7. Der Prozessor als Mikroprogrammsteuerwerk

7.1 Eine alternative Design-Idee

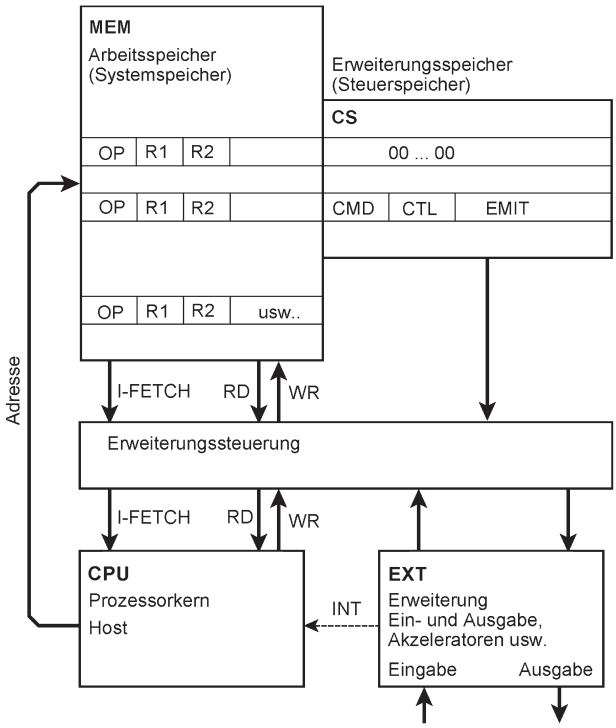
Wir kehren zum fertigen Universalprozessor zurück, bauen aber keinen eigenen, sondern bleiben bei bewährten Prozessoren und Entwicklungsumgebungen. Die meiste Software läuft schnell genug, da muß nichts beschleunigt werden. Um die verbleibenden, besonders leistungskritischen Programmstücke schneller auszuführen, werden die Befehle außerhalb des Prozessors verlängert. Der Prozessorkern selbst bleibt wie er ist. Im Innern wird nichts geändert. Dem Programmspeicher wird ein Erweiterungs- oder Steuerspeicher hinzugefügt. Er wird genauso adressiert wie der Speicher des Prozessors. Aus Sicht der Adressierung werden die Speicherwörter nur länger. Die Verlängerung wirkt aber nicht im Innern des Prozessors, sondern auf externe Steuerschaltungen. Dort löst sie zusätzliche Steuerwirkungen aus. Der Prozessorkern sieht nur Befehle, die in seiner Architektur spezifiziert sind. Sie haben aber zusätzliche Wirkungen außen, oder sie werden auf dem Wege vom Speicher zum Prozessorkern so modifiziert, daß sich andere Wirkungen ergeben. Damit können u. a. E-A-Funktionen und Beschleunigungsschaltungen (Akzeleratoren) unterstützt werden. Der Prozessor wird zeitweilig zum spezialisierten Mikroprogrammsteuerwerk. Die auf diese Weise ausgeführten Befehle werden außer-halb des Prozessorkerns zu Mikrobefehlen. Erweiterte und gewöhnliche Befehle können beliebig gemischt werden. Die Befehle des Prozessorkerns können um soviele Bits verlängert werden wie erforderlich. Alle Programme, die mit diesen Funktionen nichts zu tun haben, bleiben wie sie sind. Im Gegensatz zu einem mikroprogrammgesteuerten Prozessor eigener Architektur verhindert die Erweiterung nicht die Nutzung vorhandener Software und Entwicklungssysteme. Der Prozessor, der erweitert wird, ist der Host, die Befehle, die er aus dem Speicher liest, um sie auszuführen, ausführt, sind die Host-Befehle[1].

Beispielsweise bekommt der Prozessor anstelle des Befehls aus dem Speicher einen NOP geliefert, wenn bestimmte Bedingungen nicht erfüllt sind (Überspringen auf externe Bedingungen). Auch können auf diesem Wege Akzeleratorschaltungen ohne programmseitigen Overhead eingebunden werden. So ergibt sich ein enge Verbindung zwischen dem (fertigen) Prozessorkern (IP Core) und der Spezialhardware (Akzelerator, Coprozessor). Eine entsprechende Befehlserweiterung zeigt beispielsweise an, daß Daten, die für ein Register des Prozessors bestimmt sind, auch in ein Register des Akzelerators eingetragen werden oder daß ein Register nicht mit dem adressierten Speicherinhalt, sondern mit einem Ergebnis aus dem Akzelerator geladen wird.

Es ist praktisch eine Vereinigung der üblichen Befehlswirkungen und der Mikroprogrammierung. Da auch die speziellen Programme im Befehlsstrom des Prozessors ausgeführt werden, entfällt der Overhead, der mit der Nutzung externer Steuerwerke typischerweise verbunden ist (Auftragserteilung, Parameterübergabe, Warten auf Fertigmeldungen und Ergebnisse).

1. Diese Bezeichungen sind kurz, ihr Sinn ist intuitiv verständlich. Sie werden hier eingeführt, um sie in den folgenden Schaltungs- und Funktionsbeschreibungen zu verwenden.

Die Beschleunigung (Speedup) liegt im Bereich von 2:1 bis etwa 20:1 (Richtwerte). Die Abb. 7.1 bis 7.4 veranschaulichen das Prinzip.



I-FETCH	Befehlslesen (Instruction Fetch)
RD	Daten lesen (Read)
WR	Daten schreiben (Write)
INT	Unterbrechungsauslösung (Interrupt)

Abb. 7.1 Befehlserweiterung mittels Erweiterungsspeicher (Steuerspeicher). Die zusätzlichen Funktionen (hier angedeutet mit einem Kommandocode CMD, einem Steueranweisungsfeld CTL und einem Direktwert EMIT) werden nur ausgeführt, wenn der Befehl aus einem Speicherbereich gelesen wird, dem ein Erweiterungsspeicher zugeordnet ist. Enthält der Erweiterungsspeicher an der aktuellen Befehlsadresse nur Nullen[2], wirkt der Befehl ohne erweiterte Funktionen.

Herkömmlicherweise werden Schaltungen, die den Prozessorkern ergänzen, wie E-A-Einrichtungen oder Akzeleratoren, mit besonderen Befehlen gesteuert (Abb. 7.2a). Sofern die Unterstützung nicht von Grund auf in die Maschinenarchitektur eingebaut ist (wie bei manchen Coprozessoren), sind es typischerweise Transport- oder E-A-Befehle. In der alternativen

2. Oder – alternativ – einen NOP-Kommandocode.

Lösung von Abb. 7.1 werden Maschinenbefehle außerhalb des Prozessors verlängert, so daß sie wie Mikrobefehle wirken (Abb. 7.2b). Im einfachsten Fall üben sie Steuerwirkungen aus, die mit der Befehlsausführung im Prozessorkern nichts zu tun haben. Im allgemeinen Fall wird eine Erweiterungssteuerung in die Speicherschnittstelle des Prozessorkerns eingefügt, um dort Daten abzugreifen und einzuspeisen.

