchen, aber sonst keine Wirkung haben (No Operation, NOP). Hier ist es ein Operationsmikrobefehl, der ausschließlich aus Nullen besteht.

Weiterschalten und bedingt verzweigen – eine Minimalausstattung

Mehr braucht man eigentlich nicht. Mit diesen Abläufen lassen sich alle anderen Muster der Zustandsübergänge darstellen:

- Das Warten ist ein bedingtes Verzweigen auf sich selbst (Warteschleife; Abb. 6.22a).
- Das Verlassen der Warteschleife in eine alternative Richtung läßt sich mit einem zusätzlichen Verzweigungsmikrobefehl ausprogrammieren (Abb. 6.22b).
- Mit weiteren Verzweigungsmikrobefehlen kann man die Warteschleife in mehrere Richtungen verlassen (Abb. 6.22c).
- Das Anhalten ist ein unbedingtes Verzweigen auf sich selbst (Abb. 6.22d) oder eine Warteschleife, die so lange durchlaufen wird, bis eine Bedingung zum Fortsetzen oder erneuten Starten des Mikroprogramms erfüllt ist (Abb. 6.22e). Das unbedingte Verzweigen ist ein bedingtes Verzweigen mit einer festen Verzweigungsbedingung.

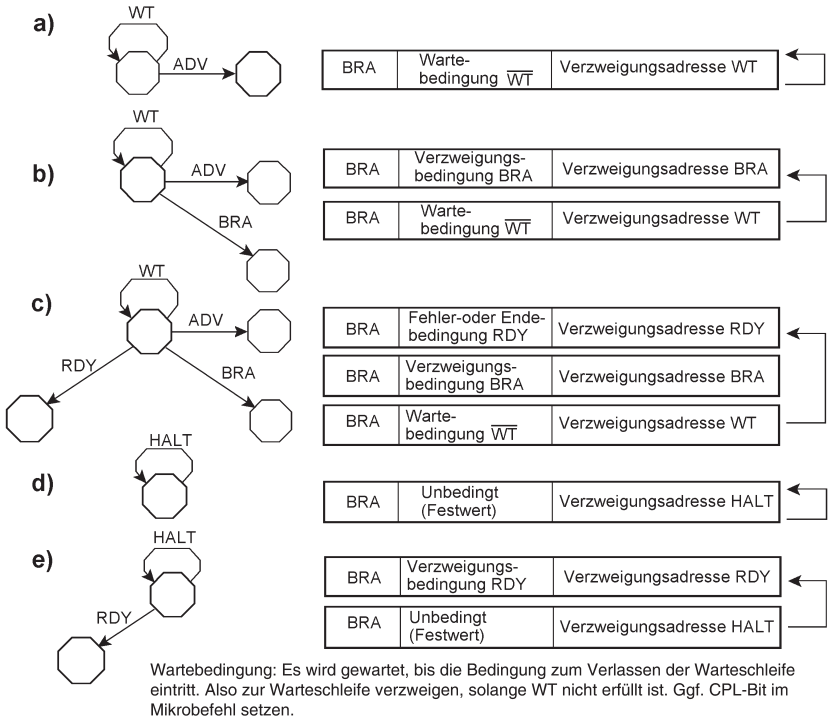


Abb. 6.22 Zustandsübergänge mit bedingten Verzweigungen (BRA = Branch) ausprogrammieren. a) Warten; b) Kombination Warten und Verzweigen; c) Kombination Warten, Weiterschalten und Verzweigen in zwei Richtungen; d) Anhalten; e) Anhalten mit Verzweigen zum erneuten Beginn (Neustart).

Einfache Mikroprogrammsteuerwerke

Abb. 6.23 veranschaulicht einen einfachen Branch Sequencer, der für die Mikrobefehlsformate von Abb. 6.21 ausgelegt werden kann. Auch lassen sich beide Mikrobefehlsformate zu einem entsprechend langen horizontalen Mikrobefehl zusammenfassen. Die Maschine kann nur zweierlei: aufeinanderfolgende Mikrobefehle ausführen und verzweigen. Warten und Anhalten sind auszuprogrammieren. Die Steuerung ist deshalb nicht besonders kompliziert. Die alternativen Implementierungen des Verzweigens wurden bereits in Kapitel 5 erläutert. Die wohl einfachste Ausführung ergibt sich, wenn man zum Verzweigen zwei Taktzyklen vorsieht. Womöglich lohnt es sich, auch den zweiten Zyklus auszunutzen (Sache der Entwurfsoptimierung). Typische RISC-Prozessoren führen im zweiten Zyklus stets den nachfolgenden Befehl aus. Wir aber können in den Verzweigungsmikrobefehlen zusätzliche Anweisungen vorsehen, um die Nutzung des zweiten Zyklus zu steuern (s. Kapitel 5, S. 252 bis 257).

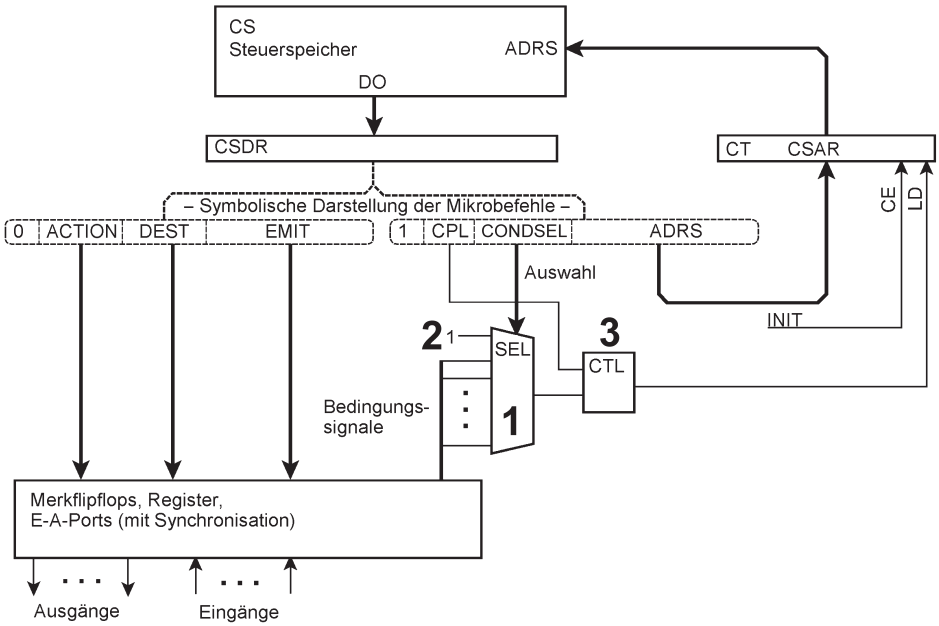


Abb. 6.23 Ein einfacher Branch Sequencer. 1 - eine einzige Auswahlschaltung für die Verzweigungs-, Warte- und Rücksetzbedingungen von Abb. 6.19; 2 - die feste Bedingung zum unbedingten Verzweigen; 3 - Ladesteuerung. Hier wird die ausgewählte Bedingung ggf. invertiert (CPL-Bit).

Warten und verzweigen

Die Frage ist, wie man beides miteinander kombiniert. Sollen es getrennte Mikrobefehlswirkungen sein oder soll ein einziges Mikrobefehlsformat Anweisungen für beide Zustandsübergänge enthalten?

Wartemikrobefehle

Warten und Verzweigen sind voneinander unabhängig. Wartemikrobefehle lassen sich beispielsweise im Format von Abb. 6.21a auslegen. Es wird auf eine auswählbare Bedingung gewartet. Abb. 6.24 zeigt, wie der Branch Sequencer von Abb. 6.23 entsprechend erweitert werden kann.

Warten mit Zeitkontrolle (Timeout)

Wenn die Wartebedingung nie erfüllt wird, bleibt die Maschine hängen. Eine naheliegende Abhilfe ergibt sich, indem man eine Watchdog-Zeitkontrolle einführt, wie sie von den Mikrocontrollern her bekannt ist. Für eine so pauschale Überwachung muß man aber vergleichsweise lange Zeiten zulassen[16]. Um kurze Zeitintervalle gezielt zu überwachen, kann man die Zeitkontrolle direkt mit den Wartemikrobefehlen verbinden (Abb. 6.25). Die Zeitzählung mit einem

16. Einige zehn Millisekunden sind typisch. Vgl. die Watchdog-Zeitgeber der Mikrocontroller.

Rückwärtszähler wurde bereits in Abb. 6.9 gezeigt. Hier dient die Zeitzählung zur Zeitkontrolle. Die Schaltung wartet auf eine Bedingung und verzeigt dann zur Fehlerbehandlung, wenn sie in der vorgegebene Zeit nicht auftritt (Timeout). Im Beispiel wird hierzu ein Rücksetzen erzwungen. Es ist von den Mikrocontrollern her bekannt, Merkflipflops zu setzen, aus denen das Rücksetzprogramm die aktuelle Ursache des Rücksetzens abfragen kann. Solche Einzelheiten sind hier nicht dargestellt. Die Alternative zum Rücksetzen wäre ein Verzweigen zu einer Timeout-Behandlungsroutine. Die Verzweigungsadresse kommt aus dem Mikrobefehl oder aus einem Voreinstellregister oder sie ist ein Festwert.

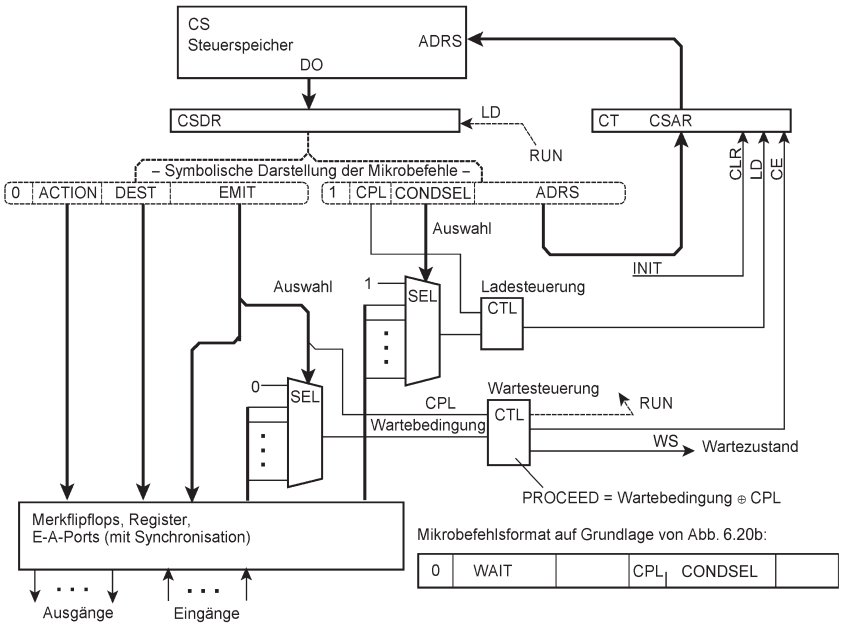


Abb. 6.24 Der mit Wartemikrobefehlen erweiterte Branch Sequencer. Die Wartesteuerung entspricht Abb. 6.6 oder 6.7. Wenn man das Weiterzählen der Mikrobefehlsadresse am Anfang des ersten Zyklus nicht aufhalten kann, muß das Laden des nächsten Mikrobefehls verhindert werden. Dazu wird das Signal RUN von der Wartesteuerung zum Ladeerlaubniseingang LD des Mikrobefehlsregisters geführt (vgl. Abb. 6.7). Die feste Bedingung (Festwert = 0) dient zum ständigen Warten = Anhalten im Endzustand.

Zeit abwarten (Zeitverzögerung)

Schaltungen hierzu wurden bereits in den Abb. 6.9 und 6.12 vorgestellt (S. 329ff). Die Zeitverzögerung bringt man in die Schaltung von Abb. 6.24 ein, indem man das PROCEED-Signal des Wartezeitzählers an die Wartebedingungsauswahl anschließt. Die Wartezeit wird aus einem ladbaren Register entnommen oder im Mikrobefehl selbst als Direktwert untergebracht. Zum Abwarten von Zeitintervallen s. weiterhin Abschnitt 8.4 (S. 470).

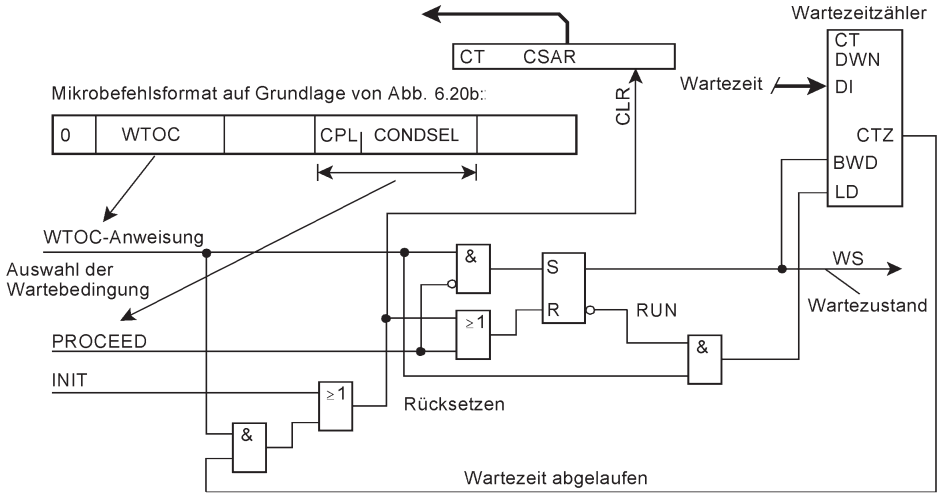


Abb. 6.25 Die Zeitüberwachung der Wartemikrobefehle. WTOC = Wait with Timeout Check. Die Wartezeit kommt aus dem Mikrobefehl oder aus einem Register oder sie ist ein Festwert.

Verzweigen und warten

Auch diese Kombination läßt sich auf Grundlage des Blockschaltbilds von Abb. 6.24 implementieren. Wann soll verzweigt werden: vor dem Warten, nach dem Warten oder während des Wartens? Solche Kombinationen wünscht man sich u. a. dann, wenn Interfacesignalfolgen zu steuern sind.

Vor oder nach dem Warten verzweigen

Das kann man auch mit getrennten Verzweigungs- und Wartemikrobefehle ausprogrammieren: Verzweigen, wenn Bedingung A, sonst warten auf Bedingung B oder warten auf Bedingung A, dann verzweigen, wenn Bedingung B.

Beispiel: Ein (fiktives) Interface hat zwei Handshaking-Signale, STROBE und REJECT. Eines von beiden muß eintreffen. Hat man ein horizontales Mikrobefehlsformat, das Auswahlfelder für beide Bedingungen und die Verzweigungsadresse enthält, könnte man z. B. auf STROBE warten und auf REJECT verzweigen[17]. Abb. 6.26 zeigt ein gegenüber Abb. 6.20 verkürztes horizontales Mikrobefehlsformat, mit dem man so programmieren kann.

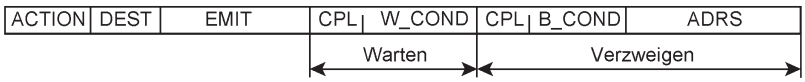


Abb. 6.26 Ein horizontales Mikrobefehlsformat zum Warten und Verzweigen.

17. Das Beispiel wurde bereits – mit Bezug auf Abb. 5.85 – auf S. 288 beschrieben.

In einem kurzen vertikalen Mikrobefehlsformat bringt man aber nur eine Bedingungsauswahl unter. Das gewünschte Verhalten läßt sich ohne Wartemikrobefehle als Warteschleife ausprogrammieren (vgl. Abb. 6.22b). Möchte man aber Wartemikrobefehle verwenden, kann man eine Wartebedingung REPLY = STROBE ∨ REJECT bilden (vgl. Abb. 5.85b), um zu warten, bis überhaupt eine Antwort kommt, und dann verzweigen, wenn z. B. REJECT signalisiert worden ist. Es kann aber sein, daß sich die Maschine die Art der Antwort (ob STROBE, ob REJECT) merken muß, damit der nachfolgende Mikrobefehl sie abfragen kann[18].

6.3 Algorithmische Steuerautomaten

Die Mikroprogrammsteuerwerke, die bisher beschrieben wurden, können nur Eingangssignale abfragen, Ausgangssignale erregen und darauf warten, daß Bedingungen eintreten oder daß Zeit vergeht. Alles, was komplexer ist – Speichern, Auswerten, Umcodieren, Vergleichen, Rechnen usw. – muß in den Schaltungen der Anwendungsumgebung erledigt werden.
Der nächste Entwicklungsschritt besteht darin, den Steuerautomaten so zu erweitern, daß er viele dieser Aufgaben selbst ausführen kann. Er soll so zu einer Plattform werden, die es ermöglicht, die Algorithmen der jeweiligen Anwendungsaufgabe zu implementieren. Das läuft letzten Endes auf eine programmierbare Universalmaschine hinaus. Wir wollen aber nicht einfach einen neuen Mikrocontroller bauen. Die Maschine muß nicht unbedingt Turing-vollständig sein. Sie soll nur soviel Universalität aufweisen, wie zur Implementierung der anwendungsseitigen Algorithmen jeweils erforderlich ist. Vor allem sollen – im Gegensatz zum üblichen Mikrocontroller – die Verarbeitungsfunktionen direkt und eng mit der Ein- und Ausgabe gekoppelt sein. Die Mikrobefehle werden dazu so lang generiert wie nötig.
Eine solche Maschine kann u. a. als E-A-Anschlußsteuerung oder spezialisierter E-A-Prozessor genutzt werden. Es gibt Aufgaben der Interfacesteuerung, bei denen es auf sehr kurze Reaktionszeiten ankommt (Mikrosekunden und weniger). Übliche Mikrocontroller können dafür auch dann zu langsam sein, wenn sie mit extrem hohen Taktfrequenzen betrieben werden. Um solche kurzen Reaktionszeiten einzuhalten, braucht man ein Mikroprogrammsteuerwerk mit horizontalen Mikrobefehlen, das bedarfsweise mit anwendungsspezifischen Adapterschaltungen ergänzt wird.

Die elementare Plattform

Sie besteht aus einem Mikroprogrammsteuerwerk, einem universellen Verarbeitungswerk (Arithmetic/Logic Unit, ALU), einem Lokalspeicher und anwendungsspezifischen E-A-Schaltungen. Das Mikroprogrammsteuerwerk ist ein programmierbarer Steuerautomat im Sinne der Abschnitte 6.1 und 6.2. Es kann auf beliebige Weise abgewandelt und erweitert werden, so wie es in Kapitel 5 dargelegt wurde. Abb. 6.27 zeigt ein besonders einfachen algorithmischen Steuerautomaten, Abb. 6.28 ein horizontales Mikrobefehlsformat. Die Operationen sind nur Ver-

18. Sache des Feinentwurfs und des Signalprotokolls (ob das abzufragende Signal am Interface noch aktiv sein wird oder schon verschwunden sein kann).

knüpfungen mit Direktwerten aus dem Mikrobefehl[19]. Um die Schnittstellen zur Anwendungsumgebung übersichtlich darzustellen, sind sie hier in E-A-Ports zusammengefaßt. Die Register der E-A-Ports sind adressierbar. Die E-A-Adressen sind zusätzliche Lokalspeicheradressen; die E-A-Register werden genauso angesprochen wie der Lokalspeicher (Memory-mapped I/O). Der E-A-Adapter sorgt dafür, daß die E-A-Zugriffe wie Lokalspeicherzugriffe ablaufen. Auch Voreinstellregister (Staticizers) sind wie E-A-Ports angeschlossen.

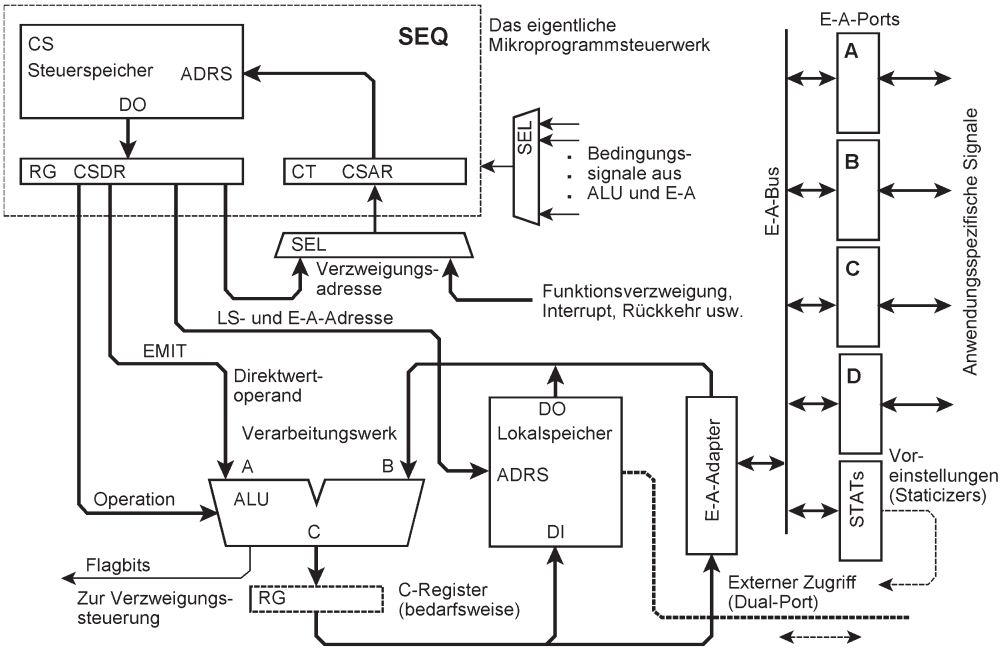


Abb. 6.27 Ein Mikroprogrammsteuerwerk wurde mit einem Lokalspeicher und einem Verarbeitungswerk erweitert. Das ergibt einen einfachen algorithmischen Steuerautomaten, der beispielsweise als speicherprogrammierbare Steuerung (SPS; Programmable Logic Controller PLC) oder als E-A-Prozessor eingesetzt werden kann.

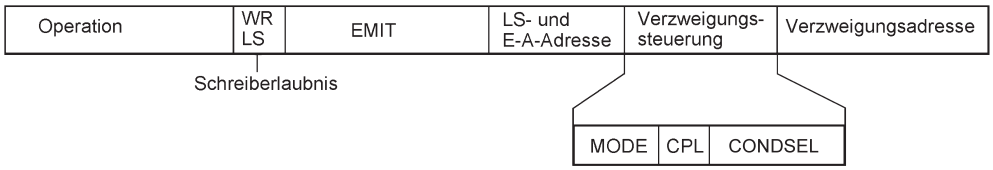


Abb. 6.28 Ein horizontales Mikrobefehlsformat.

19. Zu einem Ausführungsbeispiel vgl. [MD27].

Grundsätzliche Entwurfsgedanken:

- Die Schnittstellen zur Anwendungsumgebung nicht unbedingt so aufbauen wie die Ports und peripheren Funktionseinheiten der Mikrocontroller (obwohl man das natürlich tun kann). Weshalb müssen es universelle programmierbare Ports sein (mit Richtungsumsteuerung usw.), wenn man die Adapterschaltungen auf dem FPGA anwendungsspezifisch so synthetisieren kann, wie man sie braucht?
- Die Maschine einfach halten, aber genügend Verarbeitungsbreite und Adreßraum vorsehen. Wenn es zweckmäßig ist, 27 Bits auf einmal zu bewegen, wird die Maschine so generiert ...
- Mikrobefehlsformate und Mikrobefehlsadressierung so wählen, wie es jeweils zweckmäßig ist. Welche Steueranweisungen und Parameter (Direktwerte, Auswahl- und Operationscodes, Adressen usw.) braucht die Maschine in jedem Zyklus? Was ist zweckmäßiger, alles auf einmal im Mikrobefehl anzubieten oder teils in Voreinstellregistern (Staticizers) zu halten?
- Es ist ohne weiteres möglich, die Mikroprogrammsteuerung um Funktionen der Wartezeitzählung, der Unterbrechungsauslösung und des Unterprogrammaufrufs zu erweitern. Das wird hier nur der Überschaubarkeit wegen nicht dargestellt.
- Die Kommunikation im System. Das Problem ergibt sich, wenn die Maschine Teil eines komplexeren Systems ist, beispielsweise als Peripherieprozessor irgendwo in einem größeren FPGA. Dann spielt sie typischerweise die Rolle eines Slave. Es liegt nahe, entsprechende Koppelstufen zu einem internen Systembus (SoC Bus System[20]) vorzusehen. Das ist aber recht aufwendig. Eine dem Prinzip nach einfache, aber effektive Lösung besteht darin, den Lokalspeicher als Dual-Port-RAM zu implementieren und einen externen Zugriffsweg einzurichten. Eine bewährte Sparlösung ([MD27]): Wird die Maschine als Slave angesprochen, kann der jeweilige Master auf den Lokalspeicher zugreifen und den Mikroprogrammablauf beeinflussen (Rücksetzen, Betriebsarten des Debugging und der Diagnose usw.). Ist sie selbst Master, kann sie lediglich Interrupts auslösen[21].
- Virtuelle Peripherie. Programmieren statt entwerfen und bauen. Das Mikroprogramm emuliert auch periphere Funktionseinheiten, wie Zähler, Zeitgeber oder Impulsmustergeneratoren (Timing Pattern Generators). Hierzu wird das Prinzip des Hardware-Multitasking implementiert. Die Maschine schaltet dann zyklusweise zwischen mehreren unabhängigen Mikroprogrammabläufen um[22].

Verarbeitungsoperationen

Die Maschine soll nicht zu kompliziert und nicht zu aufwendig werden. Deshalb beschränken wir uns zunächst auf eine Operation zu einer Zeit mit maximal zwei Operanden (A, B), die ein Ergebnis (C) und einige begleitende Bedingungsbits (Flagbits) liefert (C := A **op** B).

20. Hier ist vor allem auch an Industriestandards zu denken wie CoreConnect, Amba, OCP, Fast Simplex Link usw.
21. Kostet viel weniger als die Unterstützung echter Masterzugriffe auf den Speicheradreßraum des Systems.
22. Sie funktioniert dann so ähnlich wie die Peripherieprozessoren der guten alten CDC 6600 ...

Für anwendungsspezifische Operationen können Einzweckschaltungen oder Zuordnerspeicher vorgesehen werden. Die arithmetisch-logische Einheit (Arithmetic/Logic Unit, ALU) ist die naheliegende Lösung, um Universalität darzustellen. Welche Operationen soll sie ausführen? Hierzu kann man sich von den Befehlslisten der bewährten Mikrocontroller und den Operationen der herkömmlichen ALU-Schaltkreise anregen lassen. Auch kann man die im jeweiligen Anwendungsfall zweckmäßigen Operationen als Verhaltensbeschreibung erfassen, um das Verarbeitungswerk zu synthetisieren[23].