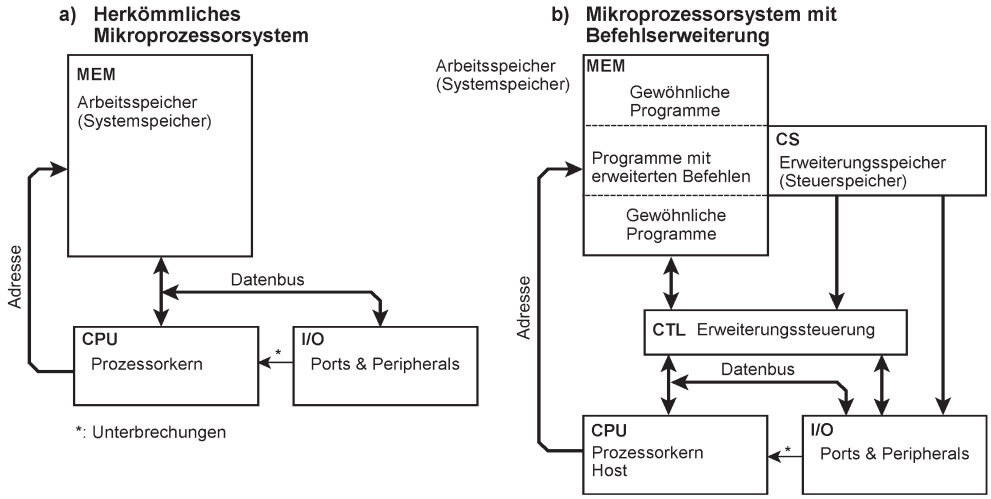


Abb. 7.2 Herkömmliche und erweiterte Mikroprozessorsysteme.

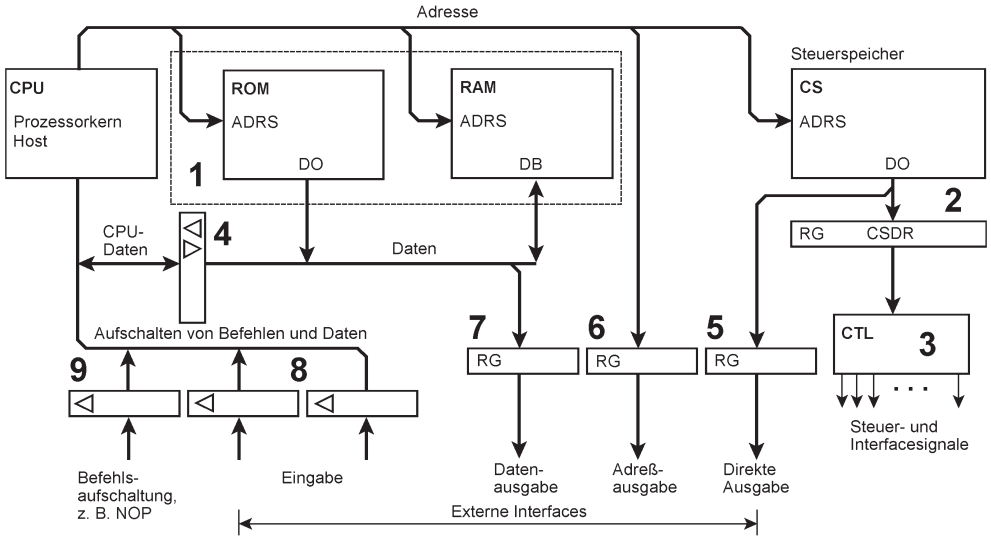
Der nicht erweiterte Mikroprozessor (a) kann die E-A-Einrichtungen zumeist nur mit Zugriffs- oder Transportbefehlen ansprechen. Alles, was komplexer ist, muß ausprogrammiert werden. *Einlesen – Auswerten – Verzweigen* und *Parameter ausgeben – Berechnen – Ergebnisse einlesen* sind typische Programmabläufe. Die Erweiterung (b) ermöglicht es hingegen, daß Maschinenbefehle direkt mit der Ein- und Ausgabe zusammenwirken und daß Abläufe, die herkömmlicherweise mehrere Befehle erfordern, mit einem einzigen Befehl erledigt werden.

Beispiele erweiterter Befehlswirkungen:

- Unterbrechungen zeitweilig verhindern,
- Daten ausgeben,
- Zusatzschaltungen aktivieren, beispielsweise Beschleunigungseinheiten (Akzeleratoren) oder anwendungsspezifische Peripherie,
- auf Bits der Ein- und Ausgabe verzweigen,
- Befehle in Abhängigkeit von Bedingungen ausführen (Predication),
- alternative Befehle ausführen,
- das Befehlslesen nutzen, um Daten auszugeben (Befehlsausgabe),
- Zusatzeingabe
- Funktionsverzweigung,

- EXECUTE-Befehle,
- die vom Prozessor gelieferten Adressen als zusätzliche Ausgabedaten verwenden,
- Zustandsübergänge universeller Steuerautomaten unterstützen,
- den Start-Stop-Betrieb unterstützen.

Abb. 7.3 zeigt, wie sich der Steuerspeicher und die ergänzenden Schaltungen in ein herkömmliches Mikroprozessorsystem einfügen. Der Steuerspeicher wird beim Befehlslesen genauso adressiert wie die herkömmliche Speicherausstattung 1. Dabei wird die Erweiterung – im Grunde ein zusätzlicher Mikrobefehl – ins Steuerregister (CSDR) 2 geladen. Die Erweiterungssteuerung 3 erregt Steuersignale, um Daten oder Adressen in Ausgaberegister zu laden und eingegebene Werte oder alternative Befehlscodes auf den Datenbus aufzuschalten.



1 Die herkömmliche Speicherausstattung des Host
2 Steuerregister
3 Erweiterungssteuerung
4 Datenbuspuffer. Trennt den Datenbus der Speicher 1 vom Prozessorbus ab, wenn Daten oder Befehlscodes von außen aufgeschaltet werden
5 Direkte Ausgabe. Die auszugebenden Bitmuster stammen aus der Erweiterung
6 Adreßausgabe. Die während der Ausführung des aktuellen Befehls gelieferte Datenadresse wird als Ausgabebitmuster verwendet
7 Datenausgabe. Die auszugebenden Bitmuster kommen vom Datenbus
8 Eingangsaufschaltung. Eingangssignale werden auf den Datenbus aufgeschaltet
9 Befehlsaufschaltung. Anstelle des im Speicher adressierten Host-Befehls wird dem Prozessor ein anderer Befehlscode zugeführt. Ein typisches Beispiel ist ein NOP, der bewirkt, daß die Wirkung des Host-Befehls übergangen wird

Abb. 7.3 Ein herkömmliches Mikroprozessorsystem (Host) mit Erweiterung.

Speicheradressierung und Speicherzugriffe
Es soll alles überschaubar bleiben. Wir wollen nur in die Datenwege eingreifen, nicht in die Adressierung des Speichers und in die Art der Speicherzugriffe. Der Speicherzugriff wird stets so ausgeführt, wie ihn der Maschinenbefehl im Host angewiesen hat. Der Prozessorkern sieht nur ein Speichersubsystem, das gelegentlich – abhängig von der Art der Erweiterung und der Schaltungsauslegung – eine etwas längerer Zugriffszeit aufweist oder Wartezustände einfügt.

Den Speicher ruhigstellen
Manche Erweiterungen erfordern, daß keine Schreibzugriffe ausgeführt werden, daß die Adresse nicht decodiert wird, daß Fehlersignale nicht ausgewertet werden usw. Dem Prozessorkern gegenüber muß man aber so tun, als würde der Speicherzugriff normal ablaufen[3].

Der Prozessorkern wird zeitweilig zum Mikroprogrammsteuerwerk
Der erweiterte Befehl wird zum Mikrobefehl (Abb. 7.4). Alle Programme, die mit gewöhnlichen Befehlen auskommen, können mit beliebigen Entwicklungswerkzeugen erstellt werden; Anwendungslösungen kann man so entwickeln, wie es in der jeweiligen Prozessorfamilie oder Systemarchitektur üblich ist. Auch wird die Lauffähigkeit beliebiger anderer Software nicht beschränkt. Das Schreiben von Programmen, die die Mikrobefehlsfunktionen ausnutzen, entspricht dem Entwickeln von hardware-naher Gerätetreiber- und Steuerungssoftware.

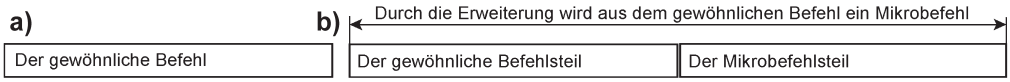


Abb. 7.4 Gewöhnliche Befehle (a) werden zu Mikrobefehlen erweitert (b).

Gut und weniger gut geeignete Prozessorkerne
Wir arbeiten mit Prozessorkernen mittleren Leistungsvermögens – keine spekulative Befehlsausführung, keine Sprungvorhersage, kein virtueller Speicher. Der Prozessor muß die Befehle so ausführen, wie er sie aus dem Speicher holt. Um Befehle zu modifizieren oder zu ersetzen, werden wir auf dem Befehlsleseweg vom Speicher oder Cache zum Befehlsregister des Prozessors eingreifen. Das Prinzip funktioniert mit allen Prozessoren, die beim Befehlslesen (Instruction Fetch) nichts zwischenpuffern[4]. Prozessorkerne mit interner Befehlspufferung (wie 8086) oder eingebauten, nicht umgehbaren Caches, mit tiefen Pipelines und spekulativer Befehlsausführung scheiden aus. Solche Maschinen sind ohnehin keine wirklich guten E-A-Prozessoren. Manche Beschleunigungsmaßnahmen muß man abschalten, wenn man E-A-Zugriffe ausführen möchte. So sind die Caches und der virtuelle Speicher zu umgehen; die zur Ein- und Ausgabe genutzten Adreßbereiche müssen als Non-Cacheable Regions konfiguriert werden usw.

3. Am einfachsten wäre es, den Speicheradapter so abzuwandeln, daß er in solchen Fällen gar keine Speicheranforderungen stellt, aber dem Host gegenüber so tut, als ob er einen Zugriff ausführt. Das gelingt aber nicht immer.
4. Mit herkömmlichen 8-Bit-Mikroprozessoren funktioniert es ([MD27]). Mit Soft Cores sollte es sich einrichten lassen. Mit ARM, RISC V usw. dürfte man es hinbekommen, sofern man einen geeigneten Prozessorkern aussucht.

Prozessorkerne abwandeln und selbst entwerfen

Die Entwicklungen, die hier beschrieben werden, hatten ursprünglich das Ziel, Prozessoren zu erweitern, an denen gar nichts geändert werden kann, wie beispielsweise Mikroprozessorschaltkreise. Nun gehört es zum Stand der Technik, die Schaltungsentwickung auf IP Cores zu stützen. Es liegt somit nahe, einen Prozessorkern als IP Core auszuwählen, passend zuzurichten und dabei auch die interne Logik abzuwandeln. U. a. könnte man die spekulative Befehlsausführung, Sprungvorhersage usw. zulassen, wenn die Befehle aus Speicherbereichen gelesen werden, denen kein Erweiterungsspeicher zugeordnet ist. Auch könnte man sich Erweiterungswirkungen ausdenken, die die Befehle selbst betreffen, wie neue Operationen, alternative Registersätze, von außen eingespeiste Bedingungen, Wartezustände usw.

Darüber hinaus könnte man daran denken, solche Maschinen von Grund auf zu entwerfen. Sie implementieren eine bewährte, weit verbreitete Architektur, können aber auch Mikroprogramme ausführen, und zwar sozusagen live mitten im Befehlsstrom, also ohne den Overhead, der mit dem Aufrufen mikroprogrammierter Unterstützungsfunktionen (Microprogrammed Assists) oder der Nutzung von Akzeleratoren verbunden ist. Wenn man den Prozessorkern von Grund auf entwirft, könnte man auch die Mikrobefehle in den Formaten der üblichen Befehle unterbringen. Der Erweiterungsspeicher verkürzt sich damit zu einem Kennzeichenspeicher, in dem u. a. die Unterscheidung zwischen gewöhnlichen Maschinenbefehlen und Mikrobefehlen codiert ist (Abb. 7.5).

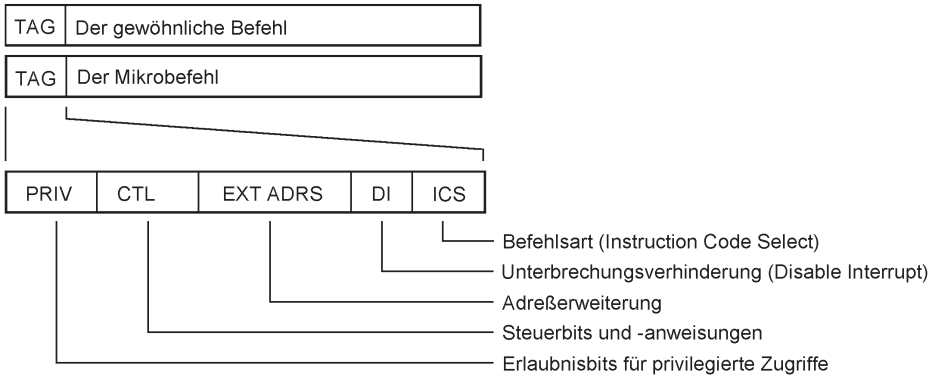


Abb. 7.5 Wenn wir den Prozessorkern von Grund auf entwerfen, können wir die meisten Mikrobefehlsfelder im Format der Maschinenbefehle unterbringen. Zusätzliche Kennzeichenbits (TAG-Bits) im Erweiterungsspeicher unterscheiden zwischen Maschinen- und Mikrobefehlen.

Der Erweiterungsspeicher enthält zudem die Steuerbits und Felder, die nicht in das vorgegebene Maschinenbefehlsformat passen. Das betrifft beispielsweise die Unterbrechungssteuerung, die Registeradressierung in den peripheren Funktionseinheiten und Erlaubniscodes für privilegierte Zugriffe.

Der Grundgedanke: Alles das außen unterbringen, was über die gegebene Host-Architektur hinausgeht, also nicht kompatibel ist. Die herkömmliche Lösung, einen Prozessor mit inkompatiblen Erweiterungen auszurüsten, besteht darin, eine Art Coprozessor oder Akzelerator zu entwerfen. Die Nutzung solcher Einrichtungen ist aber mit einem merklichen Overhead verbunden. Verlängerte Befehle hingegen wirken so, als würden sie seit jeher zur Maschine gehören.