Lokalspeicher und Direktwert

Für viele Steuerungsaufgaben reicht die Verknüpfung mit einem Direktwert aus, der vom aktuellen Mikrobefehl geliefert wird. Im Lokalspeicher kann man Zustandsbits, Merkbits, gelesene Eingangssignale usw. speichern, Bitmuster zum Ausgeben vorbereiten usw. So müßte man in den Branch Sequencers des Abschnitts 6.2 Merkbits mit Flipflops implementieren. Man würde eigens Mikroanweisungen brauchen, um sie zu setzen und zu löschen. Um sie als Verzweigungsbedingung abzufragen, müßten sie an die Bedingungsauswahl angeschlossen werden. Im Steuerautomaten von Abb. 6.27 wären es hingegen Bits im Lokalspeicher, die man mit ODER-Verknüpfungen setzen und mit UND-Verknüpfungen löschen oder abfragen kann[24].

Abb. 6.29a entspricht dem Verarbeitungswerk von Abb. 6.27. Der Lokalspeicher muß aber die direkte Rückführung vom Ausgang der ALU auf seinen Dateneingang zulassen. Ein Beispiel dafür sind Block-RAMs, die die Betriebsart Read-before-Write unterstützen. Beim Schreiben erscheint dann der bisherige Inhalt am Datenausgang. Die adressierte Speicherzelle verhält sich praktisch wie ein Flipflopregister in einem synchronen Schaltwerk. Wenn der Speicher eine solche Betriebsart nicht aufweist, muß man Register zwischenschalten (Abb. 6.29b).

Lokalspeicher und A-Register – die Einadreßmaschine

In komplizierteren Algorithmen sind nicht nur Daten mit Direktwerten, sondern auch Daten mit Daten zu verknüpfen. Wir behalten zunächst den einfachen Lokalspeicher bei und sehen ein Arbeitsregister vor, das auch als Akkumulator genutzt werden kann, das A-Register. Wenn das Ergebnis ins A-Register zurückgeschrieben wird, wirkt es als Akkumulator, sonst als Arbeitsregister. Wo aber lassen wir den Direktwertoperanden einfließen? Abb. 6.30 veranschaulicht die Alternativen.

a) In den A-Eingang der ALU alternativ zum A-Register. Damit werden folgende Verknüpfungen unterstützt:

- <A-Register> **op** <LS/E-A>,
- Direktwert **op** <LS/E-A>.

23. Praxistip: Nicht zuviel weglassen. Die üblichen universellen Operationen (vgl. Anhang 2) sollten drinbleiben, um ggf. darauf zurückgreifen zu können (Fehlerbeseitigung, Erweiterungen (Updates) usw.).
24. Mikrobefehle zum Abfragen führen die Verknüpfung aus und verzweigen gemäß der jeweiligen Bedingung, speichern aber kein Ergebnis.

b) In den B-Eingang der ALU alternativ zum Lokalspeicher. Damit werden folgende Verknüpfungen unterstützt:

- <A-Register> **op** <LS/E-A>,
- <A-Register> **op** Direktwert.

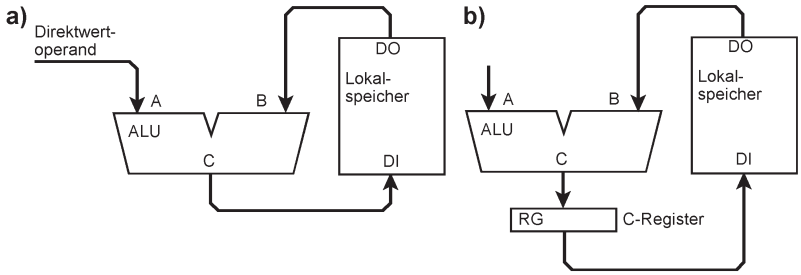


Abb. 6.29 Verarbeitungswerke, die nur Direktwertoperanden verknüpfen. a) Es gibt Speicher, die die Rückführung vom Datenausgang auf den Dateneingang unterstützen, z. B. manche Block-RAMs. b) Wenn die Rückführung nicht möglich ist, muß man Register zwischenschalten. Hier ist es das C-Register am Ausgang.

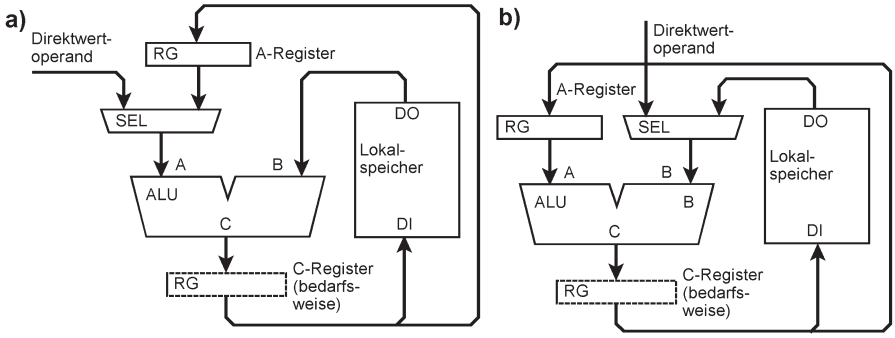


Abb. 6.30 Verarbeitungswerk einer Einadreßmaschine. Das A-Register wirkt als Arbeitsregister oder als Akkumulator.

Zwei Lokalspeicher – die Zweiadreßmaschine

An die Stelle des A-Registers tritt ein zweiter Lokalspeicher. Das ist eine seit langem bewährte Lösung. Wie aber sollen die Lokalspeicher adressiert werden? Zwei Alternativen liegen nahe: als unabhängige Speicher mit getrennten Adreßräumen oder als Speicher mit einem einzigen Adreßraum und zwei Zugriffswegen (Dual-Port Memory).

Zwei unabhängige Speicher

Die Frage ist, wie man sie nutzt. Wie verknüpft man zwei Operanden, die im selben Speicher untergebracht sind? Man müßte zuvor einen Operanden in den anderen Speicher kopieren. Womöglich ist es zweckmäßiger, unterschiedliche Speicher vorzusehen (Abb. 6.31). Hier ist der

Lokalspeicher A eine Sammlung von Universalregistern, die vor allem als Akkumulatoren genutzt werden[25]. Der Lokalspeicher B ist größer. Er ist schon beinahe ein Arbeitsspeicher. Auch sind hier die E-A-Ports angeschlossen. Dieser Lokalspeicher kann zudem – insgesamt oder in Teilen – als Dual-Port-RAM ausgebildet werden, um externe Zugriffe zu unterstützen.

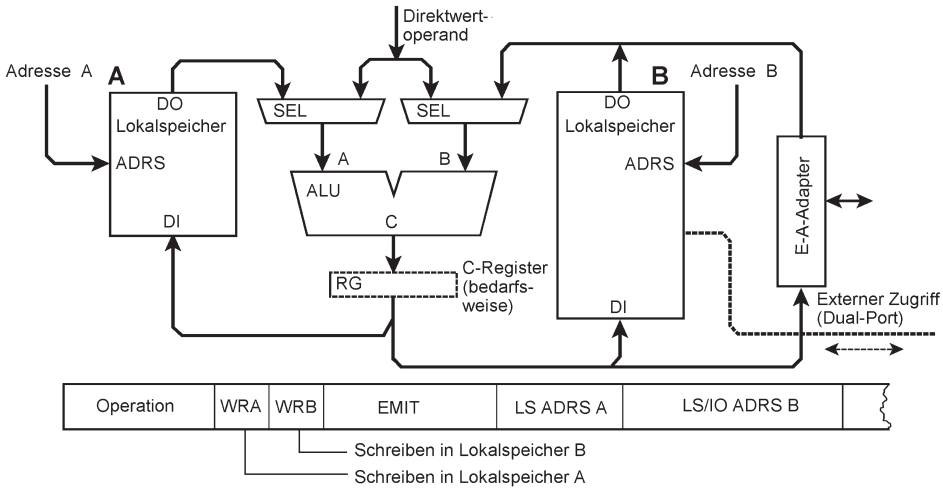


Abb. 6.31 Verarbeitungswerk mit zwei unabhängigen Lokalspeichern. Lokalspeicher A enthält die Arbeitsregister und Akkumulatoren (beispielsweise 8 bis 64 Maschinenwörter), Lokalspeicher B die Variablen und Daten (beispielsweise 256 bis 4096 Maschinenwörter).

Zwei Speicher, ein Adreßraum (Dual-Port Memory)

Beide Speicher haben den gleichen Inhalt. Auch das ist eine bewährte Lösung[26]. Abb. 6.32 zeigt, wie dieses Prinzip mit einem Dual-Port-RAM implementiert werden kann. Die Schaltung unterstützt folgende Verknüpfungen:

- <LS> := <LS-A> **op** <LS-B>,
- <LS> := <LS-A> **op** Direktwert.

Wenn man keine Dual-Port-RAMs hat, muß man zwei einfache Speicher vorsehen (Abb. 6.33). Üblicherweise wird auf eine der beiden Adressen geschrieben, in unseren Beispielen auf die Adresse A. Die betreffende Lokalspeicherzelle wirkt dann als eine Art Akkumulatoregister. Ein echter Dual-Port-RAM gestattet auch Schreibzugriffe auf beide Adressen (wie in Abb. 6.32 angedeutet). Sind es zwei einfache RAMs, muß man das Ergebnis in beide auf die gleiche Adresse schreiben. Im Beispiel von Abb. 6.33 wird hierzu der Adreßeingang des Lokalspeichers B auf die Adresse A umgeschaltet.

25. Wie in vielen Universalmaschinen üblich. Man denke an S/360, DEC PDP-10 usw. Zu unterschiedlich großen Lokalspeichern vgl. auch die Register Files der CDC 5600 (S. 517).
26. Beispiel: S/370 Modell 145 (Abschnitt 9.1.2, vor allem Abb. 9.10, S. 508).

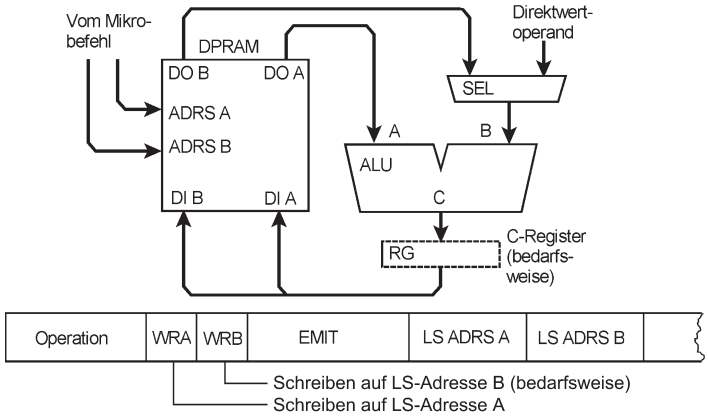


Abb. 6.32 Verarbeitungswerk einer Zweiadreßmaschine. Zweifachzugriff zum Lokalspeicher durch Einsatz eines Dual-Port-RAM (DPRAM).

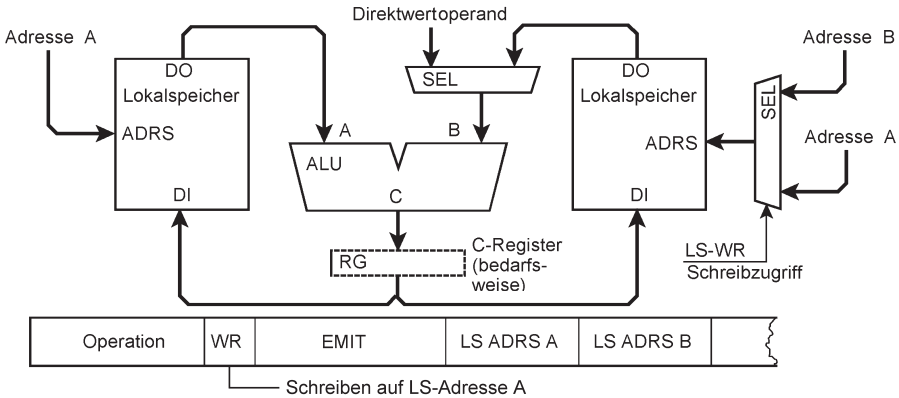


Abb. 6.33 Nachbildung eines Dual-Port-RAM mit zwei einfachen RAMs. Die Verknüpfungen: <LS-A> := <LS-A> **op** <LS-B> und <LS-A> := <LS-A> **op** Direktwert.

Drei Zugriffswege – die Dreiadreßmaschine

Dreiadreßmaschinen erfordern drei Zugriffswege, zwei zum Lesen und einen zum Schreiben. Entsprechende RAM-Arrays sind aber nicht immer verfügbar. Viele Block-RAMs in FPGAs haben nur zwei Adreßdecoder (Dual-Port-RAM). Damit kann man – wenn man mit einem einzigen Taktzyklus auskommen will – nur Zweiadreßmaschinen bauen. Eine Doppelanordnung von Dual-Port-RAMs ermöglicht es, zwei Lese- und einen Schreibdatenweg zu implementieren (Abb. 6.34). Die Schaltung unterstützt folgende Verknüpfungen:

- <LS-Adresse 3> := <LS-Adresse 1> **op** <LS-Adresse 2>,
- <LS-Adresse 3> := <LS-Adresse 1> **op** Direktwert.

Die Dateneingänge der DPRAM-Ports A und die Datenausgänge der DPRAM-Ports B werden nicht genutzt. Sie stehen ggf. für die Ein- und Ausgabe oder für anwendungsspezifische Tricklösungen zur Verfügung.

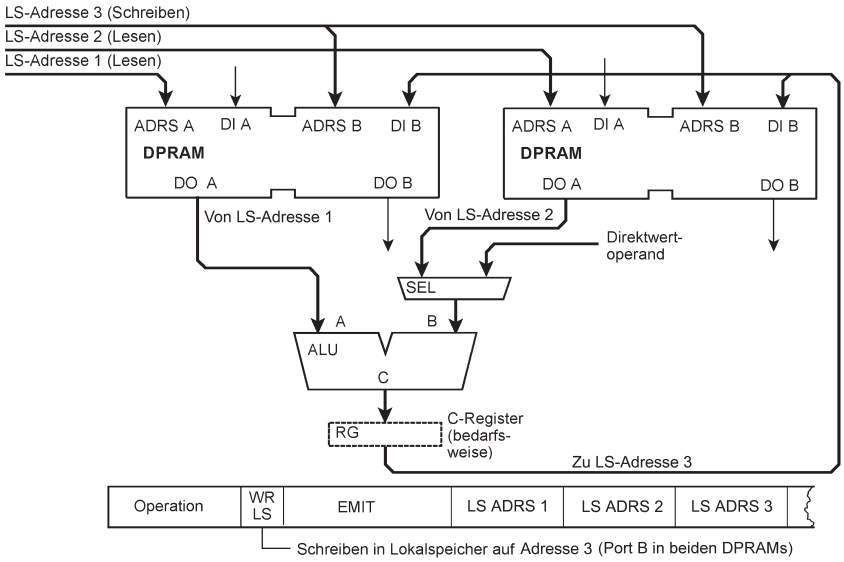


Abb. 6.34 Ein Lokalspeicher mit drei Zugriffswegen, aufgebaut mit Block-RAMs, die zwei Zugriffswege aufweisen (Dual-Port).

Schaltungsbeispiel einer Einadreßmaschine

Abb. 6.35 zeigt eine Weiterentwicklung der Schaltung von Abb. 6.27 zur Einadreßmaschine. Der Inhalt einer Lokalspeicherzelle oder eines E-A-Registers oder ein Direktwert im Mikrobefehl kann mit dem Inhalt des A-Registers verknüpft werden. Das Format <A> **op** Direktwert (vgl. Abb. 6.30b) wurde bevorzugt, weil es keine Lokalspeicheradresse braucht und weil das A-Register oft als Akkumulator genutzt wird. Um den Inhalt einer Lokalspeicherzelle oder eines E-A-Registers mit einem Direktwert zu verknüpfen (<LS/E-A> **op** Direktwert), muß der Direktwert ins A-Register geladen werden. Das erfordert einen zusätzlichen Mikrobefehl, der den Direktwert über die ALU dem A-Register zuführt (Gate-Thru-Operation). Für Einzelbitoperationen und -abfragen gibt es Mikrobefehle, die das A-Register nicht verwenden.

Mikrobefehlsformate und Wirkprinzipien

Allein schon wegen der Überschaubarkeit können wir hier nur kleine, einfache Schaltungen darstellen. Es gibt aber keine grundsätzlichen Grenzen. Die Maschine kann beliebig groß (im Sinne der Speicherkapazitäten, der Länge der Mikrobefehle, der Verarbeitungsbreite usw.) und beliebig kompliziert ausgelegt werden. Der Grundgedanke: die Maschine so bauen, wie es erforderlich ist, um die Anwendungsaufgaben zu erledigen, aber nur soweit an die Turing-Vollständigkeit annähern wie jeweils nötig.

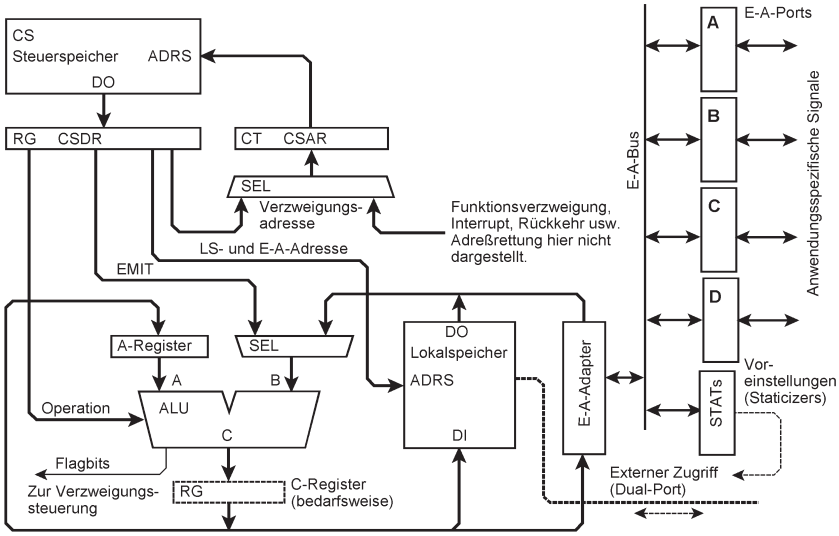


Abb. 6.35 Ein algorithmischer Steuerautomat, der als Einadreßmaschine ausgeführt ist. Man kann schon viel programmieren, aber noch nicht alles, denn die Maschine ist nicht Turing-vollständig[27].

Horizontale Mikrobefehle

Alle Steuerbits, Anweisungsfelder und Direktwertfelder werden aneinandergereiht. Abb. 6.36 veranschaulicht das prinzipielle Mikrobefehlsformat einer Einadreßmaschine, Abb. 6.37 gibt einen Überblick über Mikrobefehlsformate von Zwei- und Dreiadreßmaschinen.

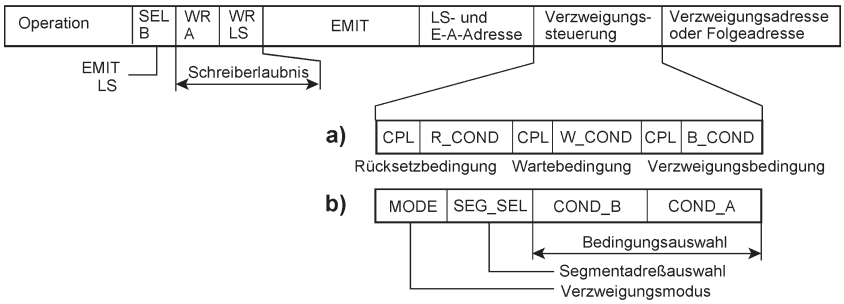


Abb. 6.36 Das prinzipielle horizontale Mikrobefehlsformat einer Einadreßmaschine. a) Mikrobefehlsadressierung durch Adreßzählung und Verzweigung; b) Folgeadresse im Mikrobefehl und Einspeisen der Bedingungen in Bitpositionen der Folgeadresse.

27. Hierzu müßte sie Vorkehrungen zur Adreßrechnung aufweisen.

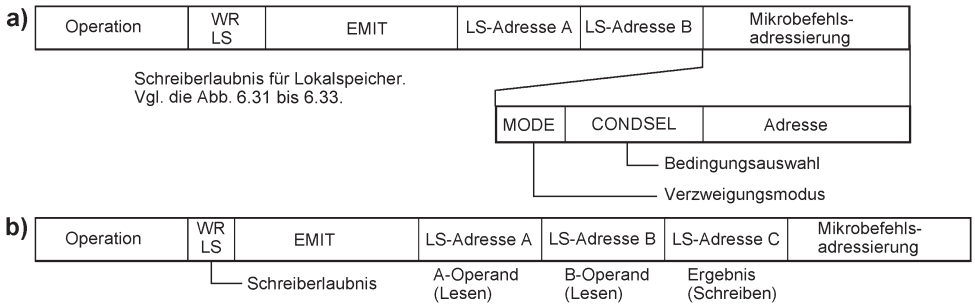


Abb. 6.37 Weitere horizontale Mikrobefehlsformate. a) Zweiadreß-, b) Dreiadreßmaschine. Zur Mikrobefehlsadressierung vgl. auch Abb. 6.36a, b.

Die Mikrobefehle von Abb. 6.37 ähneln den Befehlen der typischen RISC-Architekturen. Maschinen auf Grundlage solcher Mikroprogrammsteuerwerke sind aber einfacher als RISC-Prozessoren. Man kann sie so generieren, daß sie nur so aufwendig werden wie jeweils erforderlich. Einerseits kann man sparen (alles weglassen,was nicht benötigt wird), andererseits die Maschine an die Anwendungsaufgaben anpassen (Ausstattung mit Ressourcen, Verarbeitungsbreite, Funktionsverzweigung usw.). Es gibt kein Pipelining, die Verzweigungsmechanismen sind effektiver, und die Ein- und Ausgabe kann direkt in die Mikrobefehle eingebunden werden[28]. Da ein typischer Mikrobefehl mehr leistet als ein typischer RISC-Befehl, kommt man oftmals mit einer geringeren Taktfrequenz aus[29].

Horizontale oder vertikale Mikrobefehle?

Die Mikrobefehlsbits, die Abläufe steuern, Funktionen auswählen, Speicher adressieren usw., müssen irgendwo gespeichert sein. Es ist nur die Frage, wie man sie aus dem Mikroprogrammspeicher abruft:

- Horizontal = alle auf einmal. Dann kann man die Signale direkt an die zu steuernden Funktionseinheiten anlegen.
- Vertikal = in kürzeren Abschnitten. Alle Bits, die man im aktuellen Zyklus braucht, die aber der aktuelle Mikrobefehl nicht mitbringt, müssen dann in Registern bereitstehen (Voreinstellung).

Aus dieser Sicht ist die Mikrobefehlsformatgestaltung Sache der Entwurfsoptimierung. So kann man damit beginnen, die Maschine als Ressourcenvektormaschine zu entwerfen. Alle Ressourcen vorsehen, wie man es für zweckmäßig hält. Das erste Mikrobefehlsformat ergibt sich, indem man alle Steuerbits, Anweisungsfelder usw., die die Ressourcen benötigen, aneinanderreiht. Wie lang es wird, ist zunächst gleichgültig. Erst dann wird optimiert:

28. In Kapitel 7 zeigen wir, wie man RISC-Prozessoren in diesem Sinne erweitern kann.
29. Weniger Schaltvorgänge (di/dt) bedeuten geringere Leistungsaufnahme und Störemissionen.

- Welche Ressourcen können sinnvollerweise gleichzeitig gesteuert werden, welche nur nach dem Prinzip eine zu einer Zeit? Ist es nur eine zu einer Zeit, liegt es nahe, Mikrobefehlsbits gemeinsam zu nutzen (Abb. 6.38a, b).
- Gibt es Register, Speicher usw., auf die man ohnehin nicht gleichzeitig zugreifen kann? Wenn man jeweils nur ein Register oder einen Speicher auswählen oder laden kann, genügt ein binär codiertes Auswahlfeld für alle.
- Wieviele Bits und Felder werden in den Mikrobefehlen wirklich verwendet? Wir schreiben die Mikroprogramme zunächst mit den fiktiven langen Mikrobefehlen (oder lassen sie vom Entwicklungssystem so erzeugen). Dann untersuchen wir, was herausgekommen ist. Aller Voraussicht nach wird man die Mikrobefehle danach sortieren können, welche Bits und Felder zusammen genutzt werden und welche nicht. Dann liegt es nahe, kürzere Mikrobefehle zu bilden, mit jenen Bits und Feldern, die auffallend oft gemeinsam vorkommen. So ergeben sich "diagonale" oder vertikale Formate (Abb. 6.38c, d, e).
- Wenn man sich schon eine langes Mikrobefehlsformat leistet: kann man dann alles, was dort angewiesen wird, auch in einem Maschinenzyklus angemessener Länge erledigen? Wieviele Taktphasen würde das erfordern? Wenn es mehr als zwei bis vier werden, lohnt sich womöglich die Aufteilung in kürzere Mikrobefehle und Maschinenzyklen.
- Müßte man, um die Mikrobefehle zu kürzen, eigens Voreinstellregister laden oder Flipflops (Staticizers) setzen?
- Lohnt sich der Aufwand? Er lohnt sich womöglich dann, wenn man beim Mikroprogrammspeicher mehr spart als man in die zu steuernden Funktionseinheiten einbauen müßte. Das kann beispielsweise der Fall sein, wenn viele der langen horizontalen Mikrobefehle immer wieder gleiche Bitmuster enthalten. Es genügt dann, sie am Anfang des jeweiligen Mikroprogrammabschnitts in die Staticizer zu laden (Voreinstellung). Wenn aber die Mikroprogrammabläufe immer wieder unterbrochen werden, kann es sein, daß es sich nicht lohnt, weil man bei jedem Break-In die Voreinstellungen retten und wieder zurückschreiben müßte.