Grundsätzliche Erweiterungswirkungen
Wir bleiben beim vorgegebenen Prozessorkern, an dem wir nichts ändern. Die Mikrobefehle werden zusätzlich zu den Maschinenbefehlen aus dem Erweiterungsspeicher geholt. Ihre Wirkungen unterscheiden sich darin, wie sie die gleichzeitig gelesenen Maschinenbefehle beeinflussen:

- Nebenläufige Wirkungen. Es gibt gar keine Beeinflussung. Die Mikrobefehle sind von den Maschinenbefehlen unabhängig.
- Nebenläufige Ausgabe. Die Mikrobefehle nutzen die Datenzugriffe der Maschinenbefehle aus, um Ausgabedaten zu gewinnen. Die Befehlsausführung des Prozessors wird nicht beeinflußt. Daten und Adressen der Speicherzugriffe werden gleichsam nebenher abgegriffen und als Ausgabedaten verwendet. Schreibdaten und Adressen kommen aus dem Prozessor, Lesedaten aus dem Speicher.
- Dateneinspeisung. Die Mikrobefehle wirken mit den Datenzugriffen der Maschinenbefehle zusammen, indem sie veranlassen, Daten von außen einzuspeisen. Beim Lesen gelangen die Daten von außen in den Prozessor, beim Schreiben in den Speicher.
- Befehlsmodifikation. Die Mikrobefehle verändern oder ersetzen den Befehl, den der Prozessor zu sehen bekommt (Erweiterungen mit Befehlsveränderung). Der Prozessorkern adressiert einen Befehl im Speicher, erhält aber einen veränderten oder grundsätzlich anderen Befehl von außen.
- Ergänzende Wirkungen. Sie lassen sich dann implementieren, wenn man in den Prozessorkern eingreifen darf oder gar einen eigenen Prozessor entwirft. Über die Erweiterungen wird all das eingeführt, was zur gegebenen Host-Architektur nicht kompatibel ist.

Grundabläufe der erweiterten Ein- und Ausgabe:

- Eingabe in den Prozessor (Laden eines Registers, Operation mit einem Operanden aus der Außenwelt): Ein Lade- oder Operationsbefehl wird erweitert. Beim Datenzugriff werden die Eingabedaten in den Lesedatenweg eingespeist.
- Eingabe in den Speicher: Ein Speicherbefehl oder ein Operationsbefehl mit Ergebnisspeicherung wird erweitert. Beim Datenzugriff werden die Eingabedaten in den Schreibdatenweg eingespeist.
- Ausgabe aus dem Prozessor: Ein Speicherbefehl oder ein Operationsbefehl mit Ergebnisspeicherung wird erweitert. Beim Datenzugriff werden die Ausgabedaten vom Schreibdatenweg abgegriffen. Wenn es möglich ist, Schreibzugriffe zu unterdrücken, kann man auch noch die Adresse ausgeben.

- Ausgabe aus dem Speicher: Ein Lade- oder Operationsbefehl wird erweitert. Beim Datenzugriff werden die Daten vom Lesedatenweg abgegriffen.
- Ausgabe aus dem Befehl (Direktwert): Beim Befehlslesen wird der Direktwert vom Lesedatenweg abgegriffen.

Im folgenden soll eine Auswahl an Zusatzfunktionen vorgestellt werden. Die Wirkprinzipien und Schaltungen sind so einfach, daß man sie bequem mittels Verhaltensbeschreibung erfassen kann. Deshalb können wir uns auf einige knappe Skizzen und Erläuterungen beschränken.

7.2 Nebenläufige Wirkungen

Der Host führt die aus dem Speicher gelesenen Maschinenbefehle unverändert aus. Die zusätzlichen Wirkungen werden von der Erweiterungssteuerung allein veranlaßt (Abb. 7.6 bis 7.8).

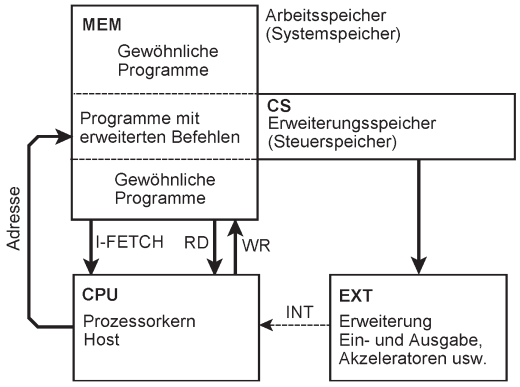


Abb. 7.6 Erweiterter Prozessor mit nebenläufigen Wirkungen. Der Steuerspeicher liefert einen Mikrobefehl, der in den zusätzlichen Schaltungen zur Wirkung kommt.

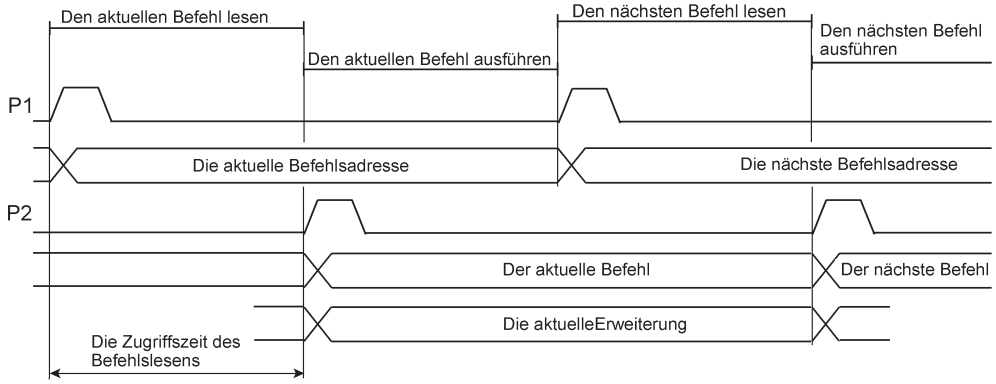


Abb. 7.7 Der einfachste Befehlsablauf.

Diese Darstellung wurde der Überschaubarkeit wegen gewählt. Alles wird nacheinander erledigt, nicht gleichzeitig. Keine mit der Befehlsausführung überlappte Befehlsadressierung, kein Pipelining, kein Einphasentakt. Der aktuelle Befehl wird gelesen (Phase P1) und dann ausgeführt (Phase P2). Die Erweiterung wird zugleich mit dem Befehl gelesen. Es ist praktisch ein ergänzender Mikrobefehl, der parallel zum Maschinenbefehl ausgeführt wird.

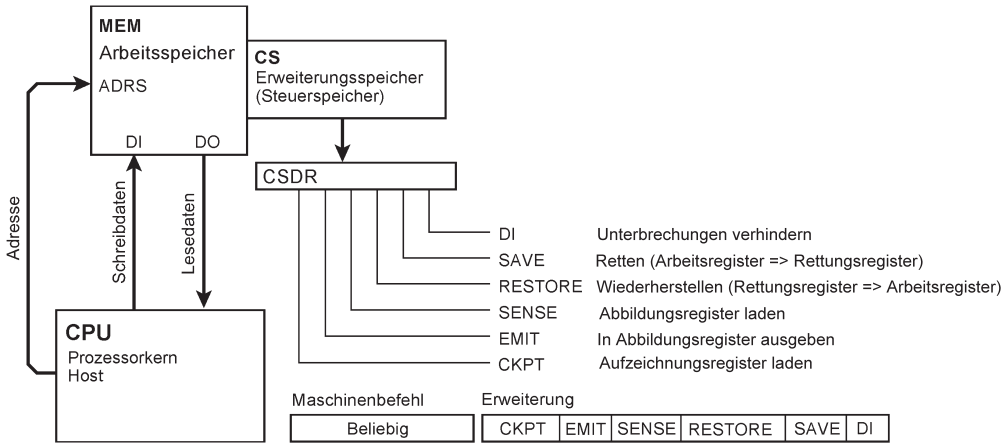


Abb. 7.8 Beispiele nebenläufiger Wirkungen. Näheres in den Abschnitten 7.2.1 und 7.2.2. Die Maschinenbefehle des Prozessors sind davon nicht betroffen.

7.2.1 Unterbrechungen zeitweilig verhindern

Impulse und Signalspiele werden oftmals programmseitig implementiert. Beispielsweise erzeugt eine Befehlsfolge der Art *Bit setzen – Zeit abwarten – Bit löschen* einen Impuls, dessen Dauer sich aus den Ausführungszeiten der Befehle ergibt. Werden nun zwischen diesen Befehlen Unterbrechungen wirksam, kann sich die Impulsdauer unvorhersagbar verlängern; aus wenigen hundert Nanosekunden können viele Millisekunden werden. Eine Lösung besteht darin, die Unterbrechungen zu Beginn der Befehlsfolge zu verhindern (DI-Befehl) und am Ende wieder zuzulassen (EI-Befehl)[5]. Wird der Ablauf aber in einem Programm ausgeführt, das nicht unterbrechbar sein darf, so ist der Erlaubnisbefehl (EI) falsch. Es können dann Unterbrechungen zur unrechten Zeit wirksam werden und zu Fehlern im Programmablauf führen (Absturz). Um dieses Problem zu lösen, wird eine Bitposition im Erweiterungsspeicher genutzt. Damit wird die Unterbrechungsauslösung im Prozessor zeitweilig verhindert (Abb. 7.9). Diese Sperre darf die eingebaute Erlaubnissteuerung (beispielsweise ein Erlaubnisbit im Bedingungsregister (Interrupt Flag IF) nicht beeinflussen.

5. DI = Disable Interrupt, EI = Enable Interrupt. Das sind typische Mnemonics. Solche Befehle gibt es in allen Maschinen, die ein Unterbrechungssystem aufweisen.

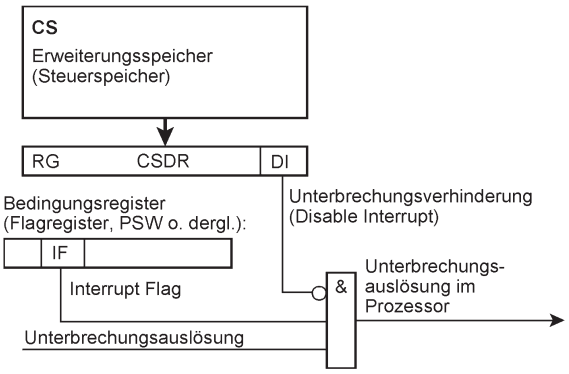


Abb. 7.9 Zeitweilige Unterbrechungsverhinderung. Sie kostet keine zusätzlichen Taktzyklen und kann den Maschinenbefehlen selektiv (einzeln) zugewiesen werden.

7.2.2 Register nebenher laden

Das Retten von Information und das Übernehmen von Daten aus der Außenwelt ist gelegentlich eine spitzfindige Angelegenheit. Herkömmlicherweise muß es anwendungsspezifisch ausprogrammiert werden. Die Alternative besteht darin, das Laden mit Steuerbits oder Anweisungen im Steuerwort auszulösen. Die Abb. 7.10 und 7.11 zeigen, wie die in Abb. 7.8 dargestellten Steuerbits genutzt werden können, um Register sozusagen nebenher zu laden, während ein beliebiger Maschinenbefehl ausgeführt wird. Typische Einsatzfälle sind Arbeits- und Rettungsregister, Abbildungsregister an den Ein- und Ausgängen sowie Aufzeichnungsregister (History Buffers o. dergl.) zum Debugging oder zur Fehlerbehandlung. Hier besteht der Vorteil der Befehlserweiterung darin, daß die Aufzeichnung keine zusätzlichen Taktzyklen kostet, also das Realzeitverhalten der Maschine nicht beeinflußt.

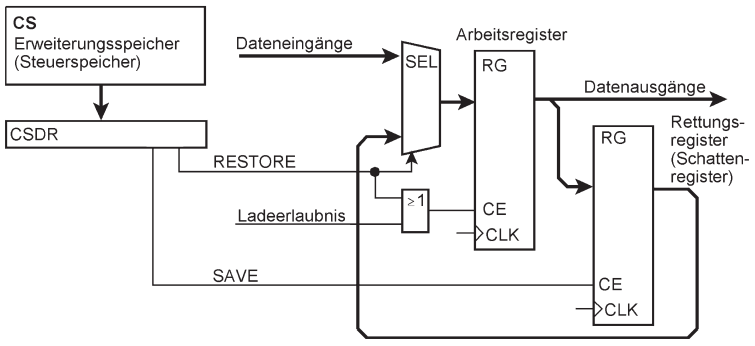


Abb. 7.10 Arbeitsregister und Rettungsregister (vgl. Abb. 5.89, S. 298). Die SAVE-Anweisung bewirkt, daß der Inhalt des Arbeitsregisters ins Rettungsregister eingetragen wird. Die RESTORE-Anweisung schreibt ihn zurück.

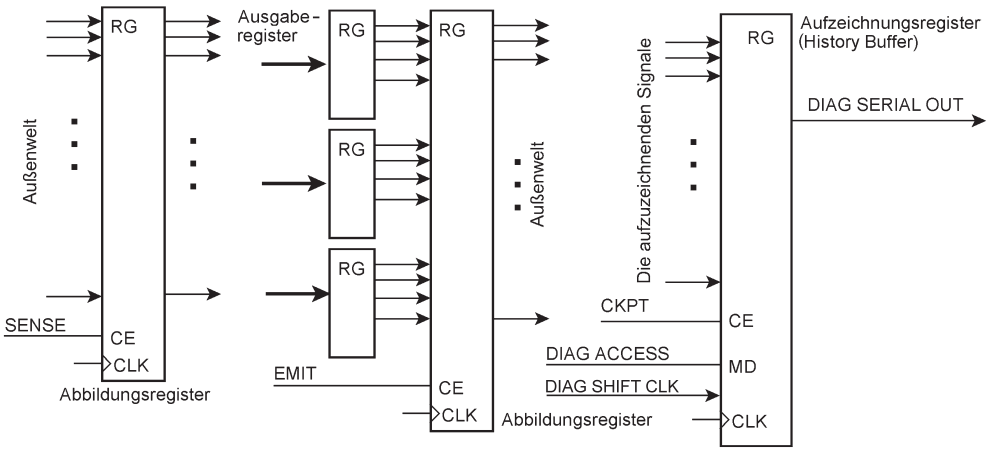


Abb. 7.11 Abbildungs- und Aufzeichnungsregister. Steuersignale der Anweisungen s. Abb. 7.8. Die SENSE-Anweisung veranlaßt die Übernahme von Daten aus der Außenwelt, die EMIT-Anweisung bewirkt, daß die Inhalte verschiedener Ausgaberegister auf einmal in der Außenwelt wirksam werden. Die CKPT-Anweisung (Checkpoint) lädt Signalwerte in ein Aufzeichnungsregister. Dessen Inhalt wird hier seriell ausgeschoben.