Abb. 6.38a veranschaulicht einen Ausschnitt aus einem langen Mikrobefehl des Ressourcenvektorentwurfs. R1, R2, R3 sind Ressourcen, R1.1, R1.2 usw. sind deren Mikrobefehlsfelder. Die Ressourcen R1 und R2 können aber nicht gleichzeitig genutzt werden. Somit ergibt sich keine Leistungsminderung, wenn wir das lange Format in zwei kürzere zerlegen, eines für jede Ressource (Abb. 6.38b). Um sie voneinander zu unterscheiden, erhalten sie einen Funktionscode, der die Ressource auswählt (FC R1, FC R2). Abb. 6.38c veranschaulicht einen langen Mikrobefehl mit vielen Feldern. In der Programmierpraxis werden sich zumeist Mikrobefehle ergeben, die jeweils nur einige dieser Felder verwenden. Abb. 6.38d zeigt zwei Beispiele, Abb. 6.38e ein weiteres Beispiel. Die mit A, B usw. gekennzeichneten Felder werden genutzt, die leeren nicht. Die genutzten Felder werden in einem kürzeren Mikrobefehl zusammengefaßt, dem ein Funktionscode (FC) vorangestellt ist. In Abb. 6.38d wurden die Felder der beiden langen Mikrobefehle in einem kürzeren Mikrobefehl untergebracht. Abb. 6.38e ist ein Beispiel eines verkürzten Formats mit ungenutzten Bitpostionen.

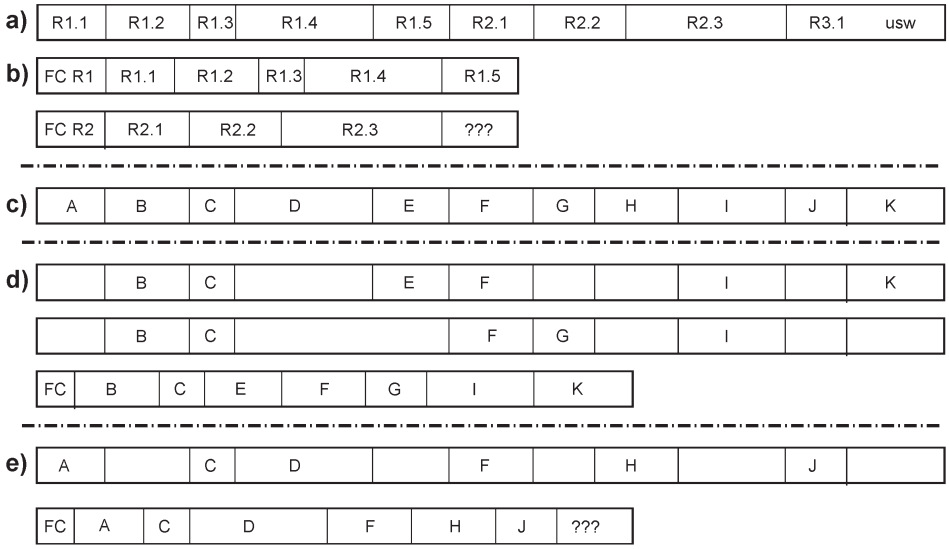


Abb. 6.38 Mikrobefehlsoptimierung. a) und b) Optimierung auf Grundlage der Ressourcennutzung. c), d), e) Optimierung auf Grundlage der Bits und Felder, die in den Mikroprogrammen tatsächlich zusammen genutzt werden. Die optimierten (verkürzten) Mikrobefehle haben einen Funktions- oder Formatcode FC. Ungenutzte Bitpositionen (???) können als Änderungsreserve verbleiben oder für untergeordnete Zwecke genutzt werden (Debugging, Diagnose usw.).

Vertikale Mikrobefehle

In Zwei- oder Mehradreßmaschinen sind vertikale Mikrobefehlsformate Sache der Optimierung. Im Grunde läuft es darauf hinaus, das lange Mikrobefehlsregister in kürzere Voreinstellregister zu zerlegen, die nacheinander geladen werden. Demgegenüber liegt es nahe, Einadreßmaschinen mit vertikalen Mikrobefehlen zu steuern, die wie typische Einadreßbefehle aussehen (Abb. 6.39 und 6.40).

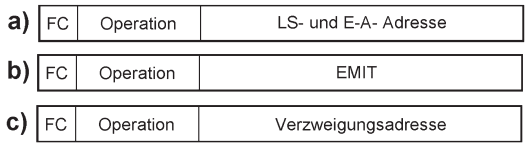
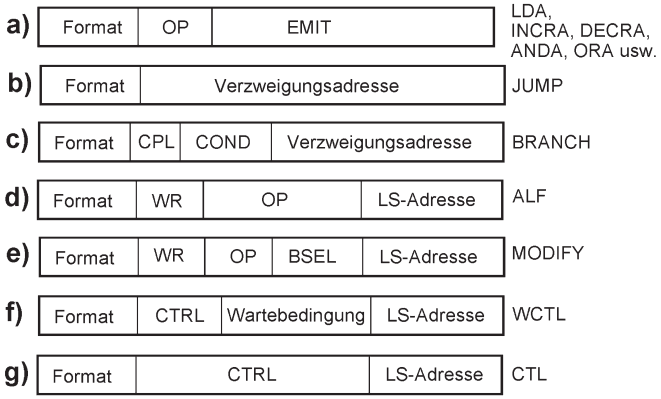


Abb. 6.39 Vertikale Mikrobefehle im Überblick. FC = Formatcode.

Die Erläuterungen beziehen sich auf eine Maschine gemäß Abb. 6.35:

a) Lokalspeicheroperationen. Es sind Ein- oder Ausgaben, Verknüpfungen mit dem A-Register und Operationen mit dem adressierten Lokalspeicherinhalt, die ohne einen zweiten Operanden auskommen (Negieren, Setzen, Löschen und Abfragen von Bits, Erhöhen und Vermindern um 1 (Inkrement / Dekrement), Verschieben usw.).

b) Direktwertoperationen. Der Direktwert kann in Register oder periphere Funktionseinheiten geladen oder mit dem Inhalt des A-Registers verknüpft werden.
c) Verzweigungen. Im Operationsfeld sind die Art der Verzweigung und die Bedingungsauswahl codiert.



OP	Operationscode
EMIT	Direktwert
CPL	Die Bedingung invertieren (Complement)
JUMP	Unbedingt verzweigen
ALF	Arithmetic/Logic Functions
BSEL	Bit Select = Bitauswahl
WCTL	Wait and Control = Warten und Steuerwirkungen ausüben
COND	Bedingungsauswahl
WR	Resultatspeicherung = im A-Register, im Lokalspeicher, in beiden oder gar nicht
CTRL	Steuersignale und Steueranweisungsfelder
BRANCH	Bedingt verzweigen
MODIFY	Modifzieren = Verändern des LS-Inhalts (Bits setzen, löschen usw.)
CTRL	Control = Steuerwirkungen ausüben

Abb. 6.40 Beispiele vertikaler Mikrobefehle.

a) Direktwertverknüpfungen mit dem Inhalt des A-Registers. LDA = Laden, INCRA = Addieren, DECRA = Subtrahieren, ANDA = UND-Verknüpfung usw. Weitere Operationen betreffen die Nutzung des Direktwertes in anderen Funktionseinheiten, beispielsweise zu Einstellzwecken.
b) Unbedingte Verzweigung im gesamten Adreßraum des Mikroprogrammspeichers. Der Mikrobefehl lädt auch die neue Segmentadresse. Zur Segmentierung s. Abschnitt 5.2.2.
c) Bedingte Verzweigung innerhalb eines Segments (von beispielsweise 256 Mikrobefehlen).
d) Arithmetisch-logische Funktionen. Sie betreffen den Lokalspeicherinhalt allein oder sind Verknüpfungen mit dem Inhalt des A-Registers.
e) Verändern des Lokalspeicherinhalts (Löschen, Setzen, Abfragen usw.).

f) Warten und steuern. So lange warten, bis die Wartebedingung erfüllt ist. Dann die im CTRL-Feld angegebene Steuerwirkung ausüben. Diese kann mit einem Lokalspeicherzugriff verbunden sein. Beispiel: warten, bis eine Anfrage von einem Interface kommt, dann ein Byte aus dem Lokalspeicher ausgeben und das zugehörige Handshaking-Signal erregen.
g) Allgemeine Steuerwirkungen. Sie können mit einem Lokalspeicherzugriff verbunden sein. Beispiel: in einer peripheren Einrichtung etwas einstellen und ein Byte aus dem Lokalspeicher ausgeben.

Weiterentwickelte Einadreßmaschinen

Es gibt viele Quellen von Anregungen: die Maschinen der Pionierzeit, die Minicomputer, die ersten Mikroprozessoren usw. Die Maschine in Abb. 6.41 geht auf den echten Anfang der Entwicklungsgeschichte zurück, auf Konrad Zuses Z3[30].

Was die Z3 als Vorbild hier interessant macht, sind die besonders kurzen Befehle und das Registermodell der Verarbeitungseinheit. Es gibt zwei programmseitig zugängliche Register, R1 (A) und R2 (B). R1 (A) dient als Akkumulator, R2 (B) nimmt den zweiten Operanden auf. Die Z3 kommt mit sehr kurzen Befehlen aus, weil die Datentransporte und die Operationen voneinander getrennt sind (Load-Store-Architektur).

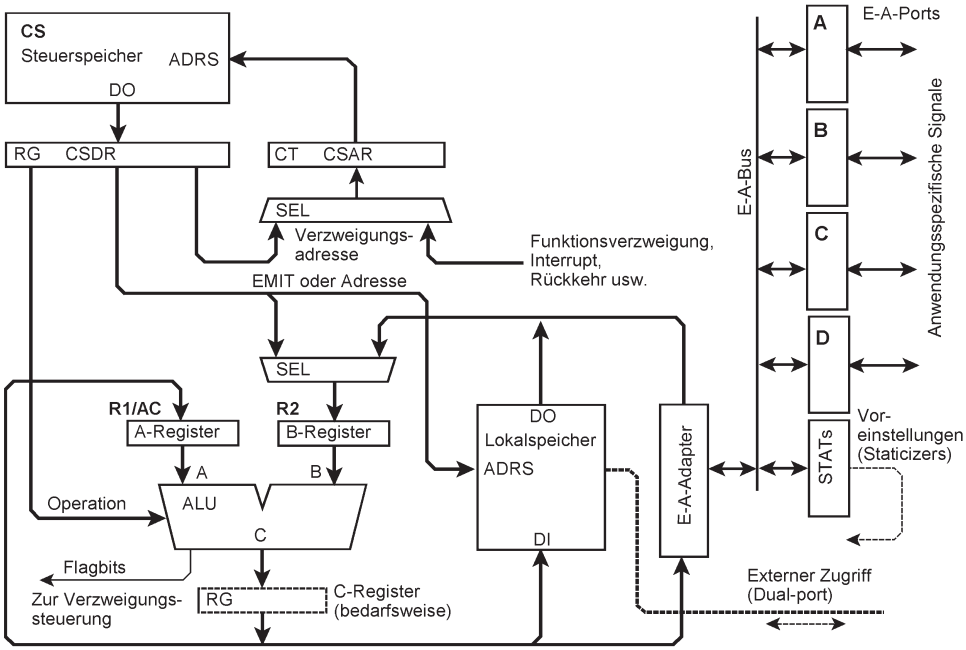


Abb. 6.41 Eine Einadreßmaschine, angeregt von Konrad Zuses Z3.

30. Vgl. beispielsweise [41], [89], [107] und [108]. Für eine ersten Überblick s. auch [56].

Abb. 6.42a zeigt die ursprünglichen Befehlsformate[31]. Die Z3 hat drei Befehlstypen, Laden (LD), Operationsausführung (OP) und Speichern (ST). Die Befehlsformate sind einfach und kurz – ein Typcode von 2 Bits und ein Operationscode oder eine Adresse von 6 Bits. Es ist ein Bytecode. Welches der beiden Register geladen wird, hängt vom Maschinenzustand ab. Es gibt zwei Zustände, den Grundzustand und den Rechenzustand. Ein im Grundzustand ausgeführter Ladebefehl bewirkt, daß der Akkumulator R1 (A) geladen wird[32]. Danach geht der Prozessor in den Rechenzustand über. Dann beziehen sich alle Ladebefehle nur noch auf das Register R2 (B)[33]. Alle Verknüpfungen haben die Form <R1> := <R1> **op** <R2>. Wird das Ergebnis aus dem Akkumulator R1 abgespeichert (Speicherbefehl), gelangt der Prozessor wieder in den Grundzustand, so daß R1 erneut geladen werden kann. In diesen Befehlsformaten bringt man 64 Lokalspeicheradressen und und 64 Operationen unter. Das dürfte für viele kleinere Steuerungsaufgaben gut ausreichen[34].

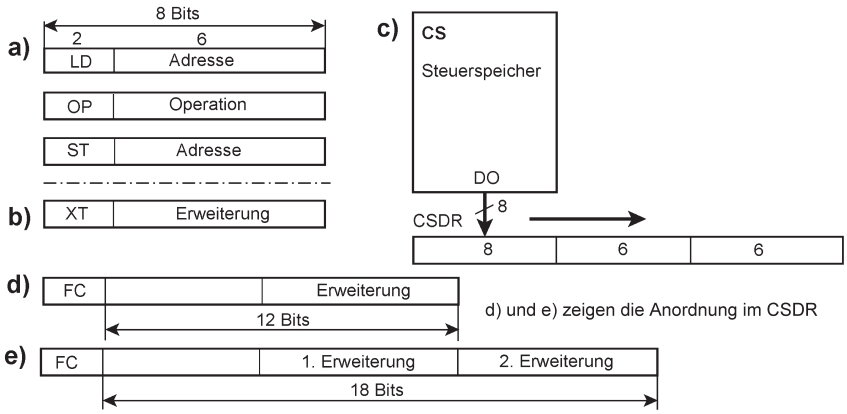


Abb. 6.42 Z3-Befehle und erweiterte Mikrobefehle. a) Z3 Originalbefehle; b) Erweiterungsbefehl; c) Prinzip der Erweiterung; d) Erweiterung auf 14 Bits; e) Erweiterung auf 20 Bits.

Die Load-Store-Architektur der Z3 erlaubt es, in schmalen Befehlsformaten vergleichsweise viele Adreßbits unterzubringen. Neben dem B-Register braucht man nur eine einfache sequentielle Steuerung, um die beiden Zustände darzustellen. Würde man das Format auf 16 Bits verlängern, hätte man 14 Adreßbits und 14 Bits für Operationscodes usw. zur Verfügung. Für einen kleinen Steuerautomaten ist das schon reichlich. Man könnte sich durchaus ein weiteres For-

31. Die folgende Darstellung ist kein Exkurs in die Technikgeschichte, sondern eine kurze Wiedergabe der hier bedeutsamen Architekturmerkmale mit eigenen Worten und in Anlehnung an übliche Begriffe und mnemonische Bezeichnungen.
32. In der Schaltung von Abb. 6.41 über das Register B. Erst B laden, dann dessen Inhalt über die ALU zum A-Register weiterreichen (Operation Gate-Thru B).
33. Deshalb ist die Sparlösung zum Laden des A-Registers hier möglich. Im Grundzustand kommt es nicht auf den Inhalt von B an, im Rechenzustand kann man A nicht laden.
34. Vgl. die Ausstattung der kleineren Mikrocontroller.

matcodebit leisten und so zu Formaten ähnlich Abb. 6.40 kommen. Dabei könnte man auch zwei Lademikrobefehle vorsehen (LDA, LDB) und so die beiden Zustände und die sequentielle Steuerung einsparen.
Nun fehlt der Z3 noch einiges, vor allem die bedingten Verzweigungen. Abb. 6.42b zeigt das verbleibende vierte Befehlsformat. Es zu Erweiterungszwecken auszunutzen liegt nahe (XT = Extend). Wenn man breitere Mikrobefehle vorsieht (16 Bits oder mehr), könnte man im Anschluß an die zwei Bits des Formatcodes bedingte Verzweigungen, das Laden von Direktwerten, Steuerwirkungen in der Peripherie usw. unterbringen. Die erweiterten Formate ähneln dann denen von Abb. 6.40.
Wir wollen uns hier aber darauf beschränken, ein Erweiterungsprinzip vorzustellen, das auf dem ursprünglichen Bytecode beruht. Der Steuerspeicher hat eine Zugriffsbreite von 8 Bits (Abb. 6.42c). Das Mikrobefehlsregister CSDR ist länger. Man kann sich vorstellen, daß es wie ein Schieberegister wirkt. Jedes Byte, das geladen wird, verschiebt den bisherigen Inhalt um 6 Bits (in der Abbildung nach rechts). Von den Erweiterungsmikrobefehlen (XT = Extend) werden nur die 6 Inhaltsbits geladen. Um einen Mikrobefehl der Formate LD, OP, ST zu erweitern, werden XT-Mikrobefehle vorangestellt. Mit nur einem XT-Mikrobefehl ergibt sich ein 14-Bit-Mikrobefehl mit dem Formatcode (FC) und 12 Inhaltsbits (Abb. 6.42d), zwei XT-Mikrobefehle ergeben einen 20-Bit-Mikrobefehl mit dem Formatcode (FC) und 18 Inhaltsbits (Abb. 6.42e). Auf diese Weise stehen je drei Formate mit 12 und mit 18 Inhaltsbits zur Verfügung. In die Mikrobefehlsdecodierung fließen sowohl der Formatcode als auch die Mikrobefehlslänge ein. Die Mikrobefehlsausführung beginnt, wenn ein Byte mit einem der Formatcodes gemäß Abb. 6.42a geladen wurde. Nach der Ausführung wird das gesamte Mikrobefehlsregister gelöscht, so daß es wieder bereit ist, neue Erweiterungen aufzunehmen. Dieses Erweiterungsprinzip kann offensichtlich beliebig weit getrieben werden. Es wird aber unwirtschaftlich, wenn man es übertreibt, da jedes Erweiterungsbyte nur 6 Bits ergänzenden Inhalt mitbringt. Wenn man mit Formaten ähnlich Abb. 6.42c,d nicht auskommt, wählt man besser eine grundsätzlich andere Lösung mit breiteren Mikrobefehlen.
Abb. 6.43 zeigt eine nochmals erweiterte Einadreßmaschine mit zwei Arbeitsregistern A, B. Hier sind beide als Akkumulatoren ausgebildet (ACA, ACB). In beiden kann man das Ergebnis speichern, beide kann man aus dem Lokalspeicher laden. Registermodelle mit zwei oder mehr Akkumulatoren sind seit langem bekannt. Wie geht man mit solchen Akkumulatoren um, welche Mikrobefehlswirkungen sollte man implementieren? Anregungen dazu holt man sich am besten aus einschlägigen Anwendungsschriften und Maschinenhandbüchern[35].
Die Abbildung soll vor allem die Datenflüsse im Verarbeitungswerk veranschaulichen, die zu implementieren sind, wenn beide Register gleichermaßen vollwertige Akkumulatoren sein sollen. Die Auswahlschaltungen 1 dienen dazu, den Inhalt eines Akkumulators mit einem Direktwert aus dem Mikrobefehl zu verknüpfen, wobei der Inhalt des jeweils anderen Akkumulators in Ruhe gelassen wird. Die Auswahlschaltungen 2 dienen zum Zurückschreiben des Ergebnis-

35. Beispielsweise aus [109] und [110] (Motorola M6800) sowie [111] und [112] (Data General Nova).

ses und zum Laden aus dem Lokalspeicher. Wie aber lädt man einen Akkumulator mit einem Direktwert? Man könnte ihn über eine der Auswahlschaltungen 1 einfließen lassen und durch die ALU hindurchleiten (Gate-Thru-Operation). Sollte das aber zu lange dauern (Schaltungstiefe), müßte man die Auswahlschaltungen 2 entsprechend erweitern (3).

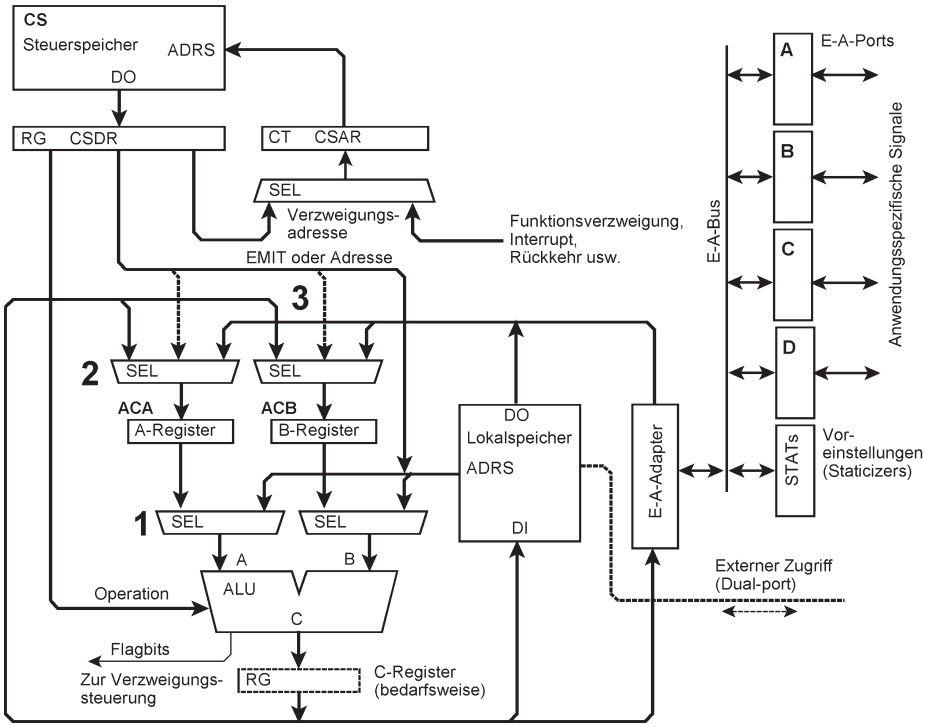


Abb. 6.43 Eine Einadreßmaschine mit zwei Akkumulatoren. 1, 2 - Auswahlschaltungen; 3 - zusätzliche Datenwege zum Laden von Direktwerten (bedarfsweise).

Kleine Stackmaschinen

Die in den Abb. 6.41 und 6.43 skizzierten Maschinen kann man mit nur wenigen Änderungen zu Maschinen umbauen, deren Arbeitsregister einen kleinen Registerstack bilden (Abb. 6.44). Die Z3 ist – mit ihrem Bytecode und der Zwangsreihenfolge des Ladens und Speicherns – im Grunde schon eine Art Stackmaschine mit einem Registerstack der Tiefe 2. Um sie zur richtigen Stackmaschine umzubauen, müßte man nur die Zustandssteuerung so abändern, daß die Auswahl zwischen den Registern A und B wie ein Stackpointer wirkt[36].

Es ist zweckmäßig, einen etwas tieferen Stack vorzusehen. Bereits ein Stack der Tiefe 3 erlaubt es, Schritt für Schritt ein Endergebnis zu bilden und dabei mit einfach geklammerten Ausdrük-

36. Vgl. [56].

ken zu arbeiten[37], also mit Schachtelungen der Art (A **op1** B) **op2** (C **op3** D) usw. Der Stack ist mit Vorwärts-Rückwärts-Schieberegistern aufgebaut. Die Register A und B gehören zum Stack. So stehen die ersten beiden Stack-Einträge parallel zur Verfügung. A = TOS, B = TOS+1. Die Mikrobefehlsformate können, wie bei der Z3, aus einem Bytecode entwickelt werden. Laden (LD) wird zu PUSH, Speichern (ST) zu POP. Das Verarbeitungsergebnis wird direkt ins Register A (= TOS) eingetragen. Wenn es aus zwei Stackeinträgen gebildet wurde, wird der Inhalt des dritten Stackregisters (T2 = TOS+2) ins Register B (TOS+1) geschoben.

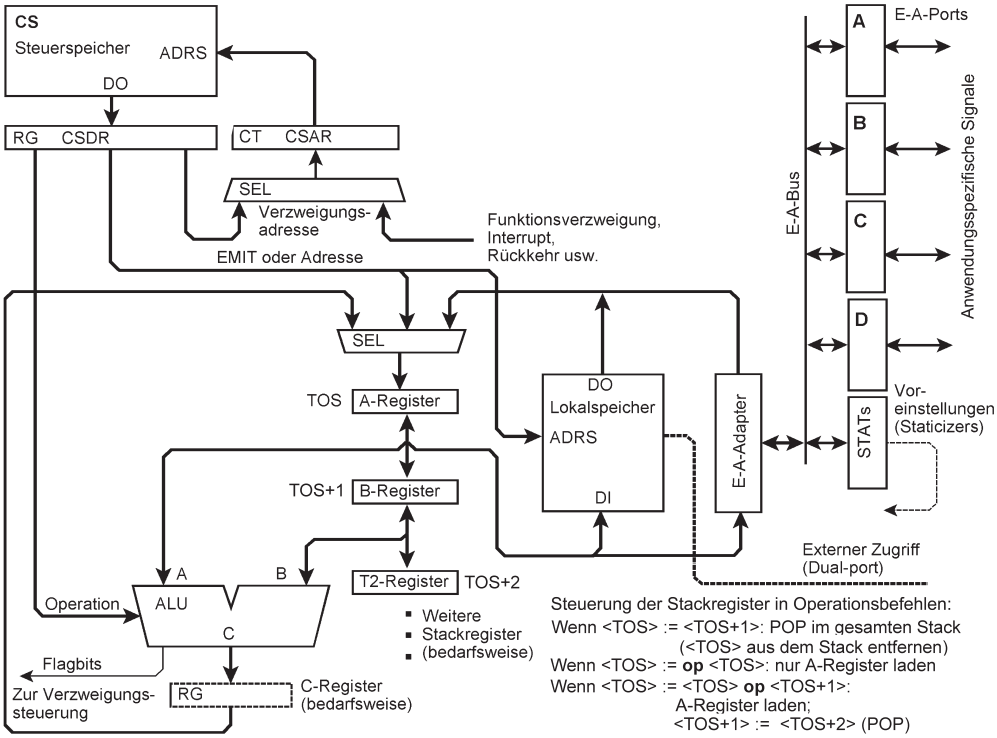


Abb. 6.44 Die zur Registerstackmaschine umgebaute Einadreßmaschine.

6.4 Universalmaschinen

Was den Maschinen des vorstehenden Abschnitts zur vollen Universalität noch fehlt, sind der universelle, dem Prinzip nach beliebig erweiterbare Datenspeicher[38] sowie Vorkehrungen zur

37. Hierfür gibt es historische Beispiele, nämlich die Transputer-Mikroprozessoren (Stacktiefe = 3) und der Taschenrechner HP 35 (Stacktiefe = 4). Als weiteres Beispiel einer Maschine mit einem kleinen Operandenstack mag die Gleitkommaverarbeitung (FPU) der x86-Prozessoren dienen. Die Stacktiefe ist hier = 8.
38. Als Annäherung an das Band der Turingmaschine.