7.2.3 Zusatzausgabe

Parallel zur jeweiligen Befehlswirkung im Prozessor wird ein Bitmuster aus dem Erweiterungsspeicher ausgegeben (Abb. 7.12).

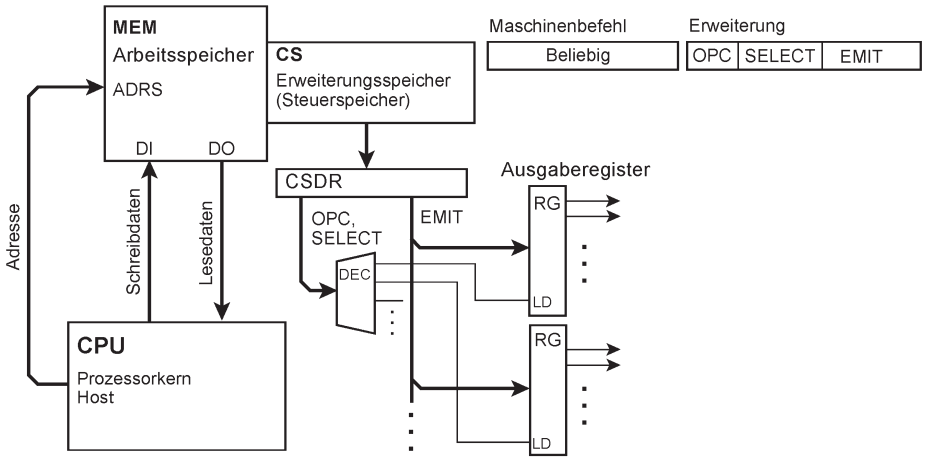


Abb. 7.12 Zusatzausgabe. Die Signale kommen aus dem Steuerspeicher. Sie können einen beliebigen Maschinenbefehl begleiten. OPC = Operationscode der Erweiterung; SELECT = Auswahl des Ausgaberegisters; EMIT = Direktwert (Emit-Feld).

7.3 Nebenläufige Ausgabe

Der Host führt die aus dem Speicher gelesenen Maschinenbefehle unverändert aus. Die zusätzlichen Wirkungen bestehen u. a. darin, Bits auf den Speicherdatenwegen abzugreifen und als Ausgabedaten zu verwenden (Abb. 7.13 und 7.14).

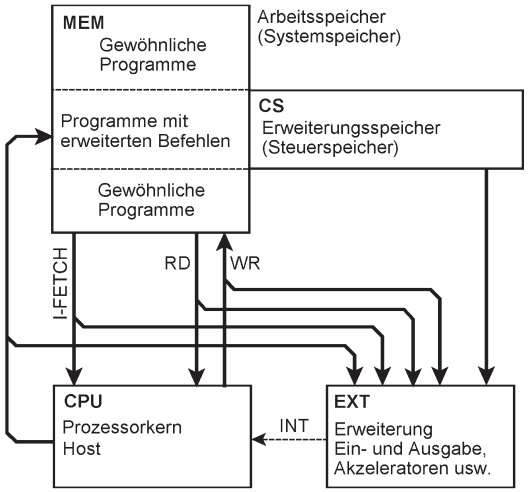


Abb. 7.13 Erweiterter Prozessor mit nebenläufiger Ausgabe. Die Daten- und Adreßwege der Speicherschnittstelle werden außen mitgelesen.

Mit Lesezugriffen kann man Daten aus dem Speicher ausgeben, mit Schreibzugriffen Daten aus dem Prozessor. Dabei ist, falls erforderlich, ein blindes Lesen oder Schreiben zu programmieren, also das Laden eines Prozessorregisters, das eigens dafür reserviert wurde oder das Schreiben auf eine entsprechend reservierte Speicheradresse. Wenn es möglich ist, die Schreibzugriffe zum Speicher zu unterdrücken, könnte man auch die Adresse ausgeben. Sinngemäß ist es möglich, Befehlsbits außen mitzulesen, um Direktwerte zu gewinnen, die ausgegeben oder in den Erweiterungsschaltungen zu Auswahl- und Steuerungszwecken verwendet werden. Abb. 7.14 veranschaulicht eine Befehlsablauf mit Datenzugriff. In diesem Beispiel wird ein Operand gelesen. Der aktuelle Befehl wird gelesen (Phase P1) und dann geladen und decodiert (Phase P2). Darauf folgt ein Datenlesezugriff (Phase P3), um den Operanden des Befehls zu lesen. Anschließend wird die Operation ausgeführt (Phase P4). Es ist offensichtlich nicht schwierig, beide Entwurfsaufgaben zu lösen, die Datenübernahme und die Dateneinspeisung (Abschnitt 7.4). Bei der Datenübernahme wird der Operand von den Speicherdatenwegen abgegriffen und in ein Ausgaberegister geladen, bei der Dateneinspeisung werden die Datenwege umgeschaltet, bevor die Phase des Operandenlesens beginnt.
In den folgenden Befehlsbeispielen bedeuten cpu_reg ein Prozessorregister, out_reg ein Ausgaberegister, in_reg ein Eingaberegister. Der senkrechte Strich (|) trennt den Maschinenbefehl (links) von den Mikroanweisungen der Erweiterung (rechts).

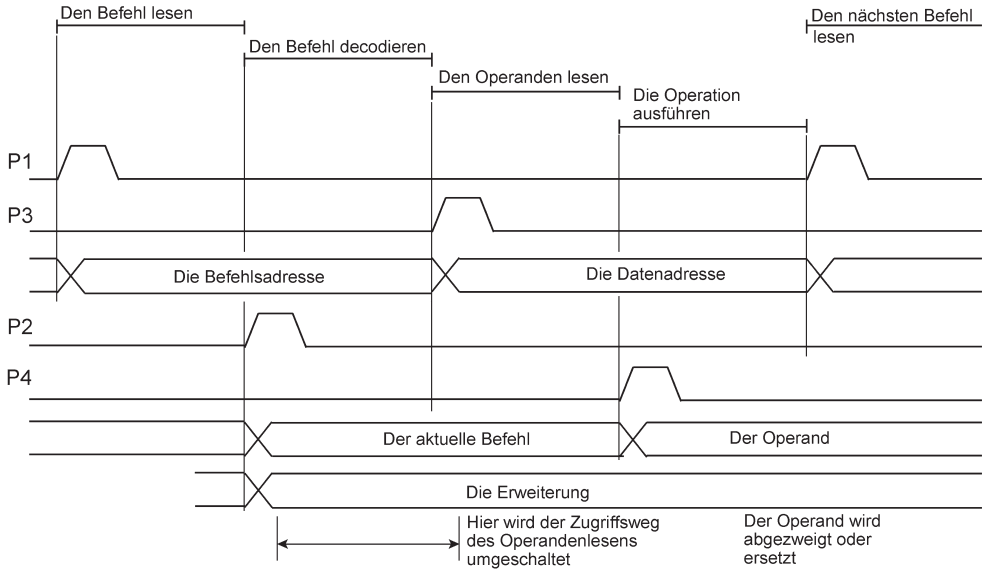


Abb. 7.14 Beispiel eines Befehlsablaufs mit Datenzugriff.

7.3.1 Datenausgabe

Die Daten werden zum Ausgeben von den Speicherdatenwegen gleichsam abgezweigt (Abb. 7.15 und 7.17).

Befehle mit Lesezugriff
Das Ausgaberegister wird mit Daten aus dem Speicher geladen. Befehlsbeispiel (Abb. 7.15a):

```
LD cpu_reg, Adresse | DAOR out_reg
```

Der adressierte Speicherinhalt wird ausgegeben.

Befehle mit Schreibzugriff
Das Ausgaberegister wird mit Daten aus dem Prozessor geladen. Dabei wird zugleich eine Kopie im Speicher angelegt. Wenn das nicht gewünscht ist, ggf. den Schreibzugriff unterdrükken. Befehlsbeispiel (Abb. 7.15b):

```
ST Adresse, cpu_reg | DAOW out_reg
```

7.3.2 Direktwertausgabe

Das Ausgaberegister wird mit einem Direktwert aus dem Befehl geladen. Befehlsbeispiel (Abb. 7.16 und 7.17):

```
LDI cpu_reg, Direktwert | DAOI out_reg
```

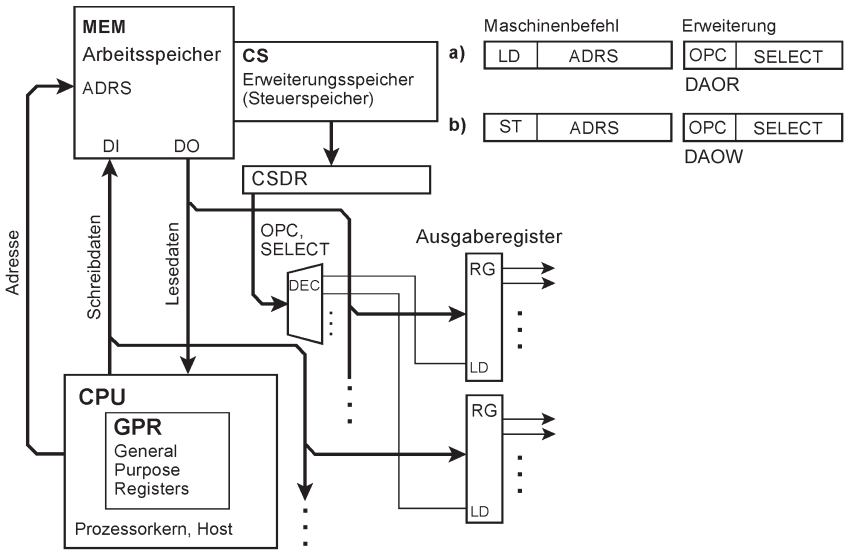


Abb. 7.15 Datenausgabe. OPC = Operationscode der Erweiterung; SELECT = Auswahl des Ausgaberegisters; DAOR = Data Output while Reading; DAOW = Data Output while Writing.

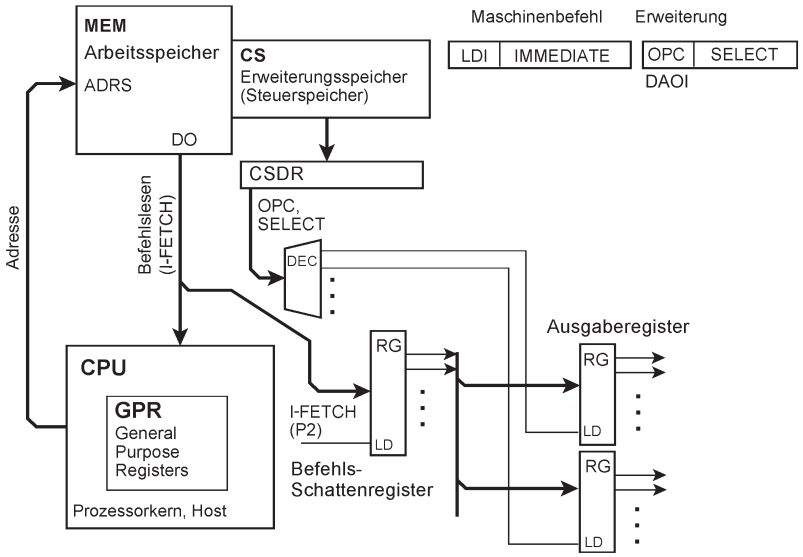


Abb. 7.16 Direktwertausgabe. Abhängig vom Taktraster der Befehlsabläufe kann es erforderlich sein, den aktuellen Befehl beim Befehlslesen (I-Fetch) in ein Schattenregister zu übernehmen. Das Ausgaberegister wird dann aus dem Schattenregister geladen. DAOI = Data Output Immediate. Vgl. auch Abb. 7.7.