Adreßrechnung. Wird der universelle Datenspeicher zusätzlich eingebaut oder wird der Lokalspeicher zum universellen Datenspeicher erweitert? Im ersten Fall ergibt sich eine Maschine mit einem Universalregistersatz, im zweiten eine herkömmliche Einadreßmaschine.

6.4.1 Einadreßmaschinen

Abb. 6.45 veranschaulicht, wie die Maschine von Abb. 6.35 zu einer Turing-vollständigen Einadreßmaschine weiterentwickelt werden kann. Der Lokalspeicher wird zum universellen Datenspeicher. Ein einziges Adreß- oder Indexregister (AD-Register) genügt, um die Adreßrechnung, das Lesen von Daten aus dem Programmspeicher und das indirekte Verzweigen zu unterstützen. Zum Retten von Rückkehradressen wurde ein Registerstack vorgesehen. Die Maschine ähnelt manchen Einadreßmaschinen der Entwicklungsgeschichte. Es gibt auch Mikrocontroller, die das gleiche Registermodell mit Akkumulator und Adreß- oder Indexregister aufweisen. In Abb. 6.46 wurde das Blockschaltbild in eine Darstellung umgezeichnet, wie sie in vielen Datenbüchern und Lehrwerken der Rechnerarchitektur üblich ist. Die zwei Speicher – einer für die Programme, einer für die Daten – ergeben eine Maschine in Harvard-Architektur. Legt man beide Speicher zusammen, ergibt sich eine v. Neumann-Maschine, deren Blockschaltbild noch einfacher und besser überschaubar ist (Abb. 6.47).

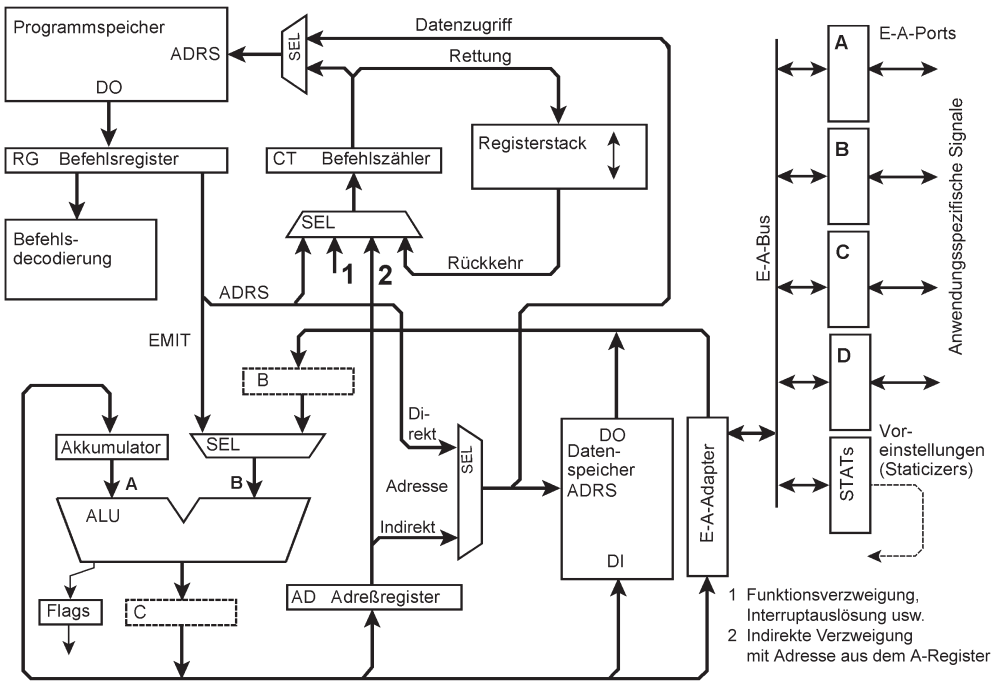


Abb. 6.45 Eine Turing-vollständige Einadreßmaschine in Harvard-Architektur, entwickelt aus einem Mikroprogrammsteuerwerk mit ALU und Lokalspeicher.

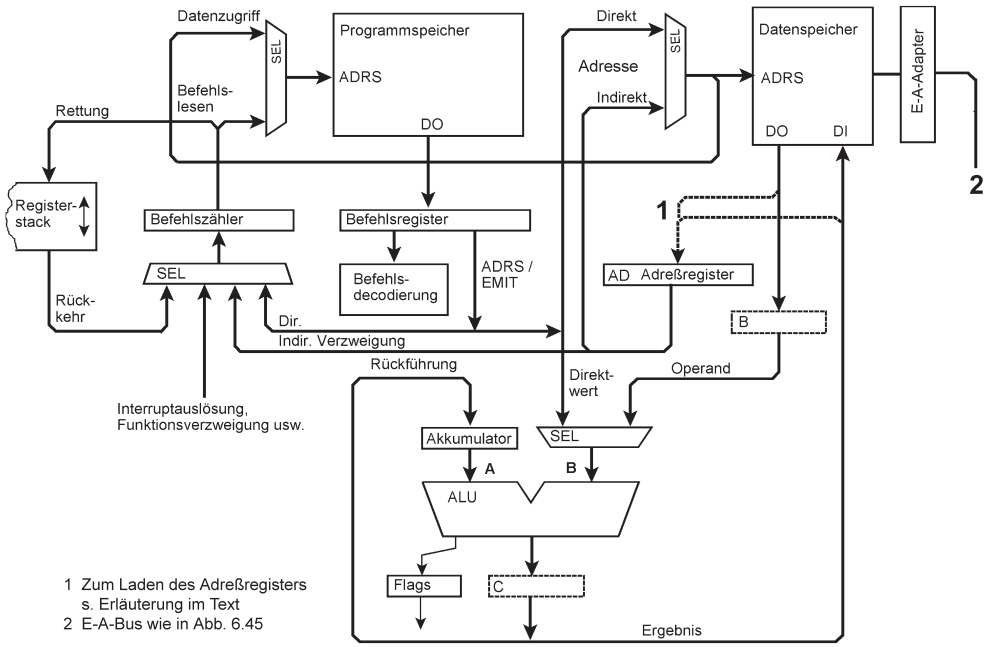


Abb. 6.46 Die Maschine von Abb. 6.45 in anderer Darstellung.

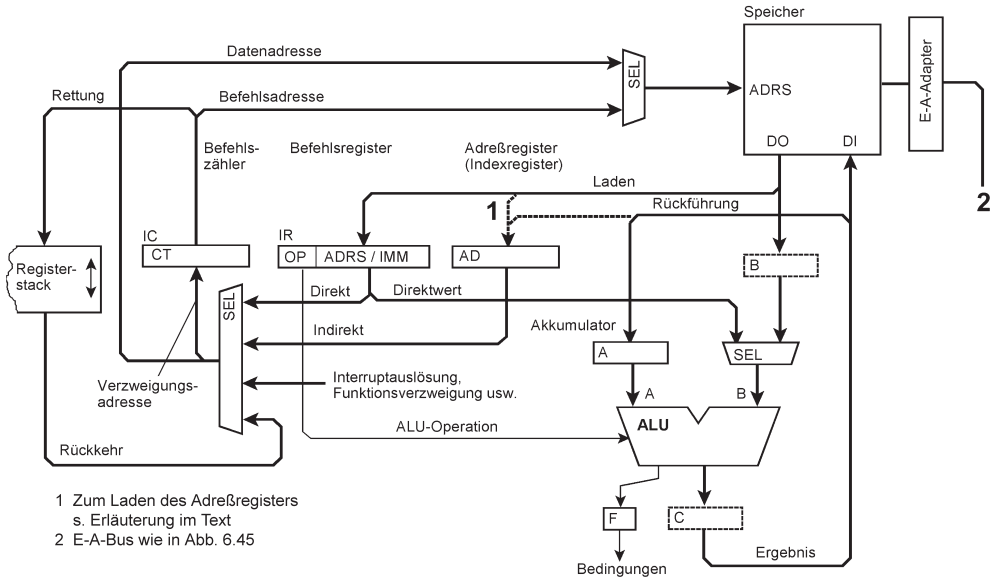


Abb. 6.47 Die Maschine in v. Neumann-Architektur.

Befehle und Mikrobefehle
Der Übergang zwischen vertikalen Mikrobefehlen und Einadreßbefehlen ist fließend. Wir wollen zunächst mit der üblichen Einadreßmaschine beginnen, wie sie auch der Architektur vieler Mikrocontroller zugrunde liegt. Deshalb reden wir zunächst von Befehlen im allgemeinen Sinne. Dann gehen wir zu den Mikrobefehlsformaten über. Der Mikrobefehl ist hier der großzügig dimensionierte und mit zusätzlichen Wirkungen angereicherte Einadreßbefehl.

Lokalspeicher und Datenspeicher
Der typische Lokalspeicher ist nicht allzu groß. Richtwert: 16 bis 256 Wörter der jeweiligen Verarbeitungsbreite. Hinzu kommen die Adressen in den E-A-Einrichtungen. Auch hier ist mit dieser Größenordnung zu rechnen (z. B. 4 Einrichtungen mit je 4 adressierbaren Registern ergeben 16 Wörter, 16 Einrichtungen mit je 16 Registern ergeben 256 Wörter). Der Lokalspeicher ist typischerweise ein SRAM im Prozessorschaltkreis, z. B. ein Block-RAM im FPGA. Die Speicherausstattung der besonders kleinen Mikrocontroller liegt auch in dieser Größenordnung. Datenspeicher sind oftmals größer. Richtwerte aus dem Bereich der kleinen Mikrocontroller: 16 kBytes bis 256 kBytes und mehr. Es kann sein, daß eine solche Speicherkapazität nicht mehr im gleichen Schaltkreis untergebracht werden kann oder daß im Speicheradreßraum unterschiedliche Speichereinrichtungen angeordnet sind (SRAM, Flash, EEPROM usw.). Wir nehmen hier an, daß der Speicher wie ein SRAM angesprochen wird und daß sich alle Adaptierungsprobleme lösen lassen[39].

Eine besonders einfache Maschine mit externem Datenspeicher
Der Steuerautomat von Abb. 6.35 wird mit der Programmspeicheradressierung gemäß Abb. 6.45 und einem außen angeschlossenen Datenspeicher erweitert. Eine besonders sparsame Lösung ergibt sich, wenn man eine Load-Store-Architektur implementiert und Operationen nur mit Lokalspeicherinhalten ausführt. Dann kann man das A-Register auch als universelles Adreßregister verwenden (Abb. 6.48 und 6.49). Um den Arbeitsspeicher zu nutzen, braucht man nur zwei Befehle, Laden und Speichern (Abb. 6.49a):

- Laden: <LS-Adresse> := Arbeitsspeicherinhalt gemäß Adresse im A-Register (Lesezugriff),
- Speichern: Arbeitsspeicherinhalt gemäß Adresse im A-Register := <LS-Adresse> (Schreibzugriff).

Es liegt nahe, zusätzliche Befehlswirkungen vorzusehen. Abb. 6.49b zeigt ein Befehlsfomat, das einen Direktwert enthält, der zum Inhalt des A-Registers addiert wird (vgl. den Signalweg 3 in Abb. 6.48). So kann man zwei oder mehr Lokalspeicherzellen nacheinander laden oder speichern, ohne das A-Register jedesmal neu laden zu müssen. Wenn im Befehlsformat noch Platz ist, kann man die Speicheradresse verlängern (Abb. 6.49c; vgl. auch den Signalweg 4 in Abb. 6.48).

39. Das Einfügen von Wartezuständen, das Umladen von Speicherinhalten vom Flash oder EEPROM in den SRAM, das Einbauen von Caches usw. gehören zum Stand der Technik.

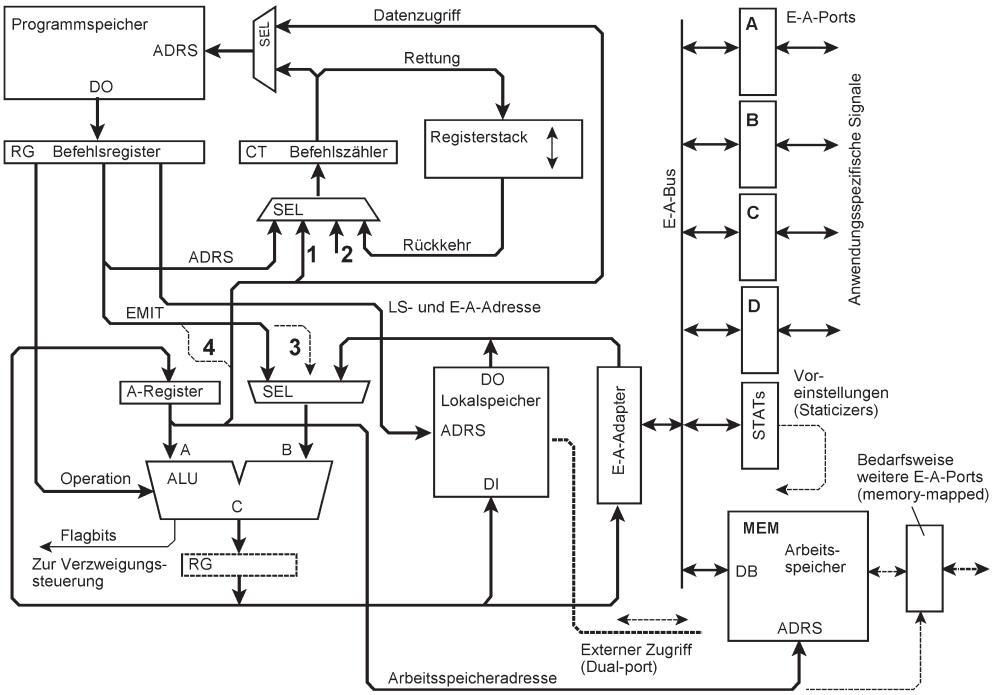


Abb. 6.48 Eine einfache Maschine mit externem Datenspeicher. Sie kommt ohne ein besonderes Adreßregister aus ([MD27]). 1 - indirekte Verzweigung mit Adresse aus dem A-Register; 2 - Funktionsverzweigung, Interruptauslösung usw.; 3 - Direktwert zur Adreßerhöhung; 4 - Adreßverlängerung. Ob die Signalwege 3 und 4 vorgesehen werden oder nicht, hängt vom Befehlsformat ab.

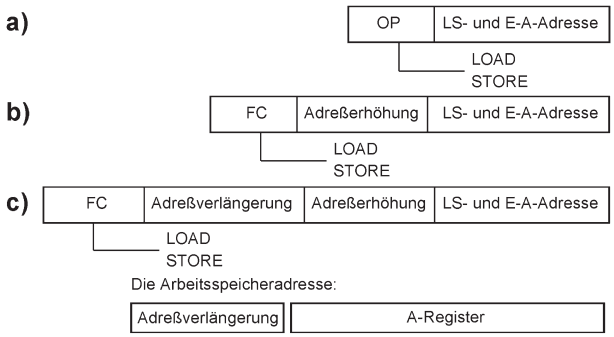


Abb. 6.49 Befehlsformate der Arbeitsspeicherzugriffe. a) elementare Lade- und Speicherbefehle; b) Mikrobefehlsformat mit zusätzlichem Direktwert, der nach dem Speicherzugriff zur Adresse im A-Register addiert wird (Post-Increment); c) Mikrobefehlsformat mit zusätzlichem Direktwert zur Adreßverlängerung.

Mit der Adreßverlängerung könnte man beispielsweise zwischen Speicher- und E-A-Zugriffen unterscheiden, womöglich sogar jeder Anwendungsfunktion (Prozeß, Partition, Task o. dergl.) einen eigenen Speicherbereich zuweisen. Die Auslegung gemäß Abb. 6.48 bietet sich vor allem für ausgesprochen kleine Maschinen mit kurzen vertikalen Mikrobefehlen an.

Mikrobefehlsformate für Einadreßmaschinen
Die nachfolgenden Erläuterungen beziehen sich vorzugsweise auf Einadreßmaschinen mit wahlweiser direkter und indirekter Speicheradressierung, wie sie in den Abb. 6.45 bis 6.47 dargestellt sind. Die Speicheradresse ist ein Direktwert im Mikrobefehl oder sie kommt aus dem Adreßregister (AD-Register). Die Sparlösung gemäß Abb. 6.48 und 6.49 ist ein Sonderfall. Die meisten Erläuterungen gelten aber auch für diese Ausführung. Unterschiede gibt es nur bei den Speicherzugriffsmikrobefehlen. Ohne AD-Register entfallen auch die Mikrobefehle und Anweisungen, die dieses Register betreffen.

Operanden- und Ergebnisregister
Ob man welche braucht, hängt von der Speichertechnologie ab, von den Zykluszeiten, den Signallaufzeiten im Schaltkreis und vom Taktsystem[40]. Ein Operandenregister (Register B) ist auch dann erforderlich, wenn man das Holen des Operanden und die Operationsausführung mit getrennten Mikrobefehlen steuern möchte (Load-Store-Architektur). Ob man zudem ein Ergebnisregister (Register C) braucht oder das Ergebnis vom Ausgang der ALU dem Speicher direkt zuführt, ist eine Frage des Feinentwurfs. Das Operandenregister B und das Ergebnisregister C sind dementsprechend nach Bedarf einzufügen.

Direkte Adressierung
Die Adresse ist ein Direktwert im Befehl oder Mikrobefehl.

Indirekte Adressierung
Die Adresse ist Ergebnis einer Adreßrechnung oder sie wird von außen eingespeist. In typischen Universalprozessoren[41] können auch in den Speicherzugriffsbefehlen Adreßrechenvorgänge stattfinden, beispielsweise nach dem Schema Basis + Displacement oder Basis + Index • Skalierung + Displacement. Mikrobefehle hingegen sollen nicht länger dauern als ein einzelner Maschinenzyklus. Deshalb können sie nur mit fertig berechneten Adressen zum Speicher zugreifen. Somit ist die Adreßrechnung auszuprogrammieren. Die Adreßwerte, die sich dabei ergeben, werden gespeichert und bei Bedarf ins Adreßregister geladen. In Universalprozessoren hat man hierfür zwei Prinzipien implementiert: das automatische und das programmierte Laden. Das automatische Laden erfordert wenigstens einen zusätzlichen Maschinenzyklus im Befehl. Die Ablaufsteuerung ist aber so einfach, daß das Prinzip auch für vertikale Mikrobefehle in Betracht kommen könnte. Deshalb sollen beide Prinzipien kurz erläutert werden (Abb. 6.50).

40. Man braucht womöglich dann keine, wenn die Datenausgänge des Speichers auf seine Dateneingänge zurückgeführt werden dürfen (betrifft z. B. Block-RAMs in der Betriebsart Read-before-Write).
41. Mit Ausnahme mancher historischer Maschinen und einfacher Mikrocontroller.

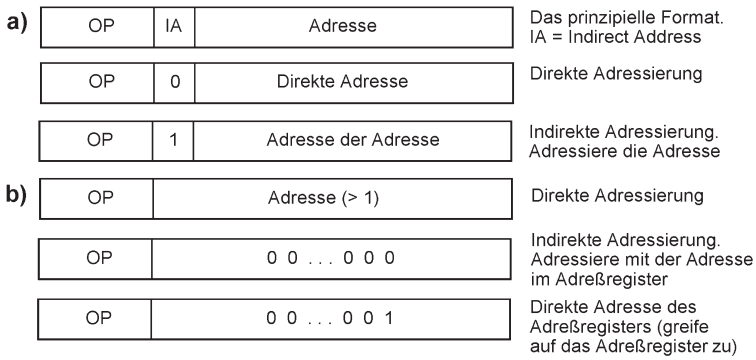


Abb. 6.50 Prinzip der indirekten Adressierung mittels Adreßregister. a) Das Adreßregister wird automatisch geladen und genutzt; b) das Adreßregister wird vom Programm geladen und genutzt.

a) Automatisch (Adresse von Adresse). Das Laden und Verwenden des Adreßregisters ist Sache der Befehlsablaufsteuerung. Ist es ein Zugriff mit indirekter Adressierung (IA = 1), so adressiert der Befehl ein Maschinenwort im Speicher, das als Adreßzeiger dient, und lädt es in das Adreßregister. Dann wird der eigentliche Zugriff ausgeführt. Dabei kommt die Speicheradresse aus dem Adreßregister. Die Adreßzeiger im Speicher entsprechen den Zeigervariablen in üblichen Programmen. Sie werden vor den Zugriffen vorbereitet (programmierte Adreßrechnung) und im Speicher abgelegt. Das Adreßregister ist programmseitig gar nicht zugänglich; es wirkt nur als Halteregister bzw. Puffer.

b) Vom Programm. Die Befehle weisen das Laden des Adreßregisters an und bestimmen, ob es genutzt wird oder nicht. Möchte man indirekt adressieren, so muß man die Adresse zunächst aufbereiten (z. B. berechnen) und ins Adreßregister laden. Im Beispiel von Abb. 6.44b dient hierzu ein Befehl mit der Adresse 1. Befehle mit Adresse 0 verwenden das Adreßregister, wenn sie auf den Speicher zugreifen. Alle Adressen > 1 sind direkte Adressen.

Die automatische Nutzung des Adreßregisters hat den Vorteil, daß man keine besonderen Befehle braucht, um Adressen ins Adreßregister zu laden. Sie hat aber auch Nachteile:

- Um zwischen direkter und indirekter Adressierung zu unterscheiden, braucht man ein Auswahlbit im Adreßteil des Befehls. Hierdurch halbiert sich das Adressierungsvermögen. So könnte man beispielsweise mit 12 Adreßbits 4096 Speicherwörter adressieren, mit dem Format gemäß Abb. 6.50a hingegen nur 2048.
- Sie verlängert die Befehlsausführung. Man braucht wenigstens zwei Maschinenzyklen, einen zum Holen der Adresse und einen für den eigentlichen Speicherzugriff.

Wird das Adreßregister hingegen programmseitig geladen, so braucht man nur zwei Adressen, eine zum Zugreifen mit dem Adreßregister (Adresse 0 in Abb. 6.50b) und eine zum Zugreifen auf das Adreßregister (Adresse 1 in Abb. 6.50b). Das Adressierungsvermögen der Befehle vermindert sich somit nur um zwei Speicherwörter (beispielsweise von 4096 auf 4094 Wörter).

Das Adreßregister laden

Zum Laden des Adreßregisters braucht man entsprechende Befehle oder Mikroanweisungen. Hier geht es aber zunächst nur um die Zugangswege. In den Abb. 6.45 bis 6.47 sind zwei Lösungen dargestellt bzw. angedeutet:

- Das Laden mit einem Speicherinhalt. Das Adreßregister ist an den an den Datenausgang des Speichers angeschlossen (vgl. Abb. 6.46 und 6.47). Auf diesem Wege kann man aber nur Adressen laden, die im Speicher abgelegt sind.
- Das Laden mit einem Rechenergebnis (vgl. Abb. 6.45 bis 6.47) Das Adreßregister ist an den Ausgang der Verarbeitungseinheit (ALU) angeschlossen. Auf diesem Wege kann man beispielsweise eine effektive Adresse berechnen, indem ein Displacement im Akkumulator zu einer Basisadresse im Speicher addiert wird. Wenn man das Adreßregister ohne zu rechnen mit einem Speicherinhalt laden will, muß die ALU auf das Durchreichen des B-Eingangs eingestellt werden (Gate-Thru B).

In der Schaltung von Abb. 6.51 ist das Adreßregister AD so mit der ALU verbunden, so daß man seinen Inhalt in die Rechengänge einbeziehen kann. Dabei werden folgende Rechengänge unterstützt: <AD> **op** Direktwert und <AD> **op** <Speicher>.

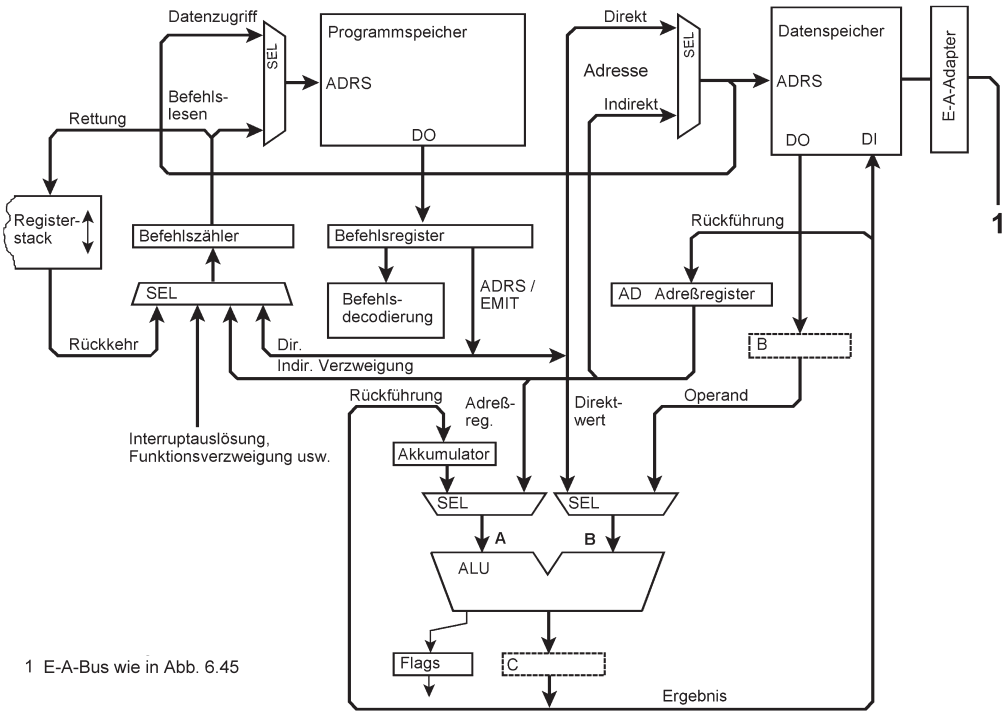


Abb. 6.51 Hier ist das Adreßregister so in die Datenwege der ALU einbezogen, daß man mit seinem Inhalt rechnen kann.

Mikrobefehlsformate
Unsere Maschine soll mehr leisten als die üblichen kleinen Mikrocontroller. Wenn wir von Grund auf entwickeln, können wir etwas längere Mikrobefehlsformate wählen, in denen wir zusätzliche Funktionen der Ein- und Ausgabe und anwendungsspezifische Wirkungen codieren. Unsere Maschinen sollen sich für die jeweils gewünschte Verarbeitungsbreite synthetisieren lassen und eine innige Verquickung des Prozessorkerns mit den Schaltungen der Ein- und Ausgabe aufweisen. Dazu gehört, Bedingungen, Auswahlsignale und Mikrobefehlsadressen gemäß den anwendungsspezifischen Anforderungen zu bilden, erforderlichenfalls auch mit Spezialschaltungen, die mittels Verhaltensbeschreibung erfaßt werden.

Voraussetzungen und Annahmen
Wir wollen die Mikrobefehlsgestaltung hier, am Beispiel der einfachen Einadreßmaschine, etwas ausführlicher diskutieren. Die Entwicklung beginnt mit den grundsätzlichen Zielvorgaben. Wir wollen einfach bauen, aber großzügig dimensionieren. Die grundlegenden Parameter sind die Verarbeitungsbreite und die Adreßlänge. Unsere Vorgabe: Verarbeitungsbreite ≥ Adreßlänge. Die Verarbeitungsbreite soll die Adreßrechnung über den gesamten Adreßraum unterstützen; es soll nicht notwendig sein, Abläufe der Adreßrechnung abschnittsweise auszuführen (wie u. a. in den typischen 8-Bit-Mikrocontrollern). Da wir uns hier auf kleine Maschinen beschränken wollen, nehmen wir zwei Beispiele an:

a) 12 Bits Verarbeitungsbreite, 4k Adreßraum. Es soll ein Gegenstück zu den kleinen 8-Bit-Mikrocontrollern werden. 12 Bits sind ein zweckmäßiges, seit langem bewährtes Format (vgl. PDP-8). U. a. kann man, wenn man mit Datenbytes arbeitet, den 8 Datenbits bis zu 4 Begleitsignale, Auswahladressen oder Funktionscodes beigeben.
b) 16 Bits Verarbeitungsbreite, 64k Adreßraum. Das ergibt ein Gegenstück zu den leistungsfähigeren 8-Bit-Mikrocontrollern und zu manchen 16-Bit-Mikrocontrollern.

Um bei den Mikrobefehlsformaten nicht von vornherein eingeschränkt zu sein[42], bauen wir eine Harvard-Maschine. Beide Adreßräume sollen gleich groß sein (4k oder 64k).

Die Nutzung des Adreßregisters
Das programmseitig ladbare Adreßregister (vgl. Abb. 6.50b) ist die zweckmäßigere Lösung[43]. Auch ist es von Vorteil, wenn man mit dem Adreßregisterinhalt rechnen kann. Deshalb wollen wir uns diesen zusätzlichen Aufwand[44] leisten und das Adreßregister gemäß Abb. 6.51 in die Datenwege der ALU einbeziehen.

42. Vor allem in Hinsicht auf die Wortlänge bzw. Zugriffsbreite. Wir möchten so breit bauen dürfen, wie wir es für zweckmäßig ansehen.
43. Das Adreßfeld im Mikrobefehl kann eine Adresse in voller Länge aufnehmen, und wir brauchen keinen Sequencer, um die indirekte Adressierung zu steuern.
44. Sollte der Aufwand nicht tragbar sein, wird dieses Architekturmerkmal weggelassen. Das Blockschaltbild der Maschine entspricht dann Abb. 6.46.

Operationscodes
Wir nehmen an, daß es vollauf ausreicht, 256 verschiedene Operationen codieren zu können.

Adressen und Direktwerte
Die Speicheradresse, die Verzweigungsadresse und der Direktwert werden jeweils in voller Länge im Mikrobefehl untergebracht (12 oder 16 Bits).

Wo das Emit-Feld einfließt
Wenn man schon einen solchen Aufwand treibt (es geht um bis zu 80 Mikrobefehlsbits), sollte das Emit-Feld auch auf beiden Seiten der ALU wirksam werden können. Dazu ist die Schaltung so zu ändern wie in Abb. 6.52 gezeigt. Dann kann man den Inhalt einer Lokalspeicherzelle oder eines E-A-Registers mit einem Direktwert vergleichen oder verknüpfen und gemäß den Bedingungen verzweigen, die sich dabei ergeben haben[45].

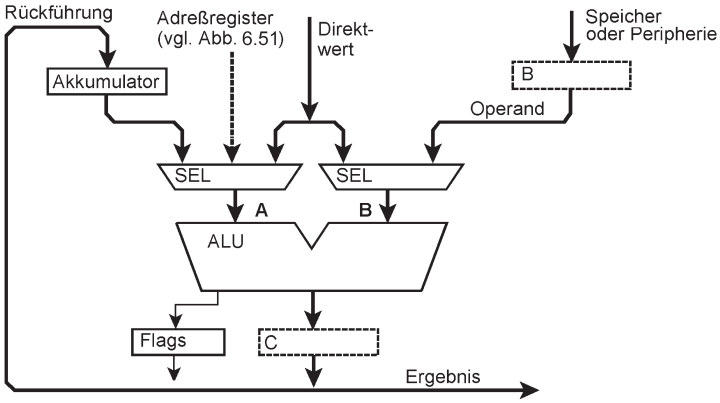


Abb. 6.52 Diese Abwandlung (für das horizontale Mikrobefehlsformat) erlaubt es, auch Speicherinhalte oder Daten aus der Peripherie mit dem Direktwert zu verknüpfen.

Unterbrechungen und Unterprogramme
Solche Vorkehrungen werden nach Bedarf implementiert. Zu Einzelheiten sei auf Kapitel 5 verwiesen (S. 289ff). Die Anweisungen und Steuerbits kann man in den Feldern der Verzweigungssteuerung und in Voreinstellregistern unterbringen.

Horizontale Mikrobefehle
Wir betrachten die Maschine von Abb. 6.51 (bzw. Abb. 6.46) als Ressourcenvektormaschine und stellen erst einmal alles zusammen, was gebraucht wird, um die Schaltung zu steuern und mit Parametern zu versorgen. Das grundsätzliche Mikrobefehlsformat entspricht Abb. 6.36. Abb. 6.53 zeigt die Formate, die sich mit unseren Annahmen ergeben.

45. Das erfordert einen entsprechend langen Mikrobefehlszyklus mit wenigstens zwei Taktphasen oder die Implementierung der Spätverzweigung.

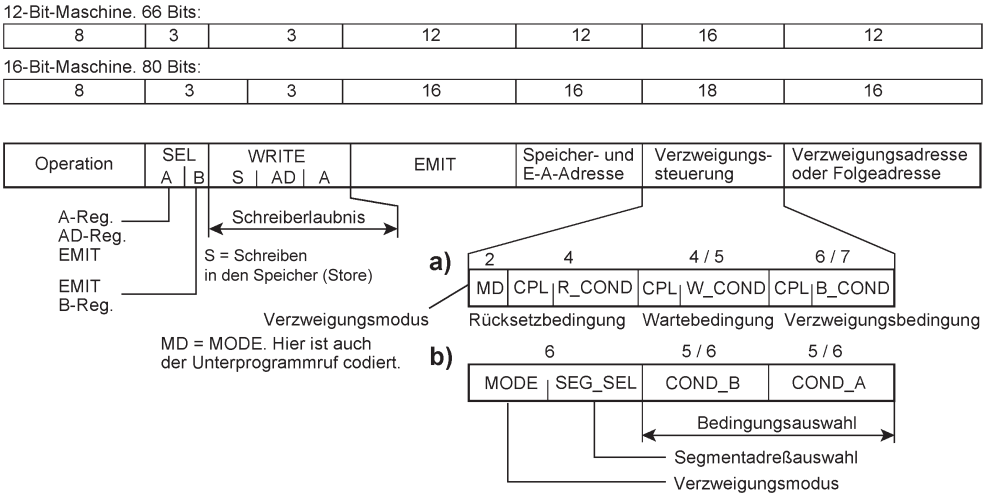


Abb. 6.53 Horizontale Mikrobefehlsformate für die Einadreßmaschine.

Verzweigungen

Was aufträgt, ist die Bedingungsabfrage. Beim bedingten Verzweigen sollten wir nicht um jeden Preis sparen, denn hier kann sich das Mikroprogrammsteuerwerk dem üblichen Universalprozessor als grundsätzlich überlegen erweisen. Wenn man es entsprechend auslegt, kann es alle Bedingungen, die überhaupt in Betracht kommen, aus den Anwendungsschaltungen abfragen und in mehr als eine Richtung verzweigen. Zum Verzweigen wollen wir in der kleineren Maschine bis zu 32 Bedingungen abfragen, in der größeren bis zu 64. Zum Rücksetzen und Warten sollten 8 oder 16 Bedingungen genügen. Die Anzahl der abfragbaren Bedingungen läßt sich durch Voreinstellung beliebig erweitern. Unterprogrammruf und Rückkehr (auch unter Bedingung) werden im Modussteuerfeld codiert. Die Verzweigungssteuerung kostet so 16 oder 18 Bits. In Abb. 6.53 sind zwei Varianten dargestellt:

a) Verzweigen nach dem Prinzip des universellen Steuerautomaten. Adreßzählung, wenn nicht verzweigt wird.
b) Mehrwegeverzweigung (Multiway Branching) durch Einspeisen von Bedingungssignalen. Hier wird mit zwei Bedingungsbits in vier Richtungen verzweigt (4-way).

Vertikale Mikrobefehle

Bis zu 80 Mikrobefehlsbits sind – für eine so kleine Maschine – schon recht viel. Also liegt es nahe, zu kürzen. Der Grundgedanke: die Felder, die am meisten auftragen, auf unterschiedliche Mikrobefehle verteilen. Das Emit- und das Adreßfeld passen nicht beide zusammen in ein Format. Demzufolge ist es nicht möglich, Direktwerte mit Lokalspeicherinhalten zu verknüpfen. Die Änderung von Abb. 6.52 wird hier nicht eingeführt. Es ergeben sich zunächst vier Grundformate:

1) Operationen mit dem Emit-Feld und dem A-Register,
2) Operationen mit dem Speicher und dem A-Register,
3) Verzweigungen,
4) allgemeine Steuerwirkungen (Voreinstellungen, Merkbits usw.). Im horizontalen Mikrobefehl hätte man hierzu das Emit-Feld ausgenutzt.

Abb. 6.54 zeigt, wie die Formate aussehen, wenn man das horizontale Format von Abb. 6.49 entsprechend zerlegt. Dabei wurde auch die Verzweigung auf ein einfaches Verzweigen umgestellt (vgl. die typischen Branch- oder Jump-on-Condition-Befehle der Mikrocontroller).

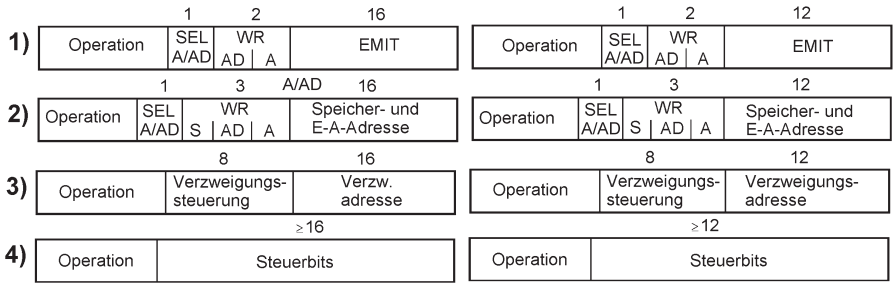


Abb. 6.54 Grundformate vertikaler Mikrobefehle. Links für die 16-Bit-Maschine, rechts für die 12-Bit-Maschine.

1) Der Inhalt des A- oder AD-Registers wird mit dem Direktwert verknüpft. Die Schreiberlaubnisbits des horizontalen Formats wurden beibehalten. Das Schreiben in den Speicher ist nur mit indirekter Adressierung möglich (Adresse im AD-Register).
2) Operationen mit den Inhalt des A- oder AD-Registers und der adressierten Speicherzelle. Es gibt zwei Varianten: (1) Verknüpfungen der Art <A> **op** <Speicher/E-A> (Mikrobefehlstyp ALF), (2) Operationen mit dem Inhalt der Speicherzelle allein (Bits setzen, löschen, abfragen usw.; Mikrobefehlstyp MODIFY). Die Schreiberlaubnisbits des horizontalen Formats wurden beibehalten.
3) Verzweigungen einschließlich Unterprogrammruf.
4) Beliebige Steuerwirkungen. In diesem Grundformat können auch Wartemikrobefehle codiert werden.

Zur Bedingungsauswahl und Verzweigungssteuerung sehen wir ein 8-Bit-Feld als ausreichend an. Wie die Auswahlschaltung vor dem B-Eingang der ALU angesteuert wird, ergibt sich aus dem Operationscode[46]. Vertikale Mikrobefehle können – dem Prinzip nach – so ausgelegt werden wie in Abb. 6.40 gezeigt. Einzelheiten können hier nur anhand weniger Beispiele kurz angesprochen werden. Wir wollen uns dabei auf drei Mikrobefehlslängen beschränken, die Industriestandards sind, nämlich auf 32, 24 und 16 Bits.

46. Je nach Operationscode ist entweder das Emit-Feld oder das B-Register (= Speicherausgang) auszuwählen.

32 Bits

In diesem Format kann man die Steuerbits, Codes, Adressen und Festwerte der 16-Bit-Maschine ohne weiteres unterbringen (Abb. 6.55). Es weist offensichtlich genügend Reserven auf, so daß man alles codieren kann, ohne zu tricksen.

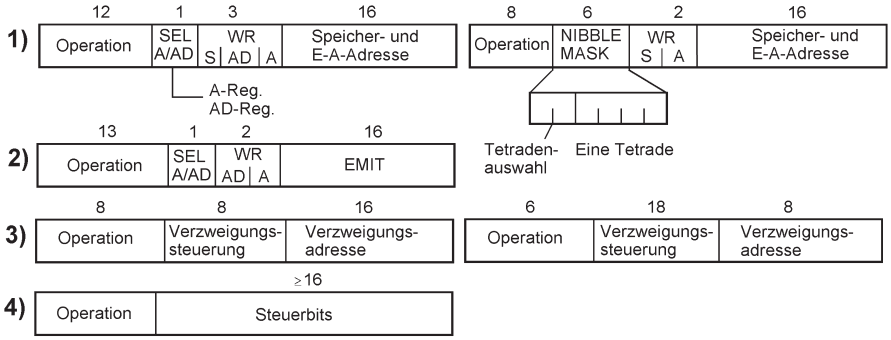


Abb. 6.55 Das prinzipielle 32-Bit-Format.

1) Links die Verknüpfungen mit dem Inhalt des A- oder AD-Registers (Mikrobefehlstyp ALF), rechts die Operationen mit dem Inhalt der Speicherzelle (Bits setzen, löschen, abfragen usw.; Mikrobefehlstyp MODIFY). MODIFY-Mikrobefehle können nicht ins AD-Register schreiben (zur Adreßrechnung braucht man kaum Bitbefehle, deshalb ergeben sich aus dieser Sparlösung praktisch keine Nachteile).
2) Operationen mit den Inhalt des A- oder AD-Registers und dem Direktwert.
3) Verzweigungen einschließlich Unterprogrammruf. Links als übliche bedingte Verzweigung, rechts als Funktionsverzweigung in vier Richtungen (4-way).
4) Beliebige Steuerwirkungen auslösen. In diesem Grundformat können auch Wartemikrobefehle codiert werden.

Abb. 6.56 zeigt 32-Bit-Formate in Anlehnung an Abb. 6.40. Da genügend Operationscodebits verfügbar sind, können wir uns einen 3-Bit-Funktionscode (FC) leisten.

a) EMIT OPs. Im Operationscodefeld kann man eine Vielzahl zusätzlicher anwendungsspezifischer Funktionen codieren.
b) ALF. Auch dieses Operationscodefeld ist breit genug, um Funktionen zu codieren, die über die übliche Ausstattung der kleineren Mikrocontroller hinausgehen.
c) MODIFY. Da der Direktwert und die Speicheradresse nicht in ein Format passen, gibt es Mikrobefehle, um den Lokalspeicherinhalt selektiv zu beeinflussen. Der Mikrobefehl enthält eine Tetrade (Nibble) als Direktwert und ein Auswahlfeld für alle vier Tetraden des 16-Bit-Worts (NIBSEL = Nibble Select). Es können bis zu 32 Operationen codiert werden (Bits setzen, löschen, testen, Tetrade einfügen, Speicherzelle löschen, Inhalt der Speicherzelle verschieben usw.).

d) JUMP, CALL. Hier sind die üblichen bedingten Verzweigungen und der (ebenfalls bedingte) Unterprogrammruf codiert.
e) BRANCH MW. Die Mehrwegeverzweigung (MW = Multiway) beruht auf der Segmentierung der Mikrobefehlsadresse. Die 8-Bit-Adresse aus dem Mikrobefehl wird mit zwei Bits aus der Bedingungsauswahl ergänzt. Somit Verzweigen in 4 Richtungen (4-way). Die Verzweigungsadresse hat insgesamt 10 Bits, so daß ein Segment bis zu 1k Mikrobefehle enthalten kann. Zu den 18 Bits der Verzweigungssteuerung vgl. Abb. 6.53b.
f) WCTL. Das Warten auf das Eintreffen einer Wartebedingung ist gelegentlich eine zweckmäßige Alternative zur Warteschleife. Das Format enthält eine Speicheradresse, um das Warten mit der Datenübertragung kombinieren zu können[47].
g) CTL. In diesem Format können beliebige Steuerbits und -anweisungen untergebracht werden, aber auch Direktwerte und Adressen. Es liegt nahe, unterschiedliche Sub-Formate vorzusehen (Abb. 6.57).
h) Das 32-Bit-Format hat genügend Reserven, um in jedem Mikrobefehl ein Bit zum Unterdrücken von Unterbrechungen vorzusehen (DB = Disable Break-In). Ist DB gesetzt, wird der nachfolgende Mikrobefehl stets ausgeführt, ohne daß ein Break-In dazwischenkommen kann (nicht-unterbrechbarer Mikrobefehlsablauf).

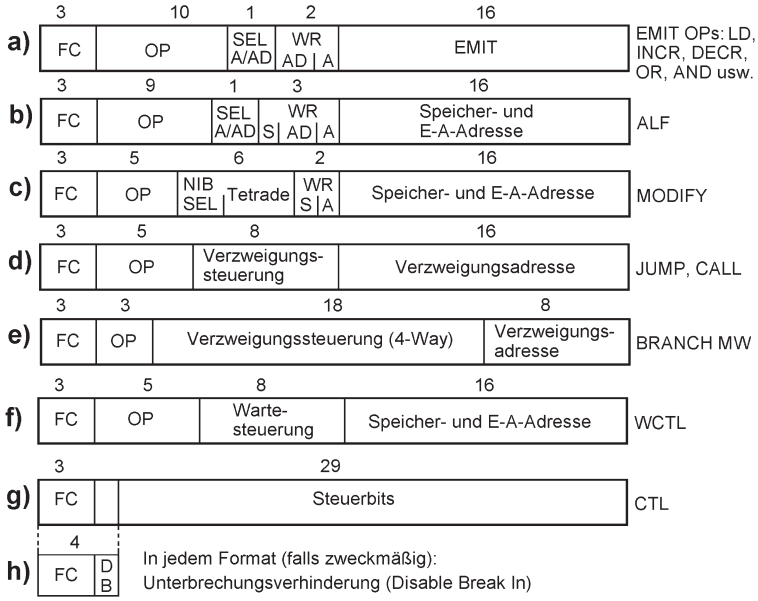


Abb. 6.56 Beispiele von 32-Bit-Formaten für die 16-Bit-Maschine. In der Praxis wird man nur die Formate implementieren, die man braucht.

47. Z. B. auf ein Interfacesignal warten und dann das gelieferte Datenbyte speichern.

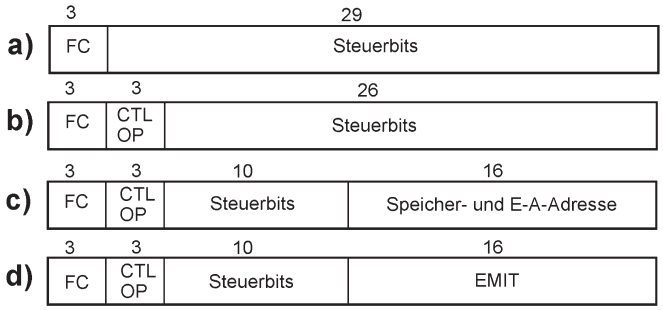


Abb. 6.57 Sub-Formate des CTL-Mikrobefehls. a) Grundformat; b), c), d) Beispiele von Sub-Formaten mit Steuerbits, einer Adresse und einem Direktwert. Codierung der Sub-Formate mit drei Bits CTL OP, einer Art Operationscode.

24 Bits

Die Entwicklungsgeschichte kennt viele Maschinen mit 24-Bit-Befehlen. Hier soll es aber darum gehen, Mikrobefehle auf einfache Weise zu codieren. Also stellt sich die Frage, was sozusagen freiwillig hineinpaßt, soll heißen ohne Voreinstellungen, Bankregister und ähnliche Behelfslösungen. Mit 16 Direktwert- und Adreßbits wird es dann schon sehr knapp. Mit 12 Bits hingegen ist es möglich, die meisten der Formate von Abb. 6.56 beizubehalten, allerdings mit Einschränkungen und geringeren Codierungsreserven (Abb. 6.58).

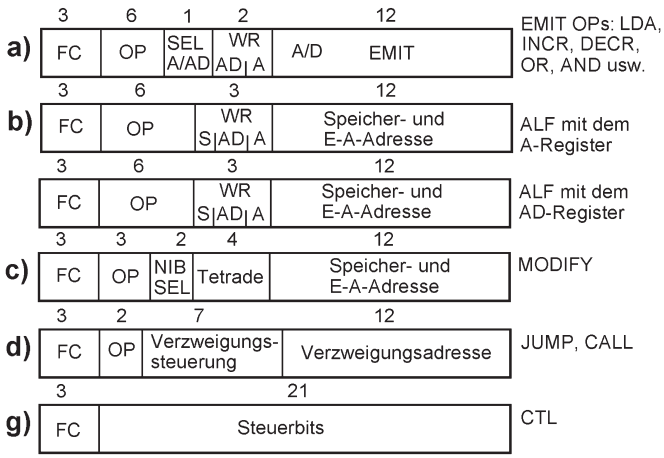


Abb. 6.58 Beispiele von 24-Bit-Formaten für die 12-Bit-Maschine.

Die Formate a), b), c), d) und g) entsprechen weitgehend denen von Abb. 6.56, manche Felder sind aber kürzer. Um im ALF-Format b) genügend Operationscodebits verfügbar zu haben, wird die Registerauswahl vor dem A-Eingang der ALU im FC-Feld codiert; es gibt also zwei Funktionscodes; der eine wählt das A-Register aus, der andere das AD-Register.

Die Mehrwegeverzweigung und der Wartemikrobefehl wurden weggelassen, weil die Anweisungsfelder nicht in dieses Format passen[48]. Die Operationsvielfalt des MODIFY-Mikrobefehls ist eingeschränkt; das Schreiben ins A-Register und in den Speicher kann nicht unabhängig von der Operation gesteuert werden. Es muß also Operationen mit und ohne Schreiben geben (letzteres z. B. um Verzweigungsbedingungen zu gewinnen).

16 Bits

Es ist ein typisches Befehlsformat vieler Minicomputer und Mikroprozessoren. Will man aber das Schema unserer vertikalen Mikrobefehle – nach dem Prinzip von Abb. 6.50 – in diesem Format implementieren, wird es knapp. Wenn die Adressen und Direktwerte 12 Bits lang sind, verbleiben nur noch vier Bits für Formatcodes, Operationscodes und Auswahlfelder. Damit ist es nicht möglich, Formate darzustellen, die denen der Abb. 6.52 und 6.54 ähnlich sind. Nun könnte man grundsätzlich von diesem Schema abweichen. So kommt eine Stackmaschine mit kurzen Befehlen aus. Um die vorstehend erläuterten Mikrobefehlsformate wenigstens dem Prinzip nach beizubehalten, verbleibt die Verkürzung der Speicheradresse. Wir wollen hier – als Beispiel – 6 Bits wählen, womit 64 Wörter adressiert werden können (Abb. 6.59). Damit kehren wir zum Lokalspeicher zurück. Größere Speicherkapazitäten sind dann mit Speichern zu implementieren, die wie periphere Einrichtungen angeschlossen werden. Die Maschine wird zur Einadreßmaschine mit Universalregistersatz. Eine besonders sparsame Ausführung haben wir bereits in Abb. 6.48 vorgestellt. Das Adreßregister (AD) entfällt; das A-Register dient auch als Adreßregister.

Speicherzugriffe

Wir implementieren die Funktionen gemäß Abb. 6.49c (S. 364). Mikrobefehle für Speicherzugriffe haben das Grundformat CTL (Abb. 6.60; S. 377). Der Funktionscode wird um ein Bit verlängert. Ist dieses Bit = 0, so ist es ein Speicherzugriff. Das nächste Bit gibt an, ob geladen oder gespeichert wird. Bei einer Lokalspeicheradresse von 6 Bits verbleiben noch 5 Bits. Im Beispiel von Abb. 6.60 werden zwei zur Adreßverlängerung genutzt (AX = Address eXtension). Die restlichen drei sind ein Direktwert, der umcodiert, auf die Verarbeitungsbreite erweitert und zum Inhalt des A-Registers addiert wird (INCR = Increment).

Beispiel einer Adreßaufteilung (AX)

Es gibt vier Bereiche: Gemeinsam (Common) – System – Anwendung (User) – E-A (I/O). Daß man die Adresse weiter verlängern kann, indem man mit den zwei Bits eines von vier Segment- oder Bankregistern auswählt, versteht sich von selbst.

Praxisbrauchbare Werte der Adreßerhöhung (INCR)

0 (nichts verändern), +1,+2, +4, –1, –2, –4. Rechengang:

$$\langle A\rangle := \langle A\rangle + \mathrm{INCR}$$

48. Wenn man so etwas implementieren will, braucht man Voreinstellregister, die die Parameter aufnehmen.

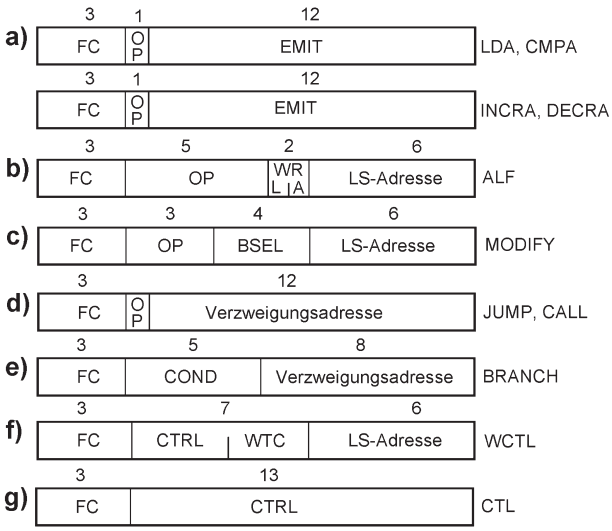


Abb. 6.59 Beispiele von 16-Bit-Formaten. Der Speicher wird zum Lokalspeicher, der Arbeitsspeicher wird außen angeschlossen. Das A-Register dient auch zur Speicheradressierung.

a) EMIT OPs. Der 12-Bit-Direktwert läßt nicht viel Platz. Zwei 3-Bit-Funktionscodes werden eingesetzt, um vier Mikrobefehle zu codieren: das A-Register laden, vergleichen, erhöhen und vermindern.

b) ALF. Es lassen sich 32 Operationen codieren. Für die typischen Operandenverknüpfungen – wie in kleineren Mikrocontrollern üblich – dürfte das ausreichen. Vgl. auch S. 540f.

c) MODIFY. Eine Tetrade mit Auswahl läßt sich nicht unterbringen. Deshalb Beschränkung auf das Auswählen einzelner Bits. Der 3-Bit-Operationscode dürfte genügen, die typischen Einzelbitoperationen der Mikrocontroller zu codieren. Die ungenutzten Codes des Bitauswahlfelds BSEL (es sind nur 12 Bits auszuwählen, keine 16) kann man verwenden, um Operationen zu codieren, die den gesamten Inhalt der Speicherzelle betreffen (Löschen, Verschieben usw.).

d) JUMP, CALL. Wir beschränken uns auf die einfache Verzweigung. Wenn man mit den hiermit adressierbaren 4k Mikrobefehlen nicht auskommt, muß die Steuerspeicheradresse verlängert werden (Segmentierung).

e) BRANCH. Bedingtes Verzweigen mit einer kurzen Adresse. Das Beispiel zeigt 5 Bedingungs- und 8 Adreßbits. Die Adreßbits kann man in die Mikrobefehlsadresse einspeisen oder zur Mikrobefehlsadresse addieren. Wenn man mehr Bedingungen auswählen möchte, wird die Adresse kürzer. Die extreme Lösung: Überspringen (Skip) statt Verzweigen. Dann braucht man gar keine Adreßbits.

f) WCTL. Implementierung bei Bedarf.

g) CTL. Das Grundformat hat 13 Bits. Erweiterung nach dem Schema von Abb. 6.57 (S. 374).

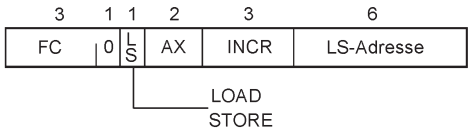


Abb. 6.60 Das Mikrobefehlsformat der Speicherzugriffe (Laden und Speichern).

Unterschiedlich lange Mikrobefehle

Es ist ein methodischer Ansatz, beim Konzipieren und Entwerfen nicht immerfort ans Sparen zu denken. Erst die Entwurfsabsicht umsetzen und so lange ändern, bis es funktionieren müßte, also erst klotzen, dann optimieren. In der Praxis – wenn man die Maschine wirklich bauen will – werden immer Kompromisse zu schließen sein; man wird nie Schaltungsressourcen unbeschränkt verbrauchen dürfen. Es ergibt sich dann die Frage, wo und wie gespart und optimiert wird. Nun kann man das von den Maschinen der Vergangenheit, den Minicomputern, Mikroprozessoren und Mikrocontrollern lernen. Wenn man mit kurzen Befehlsformaten auskommen muß, liegt es nahe, alles, was im aktuellen Befehl nicht unterzubringen ist, anderswo bereitzustellen, beispielsweise in Bankregistern, in Registern der Voreinstellung (Staticizers), in einem Lokalspeicher oder auf einem Stack. Eine Alternative besteht darin, Befehle unterschiedlicher Länge vorzusehen. Das Laden und Ausführen solcher Befehle erfordert aber einen höheren Steuerungsaufwand. Eigentlich ist das nur etwas für Maschinen mit Mikroprogrammsteuerung[49]. Um Ähnliches im Mikroprogrammsteuerwerk selbst vorzusehen, gibt es zwei grundsätzliche Alternativen:

- Einfachlösungen der Verlängerung, die mit wenig Steuerungsaufwand auskommen[50].
- Die Ablaufsteuerung der Mikrobefehle wird einem weiteren nachgeordneten speicherprogrammierten Steuerwerk übertragen (Nanoprogrammierung; vgl. Abschnitt 4.6, S. 198).

6.4.2 Universalregistermaschinen

Abb. 6.61 veranschaulicht eine Maschine mit einem universellen Arbeitsspeicher für Programme und Daten. Die architekturseitigen Register befinden sich im Lokalspeicher, ebenso die Arbeitsregister, die zum Emulieren der Befehlsfunktionen gebraucht werden (Scratchpad Area). Er hat drei Zugriffswege (Triple-Port Memory), zwei zum Lesen und einen zum Schreiben. Die Maschine führt Befehle aus, die im Arbeitsspeicher untergebracht sind. Das Mikroprogrammsteuerwerk steuert die Befehlsausführung. Die Adreßrechnung wird mit Mikrobefehlen ausprogrammiert. Im Interesse des Leistungsvermögens sind ein Befehlszähler und ein Befehlsregister für die Maschinenbefehle vorgesehen. Der Befehlszähler (Instruction Counter IC) erlaubt es, bereits den nächsten Befehl zu lesen, während der aktuelle ausgeführt wird (Instruction Lookahead). Das Befehlsregister (Instruction Register IR) hält die Parameter des aktuellen Befehls

49. Ja, es geht auch sequentiell. Eine Frage der Kompliziertheit und des Aufwands.
50. Als Beispiel siehe S. 357f und 393f, insbesondere die Abb. 6.42, 6.72 und 6.73. Vgl. zudem [56].

bereit (Operationscode, Direktwerte, Registeradressen usw.), so daß sie direkt in die Schaltung eingespeist werden können, ohne Umweg über den Lokalspeicher. Um eine solche Anordnung effektiv zu steuern, braucht man horizontale Mikrobefehle, die mehrere Signalwege gleichzeitig schalten können.

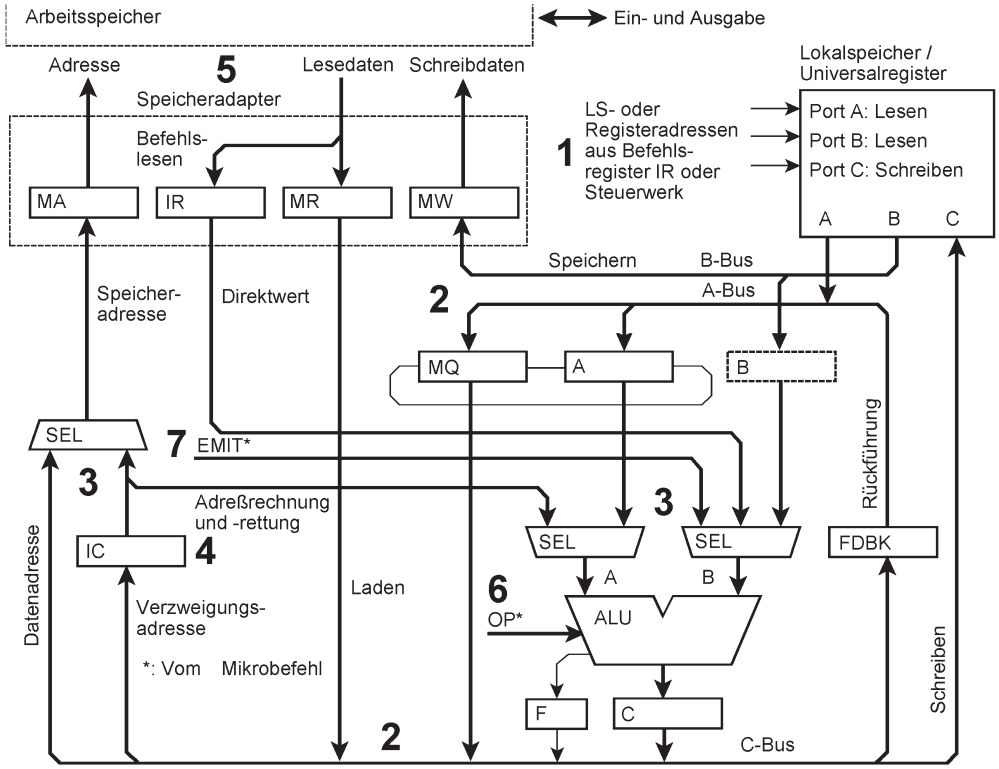


Abb. 6.61 Das Verarbeitungswerk einer typischen Universalregistermaschine. Sie hat einen Arbeitsspeicher für Programme und Daten (v. Neumann-Architektur) und einen Lokalspeicher. Wie das Steuerwerk angeschlossen wird, ist aus Abb. 4.5 ersichtlich. Die Bezugszeichen 1 bis 7 betreffen die Felder im Mikrobefehlsformat (Erläuterungen S. 381ff).

Anwendungsprogrammierung und Befehlsemulation

Maschinenbefehle können den gesamten Adreßaum ansprechen und mit allen Datenformaten der Architektur arbeiten. Will man die Anwendungsprobleme direkt mit Mikrobefehlen lösen (Micro-Architecture), müssen auch die Mikrobefehlsformate diese Anforderungen erfüllen. Um zu einer überlegenen Verarbeitungsleistung zu kommen (Speedup), liegt es nahe, die Maschine so zu dimensionieren, daß die Verarbeitungsbreite der jeweils längsten der elementaren Informationsstrukturen entspricht, also der Arbeitsspeicheradresse oder dem Datenwort. Der Mikrobefehl ist dann eine Art längerer und womöglich etwas extravagant formatierter RISC-Befehl. Die in Abschnitt 6.4.3 vorgestellten Maschinen sind so ausgelegt.

Beruht die Maschine hingegen auf architekturseitig definierten Maschinenbefehlen und dienen die Mikroprogramme nur dazu, die Befehlsabläufe zu steuern (Befehlsemulation), ergeben sich andere Anforderungen an die Gestaltung der Mikrobefehle. Es sind die Maschinenbefehle der Architektur, die dafür sorgen, daß die Maschine Turing-vollständig ist. Das Mikroprogrammsteuerwerk ist hier nur eine Art Sequencer; es muß sich mit dem jeweiligen Befehlsablauf beschäftigen und mit sonst nichts. Die Mikrobefehle müssen somit nicht unbedingt Direktwerte und Arbeitsspeicheradressen in voller Länge enthalten. Sie müssen aber an die Aufgaben der Befehlsemulation angepaßt sein. Das betrifft u. a. die Auswahl der zu verknüpfenden Operanden, die Steuerung der Verknüpfungsschaltungen und der Arbeitsspeicherzugriffe, die Verzweigungsbedingungen und die Funktionsverzweigungen. Manche Maschinen sind bis in die Schaltungseinzelheiten hinein auf eine bestimmte Zielarchitektur abgestimmt[51], andere sind von vornherein als universelle Emulatoren für beliebige Zielarchitekturen gedacht. Will man so etwas nur mit elementaren Mikrobefehlswirkungen erreichen[52], so bleibt das Leistungsvermögen beschränkt, oder die Maschine wird extrem aufwendig[53]. Es ist zudem keineswegs einfach, solche Maschinen gut auszunutzen. Auch bleibt die Frage, ob die Mikro-Architektur wirklich alles enthält, was beim Emulieren der vielfältigen Zielarchitekturen von Nutzen ist[54]. Eine Art Kompromißlösung besteht darin, das Mikroprogrammsteuerwerk mit kleinen Zusatzschaltungen zu ergänzen, die besonders leistungskritische Aufgaben der Befehlsemulation mit Hardware erledigen[55]. Beispiel: eine Registeradresse befindet sich in den Bits 23 bis 27 des Maschinenbefehls. Um damit zu arbeiten, könnte man das Befehlsbitmuster so verschieben, daß diese Adresse in den Bitpositionen 0 bis 4 zu liegen kommt, und die verbleibenen Bits löschen. Mit dieser Adresse wird dann zum Lokalspeicher zugegriffen. Schneller geht es aber mit einer Adapterschaltung, die vom Mikroprogramm angesprochen wird und die Bits direkt dort einspeist, wo sie gebraucht werden. Beim heutigen Stand der Technik kann man solche Hilfsschaltungen mittels Verhaltensbeschreibung erfassen und im FPGA synthetisieren.

Wie komplex das Steuerwerk wird, hängt von der Architektur der Maschinenbefehle ab. Sind die Befehlswirkungen vergleichsweise elementar (RISC), genügt eine direkte oder eine einfache sequentielle Steuerung. Sind aber komplexere Befehle auszuführen, Befehle, die typischerweise mehrere Taktzyklen erfordern (CISC), stehen zwei Auslegungen zur Wahl, die Mikroprogrammsteuerung und die sequentielle Steuerung. Früher kam die sequentielle Steuerung solcher Maschinen nur in den gehobenen Leistungsklassen in Betracht. Heute sind – von den Entwurfsschwierigkeiten her – beide Alternativen beherrschbar. Man kann die sequentielle Steuerung mit kleinen Zustandsautomaten aufbauen (vgl. Abb. 4.5), die man mittels Verhaltensbeschreibung erfaßt. Den Rest überläßt man der Schaltungssynthese. Die Entscheidung für eine der Alternati-

51. Beispielsweise S/360 und S/370. Vgl. Kapitel 9.
52. Das sind vor allem Einzelbit- und Bitfeldoperationen.
53. Beispiel: Mathilda ([M13], [M14]).
54. Deshalb das Ressourcenprinzip als grundsätzliche Alternative. Der Grundgedanke: Die Maschine sozusagen vom anderen Ende her entwickeln. Erst die Ressourcen zuammenstellen, dann überlegen, wie man sie steuern kann.
55. Sie werden in Anpassung an die Zielarchitektur entworfen. Vgl. das Transform Feature der CDC 5600 (S. 517ff).

ven ist Sache der Entwurfsoptimierung. Was aber grundsätzlich für die Mikroprogrammierung spricht, ist die Möglichkeit, in der fertigen Maschine beliebig zu ändern, ja sogar grundsätzlich andere Mikroprogramme zu laden. Das Ändern einer seqentiellen Steuerung hingegen müßte über die Schaltungssynthese laufen. Das aber braucht Zeit, und man weiß vorher auch nicht sicher, was herauskommt, vor allem, ob die neue Maschine noch in den Schaltkreis paßt und mit der bisherigen Taktfrequenz betrieben werden kann.

Der Speicheradapter
Die Implementierung von Speicheradapterschaltungen gehört zum Stand der Technik. Hier ist es nur eine symbolische Funktionseinheit, die die Informationsflüsse der Speicherzugriffe veranschaulicht. In den Blockschaltbildern sind einige Register dargestellt:

- das Speicheradreßregister MA (Memory Address),
- das Befehlsregister IR (Instruction Register),
- das Speicherlesedatenregister MR (Memory Read Data),
- das Speicherschreibdatenregister MW (Memory Write Data).

Ein Lesezugriff beginnt damit, die Adresse ins Register MA zu laden. Der Zugriff ist dann beendet, wenn die Lesededaten im Register IR (Befehlslesen) oder MR (Datenlesen) angekommen sind. Um einen Schreibzugriff auszuführen, müssen die Schreibdaten ins Register MW und die Adresse ins Register MA geladen werden. Wie das Warten auf die Erledigung des Zugriffs organisiert wird, ist Sache der Entwurfsoptimierung. Im einfachsten Fall verbleibt der Mikrobefehl, der den Zugriff ausgelöst hat, so lange im Wartezustand, bis der Zugriff zu Ende gekommen ist. Es liegt aber auch nahe, den Zugriff nur auszulösen und zunächst weiterzuarbeiten. Beim Lesen wird dann gewartet,wenn man die Daten braucht, sie aber noch nicht da sind, beim Schreiben, wenn der voraufgegangene Schreibzugriff noch anhängig ist. Auch können weitere Beschleunigungsmaßnahmen eingeführt werden, wie Caches oder Schreibpuffer.

Die Bussysteme
Es sind symbolische Darstellungen, die die Signalflüsse veranschaulichen sollen. Die Zugriffsbezeichnungen (Lesen, Schreiben) beziehen sich auf den Lokalspeicher. Ein Lesebus ist nur ein Verteiler von einer Quelle zu mehreren Zielen. Auf einen Schreibbus hingegen müssen mehrere Quellen aufschalten können, jeweils eine zu einer Zeit. Die traditionellen Lösungen mit Open-Collector- oder Tri-State-Koppelstufen kann man aber im Innern der Schaltkreise nicht implementieren. Die logische Bustopologie – von mehreren Quellen zu einem Ziel – muß deshalb mit anderen Mitteln implementiert werden, beispielsweise mit Auswahlschaltungen.

Mikrobefehlsformate
Die Abb. 6.62 und 6.63 zeigen ein horizontales Mikrobefehlsformat für die Maschine der Abb. 6.61. Die Mikrobefehle müssen u. a. drei Lokalspeicherzugriffswege, drei Bussysteme und drei Auswahlschaltungen steuern. Horizontale Mikrobefehle sind eigentlich nichts anders als Aneinanderreihungen von Auswahl- und Steuerbits. Es liegt nahe, eine solche Maschine als Ressourcenvektormaschine zu entwerfen. Welche Signale brauchen die zu steuernden Schaltungen,

welche Bedingungssignale liefern sie zurück, welche Adreßräume sind zu erreichen, welche Direktwerte bereitzustellen? Daraus ergibt sich, wieviele Bitpositionen anfänglich vorzusehen sind. Kürzen, sparen, optimieren kann man später. Abb. 6.62 zeigt das Mikrobefehlsformat im Überblick. Die Felder sind in Abb. 6.63 genauer dagestellt. Sie werden nachfolgend beschrieben.

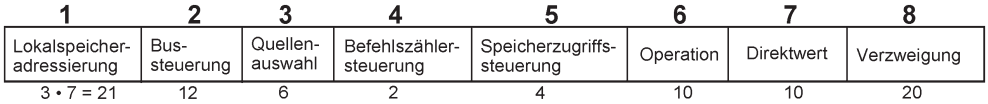
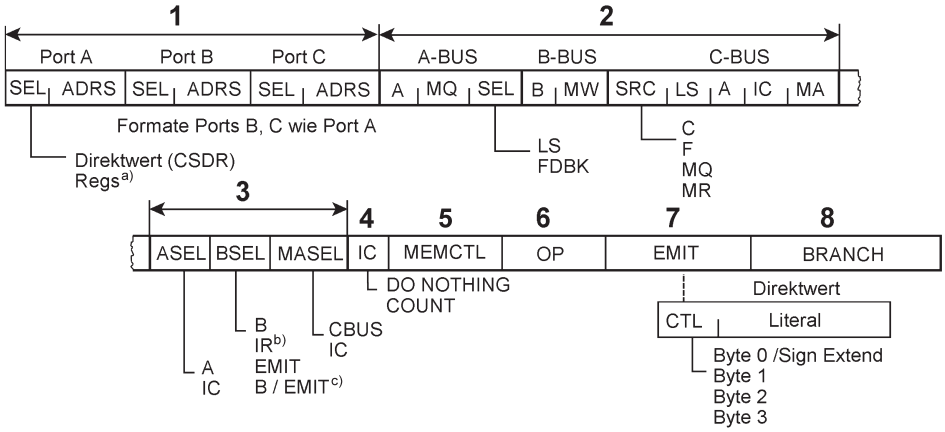


Abb. 6.62 Das Mikrobefehlsformat im Überblick. Die laufenden Nummern (Bezugszeichen) der Felder 1 bis 7 finden sich auch im Blockschaltbild (Abb. 6.61), das Feld Nr. 8 betrifft das Mikroprogrammsteuerwerk. Die Anzahl der Mikrobefehlsbits (untere Zahlenreihe) wird im Anschluß an die Beschreibung erläutert (S. 384f). Der Mikrobefehl ist insgesamt 85 Bits lang.



a) Felder im Befehlsregister (IR), STATs, Adapter. Auswahl jeweils im zugehörigen ADRS-Feld
b) Ggf. mehrere Befehlsfelder zur Auswahl
c) Ein Byte Literal gemäß EMIT-Feld, die restlichen Bytes aus dem B-Register

Abb. 6.63 Das Mikrobefehlsformat im einzelnen.

1 – Lokalspeicheradressierung

Jeder der drei Lokalspeicherports A, B, C hat ein Auswahlfeld (SEL) und ein Adreßfeld (ADRS). In unserem Beispiel hat der Lokalspeicher eine Speicherkapazität von 64 Maschinenwörtern. Eine Lokalspeicheradresse kann aus folgenden Quellen stammen:

- aus dem zugehörigen Adreßfeld (ADRS) im Mikrobefehl (Direktwert im Mikrobefehlsregister CSDR),
- aus Registern (Regs). Die Registerauswahl ist im zugehörigen Adreßfeld (ADRS) codiert. Typischerweise stammen solche Registeradreßfelder aus dem Befehlsregister (IR) oder aus Voreinstellregistern (Staticizers) und Adapterschaltungen im Mikroprogrammsteuerwerk.

2 – Bussteuerung
Jedes der drei Bussysteme hat einzelne Steuerbits zum Laden der jeweiligen Zielregister. Damit können mehrere Zielregister gleichzeitig geladen werden. Der A-Bus und der C-Bus haben zudem Auswahlfelder für die jeweilige Datenquelle:

- Die Daten auf dem A-Bus kommen aus dem Lokalspeicher oder vom C-Bus (Rückführung; FDBK = Feedback). Sie können ins A-Register und ins MQ-Register geladen werden.
- Die Daten auf dem B-Bus kommen stets aus dem Lokalspeicher. Sie können ins B-Register und ins Speicherschreibdatenregister (MW) geladen werden.
- Die Daten auf dem C-Bus kommen aus dem C-Register, dem Bedingungsregister (F), dem MQ-Register oder dem Speicherlesedatenregister (MR). Sie können in den Lokalspeicher, ins A-Register (Rückführung), in den Befehlszähler (IR) oder ins Speicheradreßregister (MA) geladen werden.

3 – Quellenauswahl
Auswahlschaltungen befinden sich vor dem A-Eingang der ALU, dem B-Eingang der ALU und dem Speicheradreßregister (MA). Jede dieser Auswahlschaltungen hat ein eigenes Auswahlfeld:

- Auswahlfeld ASEL: Am A-Eingang können das A-Register oder der Inhalt des Befehlszählers (IC) ausgewählt werden, letzteres u. a. zur Adreßrettung beim Unterprogrammruf.
- Auswahlfeld BSEL: Am B-Eingang können das B-Register oder Direktwerte aus dem Befehlsregister (IC) oder aus dem Mikrobefehl (EMIT) ausgewählt werden.
- Auswahlfeld MASEL: Das Speicheradreßregister (MA) kann mit den Daten des C-Bus oder mit dem Inhalt des Befehlszählers (IR) geladen werden.

4 – Befehlszählersteuerung
In diesem Feld ist codiert, ob der Befehlszähler weiterzählt[56] oder seinen Inhalt behält. Das Laden des Befehlszählers wird im Steuerfeld des C-Bus angewiesen.

5 – Speicherzugriffssteuerung
Das Speichersteuerfeld dient zum Auslösen von Speicherzugriffen (Lesen = RD, Schreiben = WR, Befehlslesen = IF). In diesem Feld können auch Steuersignale codiert sein, z. B. Freigeben beim Lesen, Freigeben beim Schreiben, Warten auf Lesedaten, Warten auf Ende des Schreibzugriffs. Freigeben heißt, den Zugriff nur auslösen. Gewartet wird dann ggf. später.

6 – Operation
Hier sind die Operationen der ALU codiert. Die typischen Operationen der ALU-Schaltkreise, der Mikrocontroller und der Prozessoren höherer Leistungsklassen mögen hierfür als Anregung dienen. Vgl. auch Anhang 2, S. 540ff.

56. Ggf. auch mit wählbarer Zählweite (+1, +2 usw.).

7 – Direktwert

Der Direktwert aus dem Mikrobefehl fließt in den B-Eingang der ALU ein. In Abb. 6.63 ist angedeutet, daß man mit einem kurzen Direktwertfeld auskommt, wenn man dessen Inhalt in unterschiedliche Bitpositionen einspeisen kann. Hier sind es die einzelnen Bytes des Maschinenworts. Die verbleibenden Bytes sind entweder Nullen (BSEL = EMIT) oder stammen aus dem B-Register (BSEL = B/EMIT). Ist BSEL = EMIT und CTL = 0, wird das Direktwertbyte auf die gesamte Verarbeitungsbreite erweitert (Vorzeichenerweiterung).

8 – Verzweigung

Dieses Feld steuert die Auswahl des nachfolgenden Mikrobefehls. Länge und Gestaltung richten sich nach den gewählten Adressierungsprinzipien und Verzweigungsverfahren.

Adressen und Daten aus dem Befehlsregister

Das Befehlsregister enthält die Parameter des aktuell zu emulierenden Befehls. Das Mikroprogramm kann diese Felder auswählen und in die Maschine einfließen lassen. Abb. 6.64 zeigt einige typische Befehlsformate.

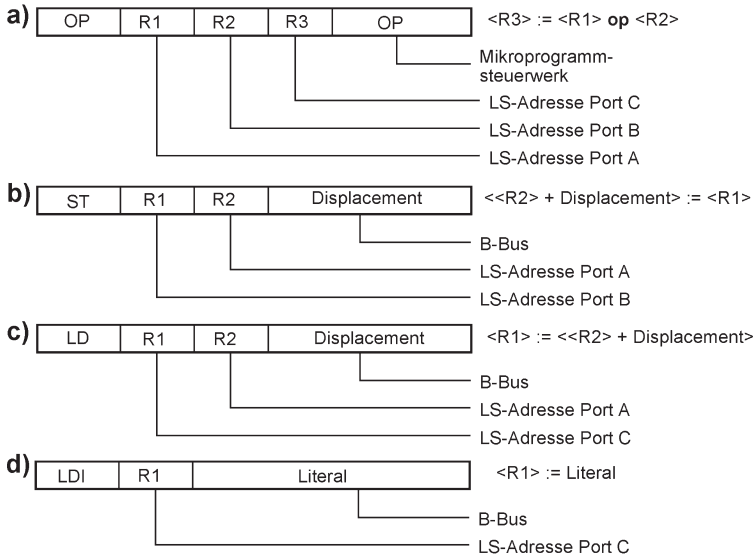


Abb. 6.64 Typische Befehlsformate. a) Operationsbefehl mit drei Registern (wie RISC); b) Speicherbefehl. Der Inhalt von R1 wird gespeichert. c) Ladebefehl. R1 wird geladen. Adressierung beim Speichern und Laden: Basisadresse in R2 + Displacement im Befehl. d) Laden eines Direktwerts (Literal) in das Register R1.

Das Mikroprogramm braucht den Operationscode, die Registeradressen usw. Nun kann man diese Felder aus dem Befehl herauslösen, indem man das Befehlswort in ein Arbeitsregister lädt, passend verschiebt, die unnötigen Bits löscht usw. Das kostet aber Mikrobefehlszyklen. Alternativ dazu kann man die jeweiligen Bits über Auswahlschaltungen direkt als Lokalspei-

cheradresse, Displacement, Direktwert usw. in das Verarbeitungswerk einspeisen. Im Beispiel müßte das Registerfeld R1 für alle drei Lokalspeicherports als LS-Adresse ausgewählt werden, das Registerfeld R2 für die Ports A und B und das Registerfeld 3 für Port C. Dem B-Eingang der ALU sind wahlweise das Displacement- oder das Direktwertfeld (Literal) zuzuführen.

Wie lang ist der Mikrobefehl?
Mit den Werten von Abb. 6.62 sind es 85 Bits. Das beruht auf 6 Bits langen Lokalspeicheradressen, einem Direktwertfeld von 10 Bits (mit 8 Bits Direktwert) und einem Verzweigungsfeld von 20 Bits[57]. In den Feldern 2 bis 5 und 8 wurden einige Bits Reserve zugegeben; das Operationsfeld wurde mit 10 Bits großzügig dimensioniert. Wenn man Wortlängen einhalten muß, die Industriestandards sind, wären mit etwas Optimierung und einigen Voreinstellregistern (Staticizers) sicherlich 64 Bits zu erreichen, wenn man mehr braucht, könnte man auf 96 Bits gehen.

Die Maschine vereinfachen
Die Maschine wird kleiner und einfacher, wenn man Register aus der Schaltung in den Lokalspeicher verlagert und die Anzahl der Zugriffswege zum Lokalspeicher verringert (Abb. 6.65).

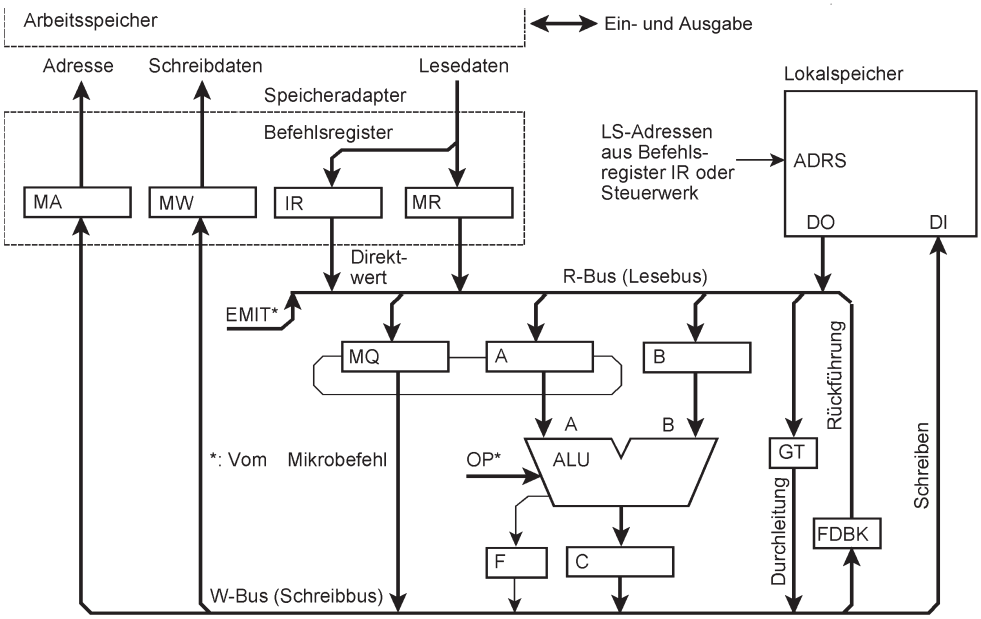
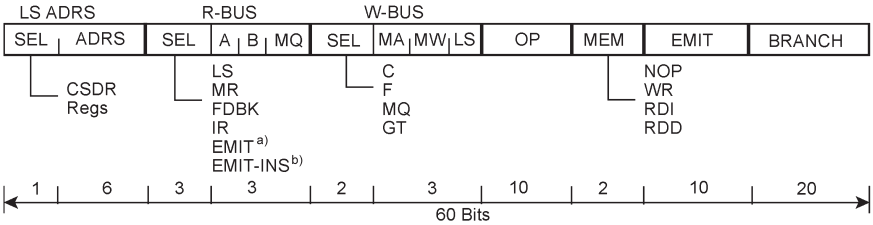


Abb. 6.65 Eine vereinfachte Maschine (Sparlösung). Der Lokalspeicher hat nur einen Zugriffsweg. Er nimmt auch die Befehlsadresse auf. Mikroprogrammsteuerwerk und E-A-Ports nicht dargestellt. GT = Durchleitung zum Schreibbus (Gate-Thru), FDBK = Rückführung auf den Lesebus.

57. Vgl. die Abb. 6.53 (S. 370). Dort sind es max. 18 Bits. Hier wurden 2 Bits Reserve hinzugefügt. Beim Verzweigen sollten wir nicht um jeden Preis sparen; die 20 Bits sind nicht zuviel ...

In dieser Schaltung gibt es nur einen einzigen Zugriffsweg und zwei Bussysteme, eines zum Lesen, eines zum Schreiben. Auch der Befehlszähler befindet sich im Lokalspeicher. Zum Befehlslesen braucht man ein Mikroprogramm. Das Befehlsregister hingegen ist nach wie vor ein Hardware-Register, weil es als Quelle von Parametern dient (Operationscodes, Direktwerte, Adressen).
Das horizontale Mikrobefehlsformat von Abb. 6.66 ist ebenso aufgebaut wie das der Abb. 6.62 und 6.63. Hier sind aber nur eine Lokalspeicheradresse bereitzustellen und zwei Bussysteme zu steuern, deshalb gibt es weniger Mikrobefehlsfelder. Die Maschine hat ein B-Register, und sie kann die Lokalspeicherdaten zum Schreibbus durchleiten (Gate-Thru, GT). Damit ist es nicht erforderlich, den Umweg über die ALU zu nehmen, wenn beispielsweise eine Speicheradresse aus dem Lokalspeicher ins MA-Register geladen werden soll. Beide ALU-Operanden A, B bleiben dabei in ihren Registern erhalten.



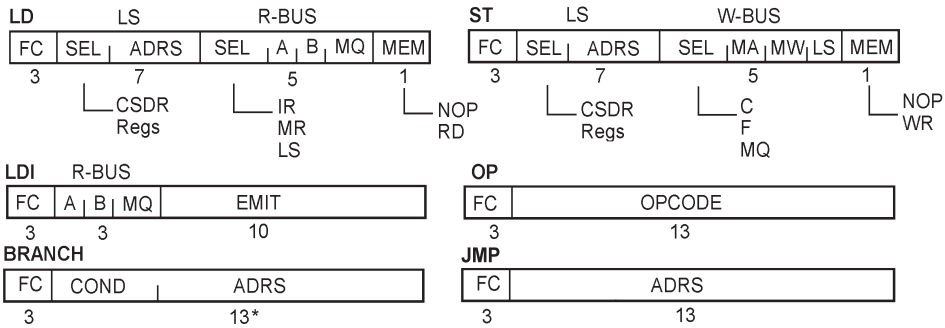
a) Den Direktwert ins Register laden.
b) Den Direktwert gemäß Byteauswahl im EMIT-Feld in das zu ladende Register einfügen (INS = Insert).

Abb. 6.66 Ein horizontales Mikrobefehlsformat für die Maschine von Abb. 6.65. Mit plausiblen Annahmen ergibt sich eine Mikrobefehlslänge von 60 Bits.

Vertikale Mikrobefehle

Für die Maschine von Abb. 6.61 lohnen sie sich kaum, weil alles, was nicht parallel aus dem Mikrobefehlsregister kommt, in Voreinstellregistern gehalten werden müßte. Womöglich kann man die Mikrobefehle verkürzen, so daß man mit einem kleineren Steuerspeicher auskommt. Dem Prinzip nach würden aber weder Flipflops noch Speicherkapazität gespart werden.
In der vereinfachten Maschine von Abb. 6.65 verhält es sich anders. Abb. 6.67 zeigt vertikale Mikrobefehlsformate, die durch Zerlegen des horizontalen Formats von Abb. 6.66 gewonnen wurden. Es liegt nahe, Mikrobefehle zum Laden (LD), zum Speichern (ST), zum Laden von Direktwerten (LDI), zum Auslösen von Operationen (OP) und zum Verzweigen (BRANCH) vorzusehen. Wie aus der Abbildung hervorgeht, ist es durchaus möglich, mit 16 Bits auszukommen[58]. Das Mikroprogrammsteuerwerk wird weniger aufwendig, die Programmierung einfacher. Dafür wird die Maschine langsamer.

58. Das vor allem, weil die Emulation eines einzelnen Befehls nicht so große Sprungweiten und lange Direktwerte erfordert. Als Beispiel einer Maschine, die mit 16-Bit-Mikrobefehlen eine komplexe Befehlsliste emulieren kann, mag IBMs 360/25 dienen (Kapitel 9, S. 502ff).



* Die Länge des Bedingungs- und des Adreßfeldes (COND, ADRS) ist Sache der Feinoptimierung (Kompromiß zwischen Bedingungsauswahl und Sprungweite). Wir brauchen eine unbedingte Verzweigung (Jump) mit maximaler Adreßlänge (12, besser 13 Bits). Die Extremlösung des bedingten Verzweigens wäre ein Übersprungformat (Skip) mit einem langen Bedingungsauswahlfeld, aber sehr kurzer oder gar keiner Adresse.

Abb. 6.67 Vertikale Mikrobefehle für die Maschine von Abb. 6.65. Hier sind sie 16 Bits lang.

6.4.3 Mikrobefehle als Maschinenbefehle

Im folgenden geht es darum, die Mikroprogrammierung von Anwendungsprogrammen zu unterstützen. Weshalb erst eine komplexe Befehlsliste implementieren, wenn man das Anwendungsproblem gleich mit dem Mikroprogramm erledigen kann? Die Absicht ist, die Maschine von Grund auf für diesen Zweck zu entwerfen[59].

In der Praxis wird man typischerweise nicht die gesamte Programmierabsicht, sondern nur leistungsentscheidende Abläufe direkt in Mikroprogramme umsetzen. Alles andere wird mit Aufrufen vorgefertigter Funktionen erledigt. Das können herkömmliche Maschinenbefehle sein, Maschinenbefehle fiktiver Maschinen[60] oder auch mikroprogrammierte Funktionen zum Interpretieren höherer Programmiersprachen (High-Level Instruction Interface). Die Betrachtungen beginnen mit einer vergleichsweise einfachen Maschine (Abb. 6.68).

Das Registermodell entspricht weitgehend dem der typischen einfachen Maschinen, wie sie aus der Entwicklungsgeschichte bekannt sind. Die Operanden-, Ergebnis- und Bedingungsregister des Verarbeitungswerks werden um einen Lokalspeicher und einen Registerstack ergänzt. Ansonsten wird gespart; die Adreßregister befinden sich im Lokalspeicher, die Adreßrechnung der Speicherzugriffe ist auszuprogrammieren. Im Beispiel wird eine Maschinenwortlänge = Adreßlänge von 32 Bits angenommen. Die Mikrobefehle sind 16 Bits lang. Die kleinste adressierbare Informationsstruktur muß deshalb das 16-Bit-Wort sein, man kann aber auch Byteadressierung vorsehen. Alle Informationsstrukturen müssen an integralen Adressen gespeichert werden.

59. Also nicht als Weiterbildung der Maschinen, die bisher vorgestellt wurden.
60. Beispielsweise JVM oder Dalvik.

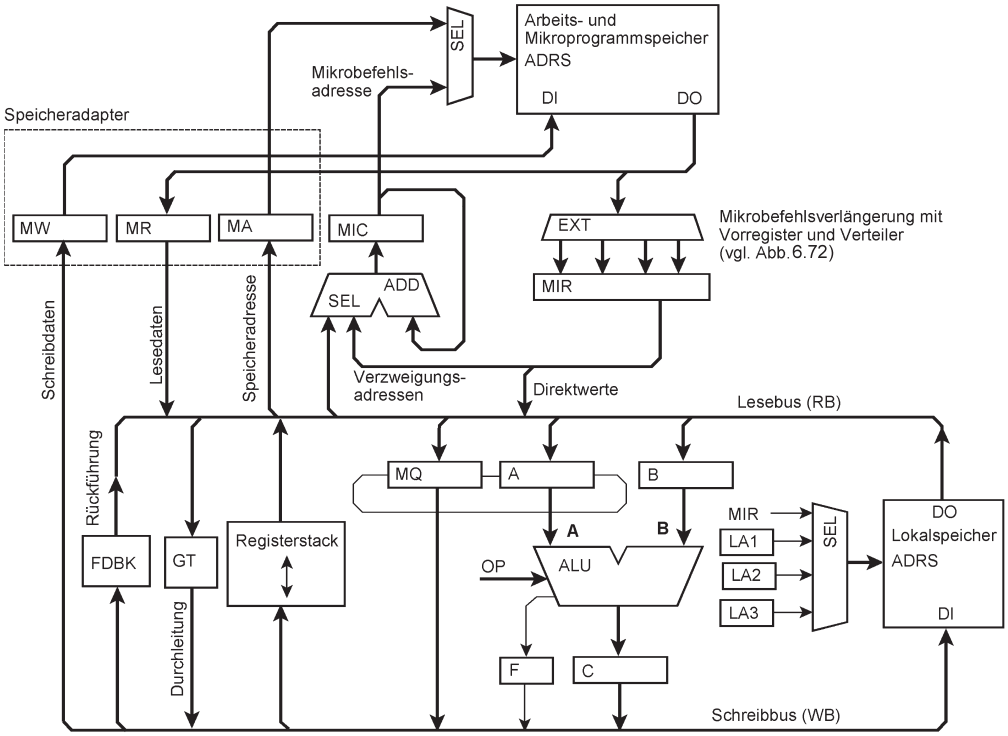


Abb. 6.68 Eine einfache Maschine, um Anwendungsprogramme mit Mikrobefehlen zu implementieren. Die Mikrobefehle sind zugleich die Maschinenbefehle.

Die Anregungen zu den grundsätzlichen Entwurfsgedanken stammen teils aus der Entwicklungsgeschichte, teils aus theoretischen Untersuchungen zur sog. Turing-Einadreßmaschine[61]. Dabei hat es sich ergeben, daß sich anwendungsseitig mikroprogrammierbare Maschinen ableiten lassen, indem man die Turing-Einadreßmaschine in systematischen Schritten entsprechend weiterbildet (Praxisverbesserung).

Arbeits- und Mikroprogrammspeicher

Es wird ein gemeinsamer Speicher für alles angenommen. Man kann aber auch den Adreßraum in verschiedenartige Speicher aufteilen (ROM, RAM, besonderer Mikroprogrammspeicher usw.) und Caches vorsehen.

61. Dabei geht es – kurz gesagt – darum, Turingmaschinen tatsächlich zu bauen, ja sogar eine gewisse Anwendungsbrauchbarkeit darzustellen. Diese Maschinen haben aber kein Band, sondern einen adressierbaren Speicher. Einzelheiten in [56].

Speicheradapter
Er ist hier nur eine symbolische Funktionseinheit, um die Informationsflüsse der Speicherzugriffe zu veranschaulichen. Für jede Zugriffsrichtung (Lesen, Schreiben) ist jeweils ein Datenpufferregister (MR, MW) vorgesehen.

Ein- und Ausgabe
Diese Vorkehrungen sind hier nicht dargestellt. E-A-Schaltungen können an die Register der Verarbeitungseinheit, an den Lokalspeicher und über den Speicheradreßraum angeschlossen werden.

Speicheradressierung
Es gibt zwei Quellen: für Datenzugriffe das Speicheradreßregister MA, für Mikrobefehlszugriffe der Mikrobefehlszähler MIC. Mikrobefehle werden wie übliche Maschinenbefehle adressiert (Adreßzählung, Laden von Verzweigungsadressen). Die Auswahl der Verzweigungsadresse und die Adreßrechnung mit dem Mikrobefehlszähler[62] als Basisadreßregister sind schematisch dargestellt.

Mikrobefehlsregister
Die Maschine wird als Ressourcenvektormaschine entworfen. Demgemäß ist das Mikrobefehlsregister MIR ein langes Register, das alle erforderlichen Steuerbits, Direktwerte, Codefelder usw. aufnehmen kann.

Lokalspeicher
Die Maschine hat nur die nötigsten Hardwareregister. Alle weiteren Register sind im Lokalspeicher untergebracht. Er hat eine Speicherkapazität von maximal 256 Maschinenwörtern[63]. Das sollte genügen, um alle Adreßregister, Hilfsregister usw. sowie jene Parameter aufzunehmen, die auf der untersten Ebene der Mikroprogramme benötigt werden. Weitere Bereiche (für Interpretationszwecke, für die Ein- und Ausgabe usw.) können im Arbeitsspeicher selbst oder im Speicheradreßraum vorgesehen werden. Die Speicherschnittstelle kann man so ausgestalten, daß solche Bereiche anderweitig unzugänglich sind (vgl. Beispiel der IBM 360/25; S. 502).

Lokalspeicheradressierung
Der Lokalspeicher kann in den Mikrobefehlen direkt oder über drei wählbare Adreßregister LA1, LA2, LA3 adressiert werden (indirekte Adressierung).

Registerstack
Da die weitaus meisten Abläufe, einschließlich der Arbeitsspeicherzugriffe, im Mikroprogramm ausprogrammiert werden müssen, ist es zweckmäßig, dafür zu sorgen, daß elementare Unterprogramme schnell – also mit geringst-möglichem Overhead – aufgerufen werden können.

62. Die Maschine kann auch für absolute Verzweigungsadressen und Segmentierung ausgelegt werden.
63. Im praktischen Schaltungsentwurf muß man nur soviel Lokalspeicherkapazität implementieren wie jeweils nötig. Praxistip: Genügend Reserven vorsehen ...

Dafür wurde ein Registerstack vorgesehen. Richtwert: 16 Einträge zu 32 Bits. Er kann Rückkehradressen und Registerinhalte aufnehmen. In besonders sparsamen Implementierungen kann die Tiefe verringert werden, bis hin zum einzelnen Linkregister[64].

Die Zugriffswege der Register

Die Hardwareregister, der Lokalspeicher und der Registerstack sind an zwei Bussysteme angeschlossen, an den Lesebus und den Schreibbus (Abb. 6.69). Die Busbezeichnungen betreffen die Zugriffe zum Lokalspeicher, zum Registerstack und zum Arbeitsspeicher. Der Lesebus dient zum Lesen aus diesen Speichern, der Schreibbus zum Schreiben. Die Register werden über den Lesebus geladen, Registerinhalte über den Schreibbus gespeichert. Der Schreibbus ist auf den Lesebus zurückgeführt (Funktionsblock FDBK = Feedback), der Lesebus kann mit dem Schreibbus verbunden werden (Funktionsblock GT = Gate-Thru). Somit sind Informationstransporte zwischen allen Einrichtungen möglich, die an beide Bussysteme angeschlossen sind.

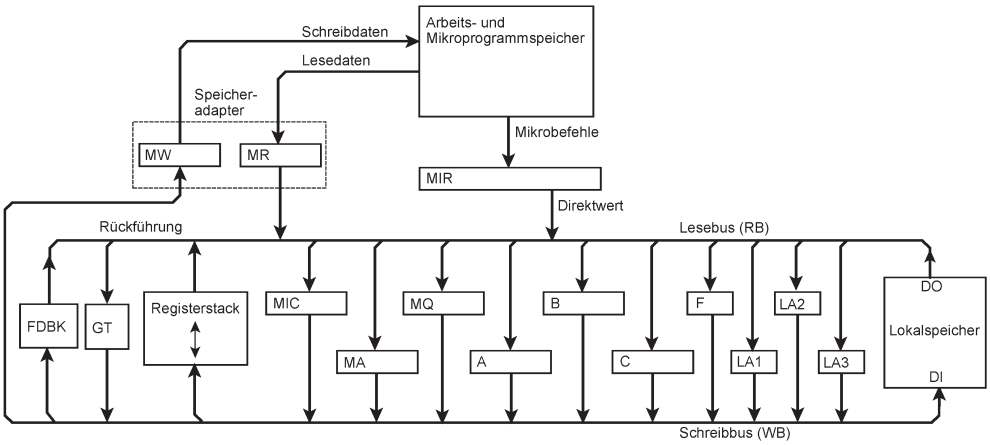


Abb. 6.69 Die Zugriffswege der Register. Sie sind hier mit zwei Bussystemen symbolisch dargestellt. Auf dem Schaltkreis werden sie mit den jeweils verfügbaren Mitteln implementiert[65].

Mikrobefehlsformate

Die elementaren Mikrobefehle sind 16 Bits lang (Abb. 6.70). Wenn die 16 Bits nicht reichen, wird der Mikrobefehl verlängert. Es ist eine Einfachlösung, ein einziges Erweiterungsfomat, das 12 zusätzliche Bits ins Mikrobefehlsregister einträgt. Das Mikrobefehlsregister kann zwei solcher Erweiterungen aufnehmen. Somit ergeben sich insgesamt drei Längen: 16, 28 und 40 Bits. Die aktuelle Länge wird in zwei zusätzlichen Bits (Längencode LC) mitgespeichert. Abb. 6.71 zeigt die 16-Bit-Formate. Tabelle 6.3 gibt einen Überblick über die Mikrobefehlstypen.

64. Was das Retten von Registerinhalten ausschließt. Zu rettende Registerinhalte müßten dann in den Lokalspeicher transportiert werden. Die Schachtelung von Unterprogrammen wäre auszuprogrammieren (Software-Stack).
65. Z. B. mit Multiplexern, Schalterelementen oder Übertragungsgattern (Transfer-Gates).

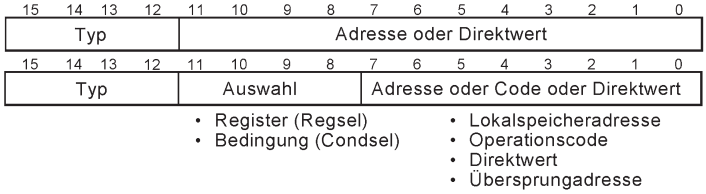


Abb. 6.70 Die elementaren Mikrobefehle sind 16 Bits lang. Einem Typcode von 4 Bits folgen 12 Bits Inhalt (Payload) nach. Die meisten Mikrobefehlstypen haben ein Auswahlfeld von 4 Bits. Die nachfolgenden 8 Bits sind eine Adresse, ein Direktwert oder ein Operationscode.

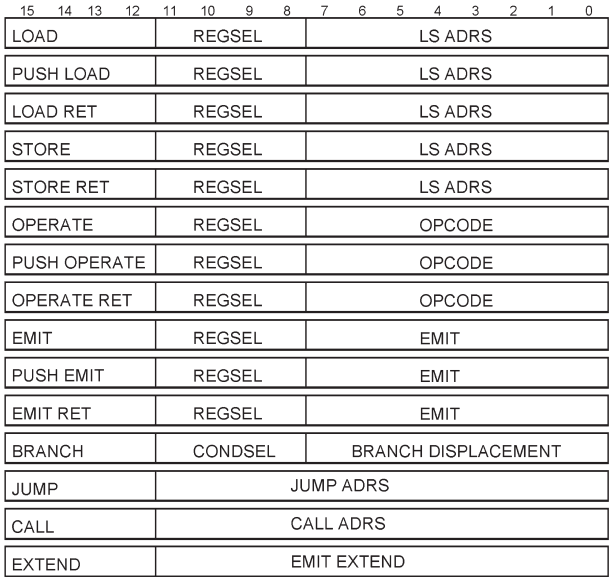


Abb. 6.71 Die 16-Bit-Formate.

Mikrobefehl	Wirkung	Besonderheiten
LOAD	Den adressierten Lokalspeicherinhalt (LS-Inhalt) in das ausgewählte Register transportieren	LOAD (MA) = LS-Inhalt nach Arbeitsspeicher (Schreibzugriff). LOAD STACK = LS-Inhalt in den Stack schaffen (Push). LOAD MIC = Verzweigen mit Mikrobefehlsadresse aus LS
PUSH LOAD	Das ausgewählte Register auf den Registerstack retten (Push), dann mit dem Inhalt des LS laden	

Mikrobefehl	Wirkung	Besonderheiten
LOAD RET	Das ausgewählte Register mit dem Inhalt des LS laden, dann Rückkehr mit Adresse aus dem Registerstack	
STORE	Den Inhalt des ausgewählten Registers in den LS transportieren	STORE (MA) = Speicherinhalt nach Lokalspeicher = Lesezugriff. STORE STACK = TOS nach LS (Pop). STORE MIC = Retten der Mikrobefehls-adresse in den LS
STORE RET	Den Inhalt des ausgewählten Registers in den LS transportieren, dann Rückkehr mit Adresse aus dem Registerstack	
OPERATE	Die Operation ausführen und das Ergebnis im ausgewählten Register (Zielregister) speichern	OPERATE (MA) = Ergebnis nach Arbeitsspeicher = Schreibzugriff. OPERATE STACK = Ergebnis in den Stack schaffen (Push). OPERATE MI = das Ergebnis als Verzweigungsadresse verwenden. OPERATE (LA1) oder (LA2/3) = das Ergebnis in den Lokalspeicher schaffen. LS-Adresse in LA1 oder LA3
PUSH OPERATE	Das Zielregister auf den Registerstack retten (Push), dann das Ergebnis speichern	
OPERATE RET	Den OPERATE-Mikrobefehl ausführen, dann Rückkehr mit Adresse aus dem Registerstack	
EMIT	Das ausgewählte Register (Zielregister) mit dem Direktwert laden	Zielregisterauswahl wie in OPERATE. EMIT MI wirkt wie BRANCH, aber als unbedingte Verzweigung
PUSH EMIT	Das Zielregister auf den Registerstack retten (Push), dann mit dem Direktwert laden	
EMIT RET	Das Zielregister mit dem Direktwert laden, dann Rückkehr mit Adresse aus dem Registerstack	
BRANCH	Bedingte Verzweigung	
JUMP	Unbedingte Verzweigung	
CALL	Unterprogrammruf mit Adreßrettung auf dem Registerstack	

Tabelle 6.3 Die Mikrobefehle im Überblick.

Anmerkungen:

- Es ist die Frage, was man in welchem Format unterbringt. Hier werden längere Lokalspeicheradressen und Operationscodes bevorzugt. Der Operationsmikrobefehl bringt nicht den zweiten Operanden mit, sondern weist an, wo das Ergebnis zu speichern ist[66].
- Der Operationscode bestimmt, welche Registerinhalte miteinander verknüpft werden. Das Regsel-Feld gibt nur das Zielregister an, in dem das Ergebnis zu speichern ist.
- Zu den OPERATE-Mikrobefehlen gehören NOP (nichts tun) und POP = das Zielregister mit dem Wert aus dem Registerstack laden.
- Weil die Maschine als universeller Interpreter oder Emulator eingesetzt werden soll, liegt es nahe, leistungsfähige Bitfeldoperationen zu implementieren, vor allem mit dem Ziel, die Gewinnung von Verzweigungsadressen zu unterstützen.
- Manche Parameterkombinationen sind offensichtlich unsinnig oder können von der hier dargestellten Hardware nicht unterstützt werden. Es versteht sich von selbst, solche Kombinationen in den Programmen nicht zu verwenden.

Registerauswahl und indirekte Lokalspeicheradressierung
Tabelle 6.4 gibt eine Überblick über die Registerauswahl im Regsel-Feld.

Indirekte Adressierung des Lokalspeichers:
- Wenn der Mikrobefehl keine Lokalspeicheradresse enthält[67]: mit Regsel = <LA1> oder <LA2/3>, wie in Tabelle 6.4 angegeben.
- Wenn der Mikrobefehl die Lokalspeicheradresse enthält:
  - Lokalspeicheradresse 0 = Adressierung mit dem Inhalt von LA1.
  - Lokalspeicheradresse 1 = Adressierung mit dem Inhalt von LA2 beim Lesen oder von LA3 beim Schreiben.

Inhalt	Auswahl
–	Nichts (kein Register ausgewählt)
A	A-Register
B	B-Register
C	C-Register (Ergebnis)
MQ	MQ-Register
F	Flagregister (Bedingungen)
MA	Speicheradreßregister
MIC	Mikrobefehlsadresse. Schreiben = Verzweigen, Lesen vor allem zum Retten
STACK	Registerstack. Push beim Schreiben, Pop beim Lesen
LA1	Lokalspeicheradreßregister 1

66. Dieses Prinzip wird auch in der Turing-Einadreßmaschine verwendet (vgl. [56]).
67. Sondern einen Operationscode oder ein Emit-Feld (vgl. Abb. 6.71).

Inhalt	Auswahl
LA2	Lokalspeicheradreßregister 2
LA3	Lokalspeicheradreßregister 3
<MA>	Speicherzugriff mit Adresse in MA
<LA1>	Lokalspeicherzugriff mit Adresse in LA1
<LA2/3>	Lokalspeicherzugriff mit Adresse in LA2 (Lesen) oder LA3 (Schreiben)

Tabelle 6.4 Die Registerauswahl im Regsel-Feld.

Mikrobefehle verlängern

Abb. 6.72 veranschaulicht das Prinzip, Abb. 6.73 zeigt die Formate, die sich durch das Verlängern ergeben. Aus Tabelle 6.5 ist ersichtlich, um wieviele Bits die einzelnen Felder erweitert werden. Wir nutzen das Prinzip des Erweiterungsmikrobefehls. Einem 16-Bit-Mikrobefehl, der eine Wirkung auslöst (das sind die Typen LOAD bis CALL), können bis zu zwei Erweiterungsmikrobefehle (EXTEND) vorhergehen. Sie tragen ihre 12 Bits Inhalt (Payload) in ein Vorregister ein, der erste dieser Mikrobefehle in die erste Hälfte, der nachfolgende in die zweite. Wird anschließend ein 16-Bit-Mikrobefehl eines anderen Typs gelesen, so wird der Inhalt des kompletten Vorregisters in das lange Mikrobefehlsregister MIR übernommen. Dann wird das Vorregister gelöscht. Der Typ des Mikrobefehls bestimmt, wie der Inhalt des Vorregisters zur Verlängerung der Mikrobefehlsfelder beiträgt.

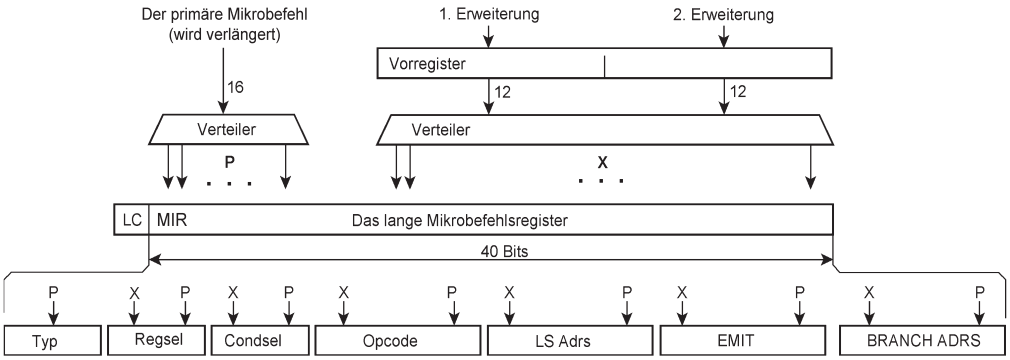


Abb. 6.72 So werden die Mikrobefehle verlängert. P = primärer Mikrobefehl (16 Bits), X = Erweiterung mit insgesamt 12 oder 24 Bits.

Horizontale Mikrobefehle

Solche Formate ergeben sich, indem man alles aneinanderhängt, was erforderlich ist, um die Maschine zyklusweise zu steuern und mit Parametern zu versorgen. Abb. 6.74 zeigt ein Beispiel (S. 395). Dieses Format ist ähnlich ausgelegt wie das von Abb. 6.66 (S. 385).

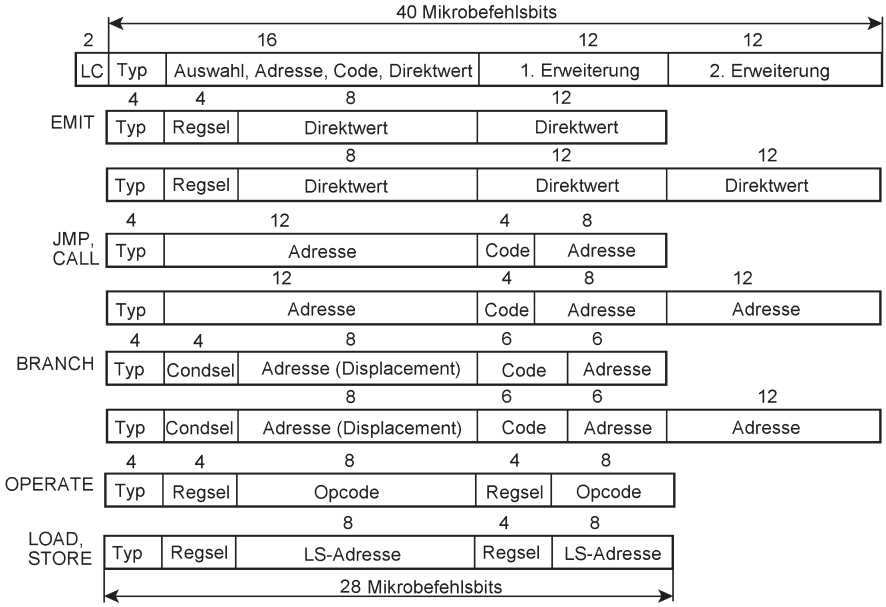


Abb. 6.73 Verlängerte Mikrobefehlsformate (28 und 40 Bits).

Mikrobe-fehlstyp	Feldinhalt	16 Bits	28 Bits	40 Bits
EMIT	Registerauswahl, Direktwert	4 Bits Regsel, 8 Bits Direktwert	4 Bits Regsel, 20 Bits Direktwert	4 Bits Regsel, 32 Bits Direktwert
JMP, CALL	Adresse	12 Bits	20 Bits + 4 Bits Code[1]	32 Bits + 4 Bits Code[1]
BRANCH	Adresse, Bedingung	4 Bits Condsel, 8 Bits Adresse	14 Bits + 6 Bits Code/Condsel[1]	26 Bits + 6 Bits Code/Condsel[1]
OPERATE	Registerauswahl, Operationscode	4 Bits Regsel, 8 Bits Opcode	8 Bits Regsel / Code[2], 16 Bits Opcode	res.[3]
LOAD, STORE	Registerauswahl, LS-Adresse	4 Bits Regsel, 8 Bits LS-Adresse	8 Bits Regsel / Code[2], 16 Bits LS-Adresse[4]	res.[3]

1. In diesen Bitpositionen können zusätzliche Funktionen oder Bedingungen codiert werden.
2. Vier zusätzliche Bits zur Erweiterung der Registerauswahl und zum Codieren von Funktionen.
3. Dieses Mikrobefehlsformat ist reserviert und wird hier nicht unterstützt.
4. Die verlängerte LS-Adresse könnte beispielsweise zu E-A-Zwecken verwendet werden.

Tabelle 6.5 Wie die Mikrobefehlsformate verlängert werden.

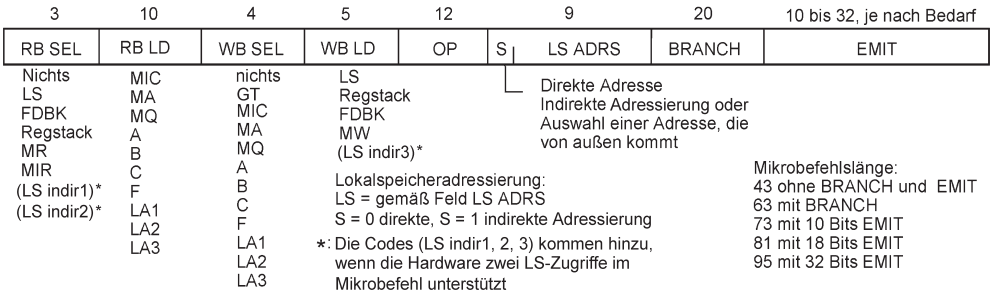


Abb. 6.74 Ein horizontales Mikrobefehlsformat für die Maschine der Abb. 6.68 und 6.69. RB = Lesebus, WB = Schreibbus. Zu den Codes (LS indir1,2 3) s. Seite 397.

Jedes der beiden Bussysteme RB, WB hat zwei Felder, eines, das auswählt, welche Einrichtung aufschaltet und eines, das anweist, welche Einrichtungen geladen werden. Da nur eine Einrichtung aufschalten darf, ist das erste Feld (RB SEL, WB SEL) binär codiert. Das zweite Feld (RB LD, WB LD) hat hingegen für jede Einrichtung, die geladen werden kann, ein einzelnes Erlaubnisbit. Somit können die Einrichtungen in beliebiger Kombination gleichzeitig geladen werden[68]. Für den Operationscode, die Lokalspeicheradressierung, die Verzweigung und den Direktwert wurden die Felder gegenüber dem 16-Bit-Format verlängert. Das Lokalspeicheradreßfeld hat ein Auswahlbit S (Select). Ist S = 0, so enthalten die restlichen 8 Bits die Lokalspeicheradresse. Ist S = 1, so wählen die restlichen 8 Bits eines der LS-Adreßregister (LA1, 2, 3) oder eine von außen einzuspeisende Adresse aus. Verzweigungsadresse und Direktwert können so lang gewählt werden wie zweckmäßig. Im Verzweigungsfeld (BRANCH) sind auch Unterprogrammruf und Unterbrechungssteuerung untergebracht. Das Mikrobefehlsformat paßt aber nicht wirklich gut zur Struktur der Maschine. Nicht alles, was man anweisen kann und was auch zweckmäßig wäre, kann die Maschine in einem einzigen Zyklus erledigen. Eine nützliche Operation wäre beispielsweise, eine Lokalspeicherinhalt mit einem Direktwert zu vergleichen und gemäß dem Vergleichsergebnis zu verzweigen[69]. Da Lokalspeicherinhalt und Direktwert über den gleichen Bus laufen, braucht man dazu aber mehr als einen Zyklus.

Kürzere horizontale Mikrobefehle

Es liegt nahe, in einem Mikrobefehl nur Abläufe anzuweisen, die auch tatsächlich in einem einzigen Taktzyklus ausgeführt werden können. Die Transporte und Operandenverknüpfungen kann man mit dem Verzweigen kombinieren. Die Maschine von Abb. 6.68 kann aber nur einen Operanden bereitstellen, entweder eine Direktwert oder einen Lokalspeicherinhalt (Abb. 6.75). Die Felder RB SEL bis BRANCH belegen 53 Bits. In ein 64-Bit-Format passen noch 11 Direktwert- oder Adreßbits. Mit 32 Bits Emit-Feld wäre der Mikrobefehl 85 Bits lang.

68. Hier könnte man ggf. sparen (binäre Codierung der einzelnen Register und der sinnvollen Kombinationen).
69. Z. B. eine Programmierabsicht der Art IF <LS21> = 3456H THEN GOTO XYZ.

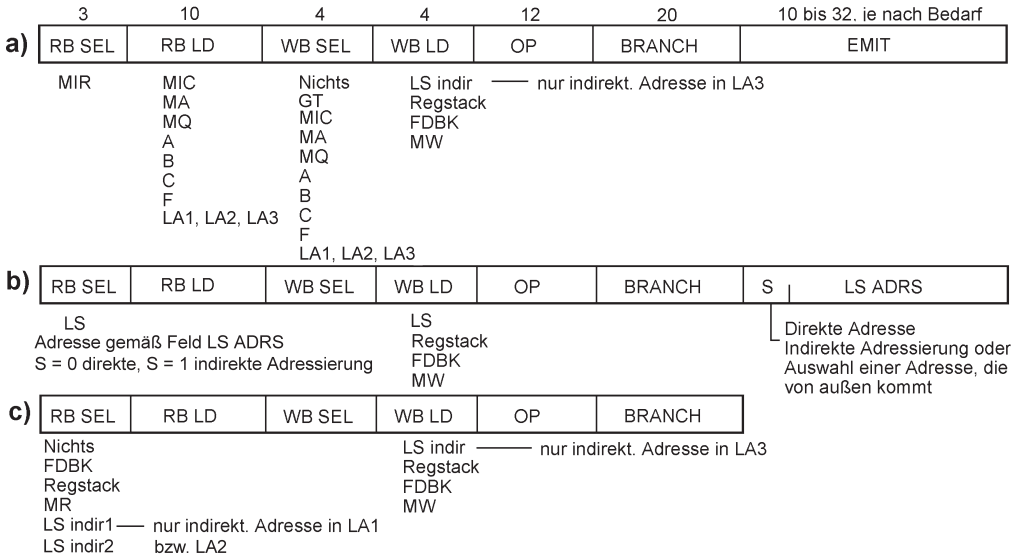


Abb. 6.75 Verkürzte horizontale ("diagonale") Mikrobefehlsformate unterschiedlicher Länge[70].

Die Maschine unterscheidet zwischen den drei Formaten, indem sie die Auswahlfelder RB SEL und WB SEL decodiert:

a) Direktwert, aber keine Lokalspeicheradresse. Der Direktwert aus dem Mikrobefehlsregister (MIR) wird auf den Lesebus geschaltet. Wird der Lokalspeicher zum Schreiben ausgewählt, kommt die Adresse aus dem Adreßregister LA3.
b) Lokalspeicheradresse, aber kein Direktwert. Der Lokalspeicherausgang wird auf den Lesebus geschaltet. Wenn S = 0, enthält das Feld LS ADRS die Lokalspeicheradresse, wenn S = 1, wählt das Feld LS ADRS die Adreßquelle aus (indirekte Adressierung oder Adresse von außen).
c) Weder Direktwert noch Lokalspeicheradresse. Deshalb können die Felder WB LD und LS ADRS entfallen. Der Mikrobefehl kann aber den Lokalspeicher implizit adressieren, beim Lesen mit der Adresse in LA1 oder LA2, beim Schreiben mit der Adresse in LA3.

Lokalspeicheradressierung und Lokalspeicherzugriffe

Die Maschine der Abb. 6.68 und 6.69 wurde zunächst für einfache vertikale Mikrobefehle konzipiert. In einem Mikrobefehlszyklus kann sie nur einen Lokalspeicherzugriff ausführen. In den horizontalen Mikrobefehlen hat aber jeder Bus eigene Felder. Damit kann man anweisen, den Lokalspeicher zu lesen (Feld RB SEL) und den Inhalt in ein Zielregister zu laden (Feld RB LD),

70. Es liegt nahe, alle Formate auf gleiche Länge zu bringen und die ungenutzten Bits für untergeordnete Zwecke zu verwenden (Diagnose, Debugging, E-A usw.).

sowie ein Register auf den Schreibbus aufzuschalten (Feld WB SEL) und den Registerinhalt in den Lokalspeicher zu schreiben. Man könnte auch in einem einzigen Mikrobefehl mit zwei Lokalspeicheradressen arbeiten. Ein solcher Ablauf würde zwei Taktzyklen erfordern, der erste zum Lesen, der zweite zum Schreiben. Da der Mikrobefehl nur ein Adreßfeld hat, muß die andere Adresse aus einem Adreßregister kommen. In Abb. 6.74 sind dafür zusätzliche Codes in den Feldern RB SEL und WB LD angegeben (LS indir1, 2, 3). Beim Lesen kommt die Adresse aus dem Adreßregister LA1 oder LA2, beim Schreiben aus LA3. Die zusätzlichen Codes wären sinngemäß in den Feldern RB SEL und WB LD des Formats von Abb. 6.75b vorzusehen.

Erweiterungen und alternative Mikrobefehlsformate

Die Grundsatzfrage ist, wie weit man mit der horizontalen Mikroprogrammierung überhaupt geht – in einer Maschine, die als sozusagen universell mikroprogrammierbar gedacht ist. Wenn man die Ausführung all dessen, was im Mikrobefehl angewiesen werden kann, tatsächlich steuern will, ergeben sich womöglich längere Taktzyklen mit zwei oder mehr Taktphasen. Auch stellt sich die Frage, ob eine solche Mikro-Architektur ein wirklich gutes Compiler-Ziel abgibt. Vielleicht sind nicht allzu lange, überschaubar formatierte Mikrobefehle, die in kurzen Taktzyklen ausgeführt werden, für eine universelle Maschine doch zweckmäßiger. Die Verarbeitungsleistung wäre dann vor allem über eine verbesserte und erweiterte Ressourcenausstattung zu gewinnen. Abb. 6.76 veranschaulicht eine Maschine gemäß Abb. 6.68, die mit einem Befehlsregister erweitert ist. Die Mikrobefehle sind zugleich die elementaren Maschinenbefehle. Die komplexeren Maschinenbefehle der zu emulierenden High-Level-Architektur sind im Grunde Funktionsaufrufe, deren Parameter im Befehlsregister übergeben werden. Für das Mikroprogramm ist ein Maschinenbefehl nichts anderes als ein elementarer Funktionsaufruf. Der Maschinenbefehl, der aus dem Speicher gelesen wird, übergibt die aktuellen Funktionsparameter. Die laden wir ins Befehlsregister, so daß sie sofort zur Verfügung stehen, ohne Lokalspeicherzugriffe o. dergl. Bezogen darauf, wie Funktionen, in höheren Programmiersprachen formuliert, in der Maschine typischerweise aufgerufen werden, enthält das Befehlsregister im Grunde den Stack Frame des aktuellen elementarsten Funktionsaufrufs.

Erweiterte vertikale Mikrobefehle

Um die Maschine von Abb. 6.76 zu steuern, sind die Felder in den 16-Bit-Mikrobefehlen gemäß Abb. 6.71 etwas knapp. Auch liegt es nahe, an erweiterte Verarbeitungsressourcen zu denken, die zusätzliche Register aufweisen. Abb. 6.77 zeigt, daß bereits wenige Bits ausreichen, um genug Platz zu schaffen, aber das Prinzip der einfachen Maschine beizubehalten. Hier wurden die Mikrobefehle auf 18 Bits verlängert[71]. Das Registerauswahlfeld ist nun 5 Bits lang, so daß bis zu 32 Register angesprochen werden können[72]. Das Lokalspeicheradreßfeld wurde um ein Auswahlbit S (Select) erweitert. Ist dieses Bit gesetzt, wählt das Adreßfeld weitere Adreßquel-

71. Ja, das Format ist kein aktueller Industriestandard. Das Problem ist aber lösbar. Vgl. Abschnitt 8.1, S. 441 bis 444.
72. Damit könnte man u. a. in der Verarbeitungseinheit die zusätzlichen Register AUX und CT unterstützen. Vgl. Anhang 2, S. 547.

len aus, u. a. eines der drei Adreßregister LA1, 2, 3 oder eines der Registeradreßfelder im Befehlsregister (wie anhand von Abb. 6.64 erläutert[73]). Das Operationscodefeld und das Emit-Feld sind jeweils 9 Bits lang. Somit können bis zu 512 Operationen ausgewählt werden[74]. Der Direktwert im Emit-Feld kann als ein volles 8-Bit-Byte mit gesondert vorangestelltem Vorzeichen genutzt und durch Vorzeichenerweiterung auf die jeweilige Wortlänge gebracht werden[75]. In der bedingten Verzweigung (Branch) wurde das 8 Bits lange Displacement beibehalten, aber die Bedingungsauswahl auf insgesamt 6 Bits erweitert. Dem liegt die Überlegung zugrunde, daß es vor allem darauf ankommt, viele Bedingungen direkt – also ohne Voreinstellung – auswählen zu können. Die Vergrößerung der Sprungweite ist demgegenüber weniger von Bedeutung.

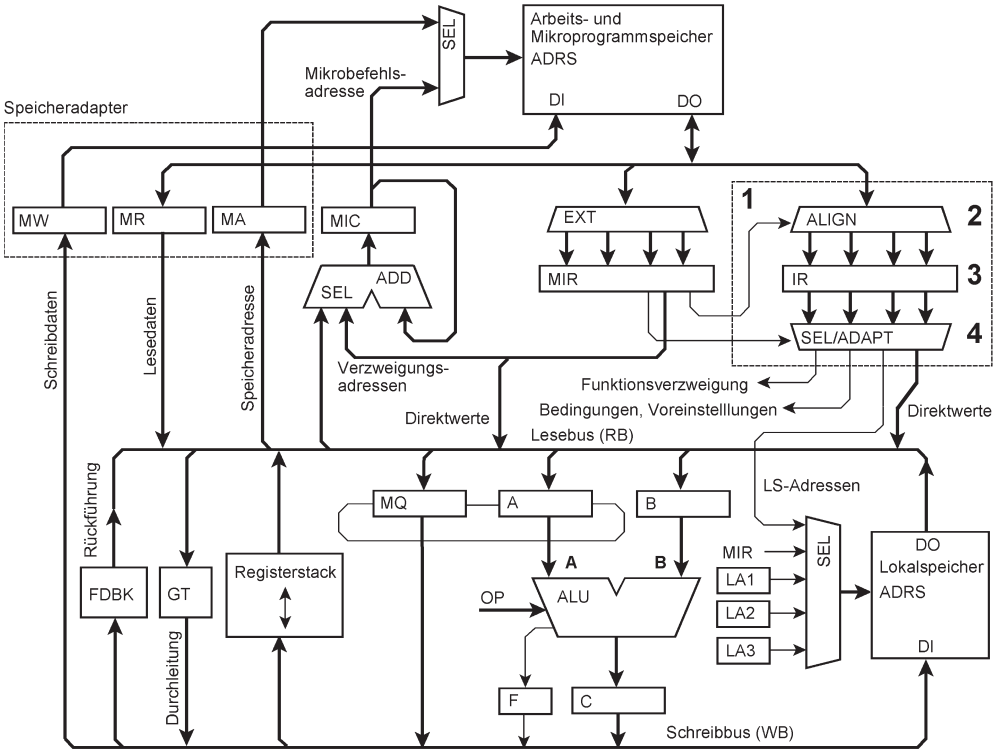


Abb. 6.76 Eine erweiterte einfache Maschine. 1 - dieser Funktionsblock ist anwendungsspezifisch zu synthetisieren; 2 - Befehlsformatzuordner; 3 - Befehlsregister; 4 - Auswahl- und Adapterschaltungen.

73. S. 383f. Als historisches Beispiel vgl. das Transform Feature der CDC 5600 (S. 517ff).
74. Ein so langes Operationscodefeld eignet sich auch zur analytischen Codierung mit einzelnen Steuerbits.
75. Ein solcher Direktwert besteht aus dem Byte des Emit-Feldes in den niedrigstwertigen acht Bitpositionen und aus Nullen oder Einsen in allen höherwertigen.

Das Laden des Befehlsregisters 3 wird vom Mikroprogramm gesteuert. Der Befehlsformatzuordner 2 ordnet die Befehlsbits so an, wie es dem Befehlsformat entspricht, unabhängig davon, wie der Befehl aus dem Speicher gelesen wurde[76]. Die Auswahl- und Adapterschaltungen 4 liefern die im Befehl codierten Parameter – erfoderlichenfalls aufbereitet oder umcodiert – an die jeweiligen Funktionseinheiten. Die Parameter können u.a. als Adreßteile der Funktionsverzweigung, als Voreinstellungen, als Lokalspeicheradressen oder als Direktwerte genutzt werden. Die Zuordner-, Auswahl- und Adapterschaltungen 2, 4 kann man auch aufgabenspezifisch entwerfen (Verhaltensbeschreibung) und als Schaltungsblöcke synthetisieren lassen. Vgl. auch die einschlägige Diskussion anhand von Abb. 6.61 (S. 378f).

17 16 15 14	13 12 11 10 9	8 7 6 5 4 3 2 1 0
LOAD	REGSEL	S LS ADRS
PUSH LOAD	REGSEL	S LS ADRS
LOAD RET	REGSEL	S LS ADRS
STORE	REGSEL	S LS ADRS
STORE RET	REGSEL	S LS ADRS
OPERATE	REGSEL	OPCODE
PUSH OPERATE	REGSEL	OPCODE
OPERATE RET	REGSEL	OPCODE
EMIT	REGSEL	EMIT
PUSH EMIT	REGSEL	EMIT
EMIT RET	REGSEL	EMIT
BRANCH	CONDSEL	BRANCH DISPLACEMENT
JUMP	JUMP ADRS	
CALL	CALL ADRS	
EXTEND	EMIT EXTEND	

Abb. 6.77 18-Bit-Mikrobefehle.

76. Man denke beispielsweise an einen Befehl aus 6 Bytes, der abschnittsweise in zwei Speicherwörtern untergebracht ist (z. B. 2 Bytes im ersten und die restlichen 4 Bytes im zweiten).