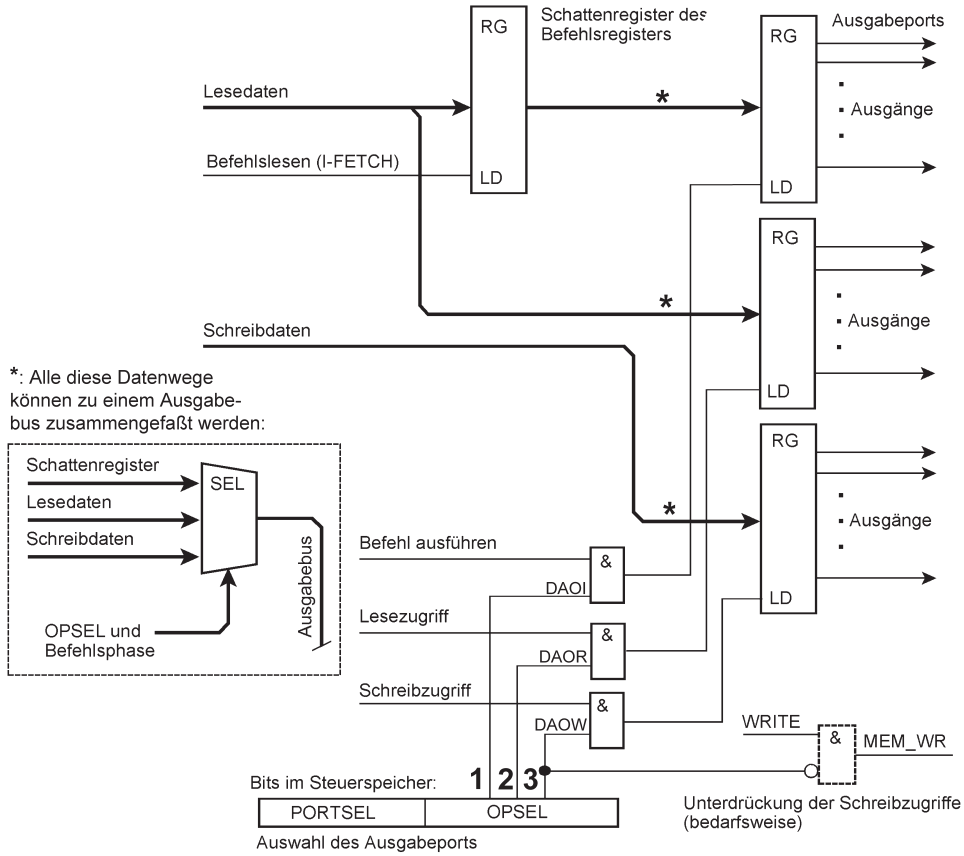


Abb. 7.17 Daten- und Direktwertausgabe. Signalflüsse im Überblick. Daten, die der Befehl liest oder schreibt, werden nebenher auch ausgegeben. WRITE = Schreibsteuersignal des Prozessors; MEM_WR = Schreibsteuersignal am Speicherbus; PORTSEL = Auswahl des Ausgabeports; OPSEL = Operationsauswahl. 1 - Direktwertausgabe (DAOI), 2 - Ausgabe beim Lesen (DAOR); 3 - Ausgabe beim Schreiben (DAOW).

7.3.3 Adreßausgabe

Die während der Befehlsausführung gelieferte Datenadresse wird zur Datenausgabe genutzt. Die Adreßbits werden als Datenbits interpretiert und einem Ausgabeport zugeführt (Abb. 7.18 bis 7.20).

Die ausgegebene Datenadresse ist eine effektive Adresse, also nicht immer der Adreßwert im Befehl allein, sondern oftmals das Ergebnis einer Adreßrechnung (Basis + Displacement oder indirekte Adressierung). Das ist beim Programmieren zu berücksichtigen (und gibt womöglich Gelegenheit zum Tricksen...).

Die Ansteuerung des Speichers

Zugriffe mit beliebigen Adressen dürfen keine Nebenwirkungen haben, kein Schreiben, keine Adreßumsetzung, keine Fehlersignale von Speicherschutzeinrichtungen usw. Einfache Mikrocontroller prüfen typischerweise nichts. Wird mit einer Adresse zugegriffen, der kein Speicher zugeordnet ist, werden feste oder undefinierte Bitmuster gelesen. Schreibzugriffe gehen ins Leere. Ansonsten richtet sich die Implementierung nach der Art des Speichersubsystems. Manchmal ist es möglich, Daten und Signale vom Speicher zu ignorieren (Don't Care), manchmal muß man dem Speicher signalisieren, daß er alle Prüfungen unterdrücken soll. Die naheliegende Einfachlösung: die Anforderungssignale gar nicht zum Speicher schicken, dem Prozesorkern gegenüber aber so tun, als ob der Speicherzugriff stattgefunden hätte.

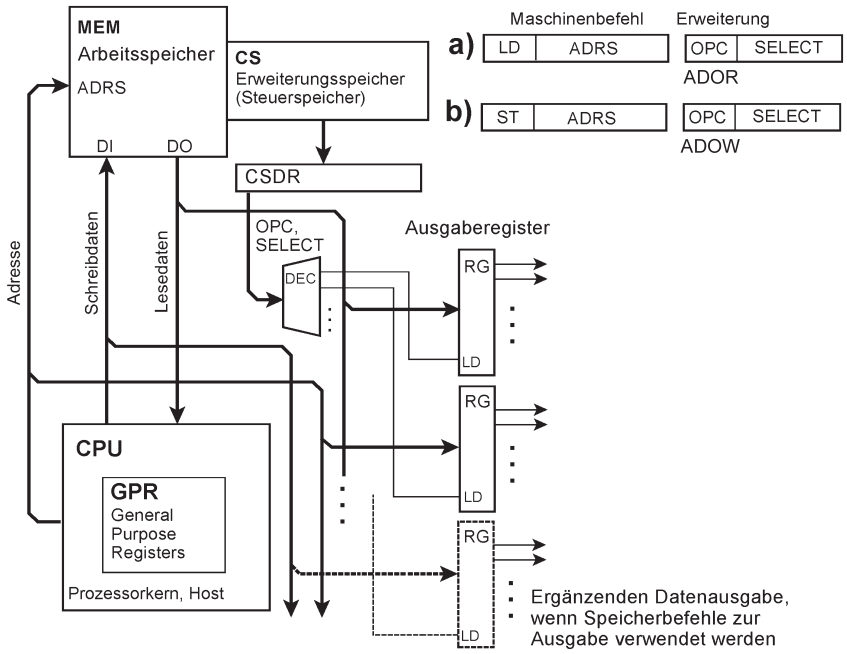


Abb. 7.18 Adreßausgabe. a) Erweiterung eines Befehls mit Lesezugriff (z. B. Ladebefehl). Nur die Adresse wird ausgegeben. ADOR = Address Output while Reading. b) Erweiterung eines Befehls mit Schreibzugriff. Adresse und Schreibdaten können ausgegeben werden. ADOW = Address Output while Writing; OPC = Operationscode der Erweiterung; SELECT = Auswahl des Ausgaberegisters.

Adreßausgabe (1). Befehle mit Lesezugriff

Es werden Befehle erweitert, die Lesezugriffe zum Speicher ausführen (Abb. 7.18a und 7.19). Befehlsbeispiel:

LD cpu_reg, Adresse | ADOR out_reg

Die Adresse des Lesezugriffs wird ausgegeben. Die Daten, die vom Speicher kommen, werden ignoriert (Don't Care). Zugriffe mit beliebigen Adressen dürfen nicht zu Problemen führen (Fehlermeldungen, Hängenbleiben). Eine Einfachlösung besteht darin, die Leseanforderung insgesamt zu unterdrücken (wie in Abb. 7.19 angedeutet). Diese Art der Befehlserweiterung kann zweckmäßig sein, wenn die Adreßlänge des Prozessors größer ist als die Verarbeitungsbreite. Ein 8-Bit-Prozessor mit 16-Bit-Adressierung kann so 16 Bits auf einmal ausgeben.

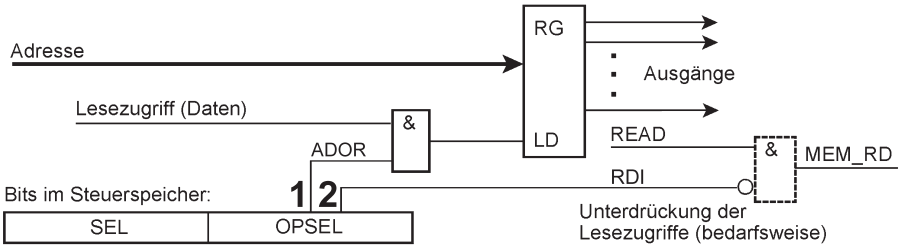


Abb. 7.19 Adreßausgabe in Lesezugriffen. Signalflüsse im Überblick. Die Adresse wird als Datenwort ausgegeben. SEL = Auswahl des Ausgaberegisters; OPSEL = Erweiterungssteuersignale; READ = Datenlesezugriff vom Prozessor; MEM_RD = Leseanforderung an den Speicher. 1 - Adreßausgabe (ADOR); 2 - Leseverhinderung (RDI = Read Inhibit).

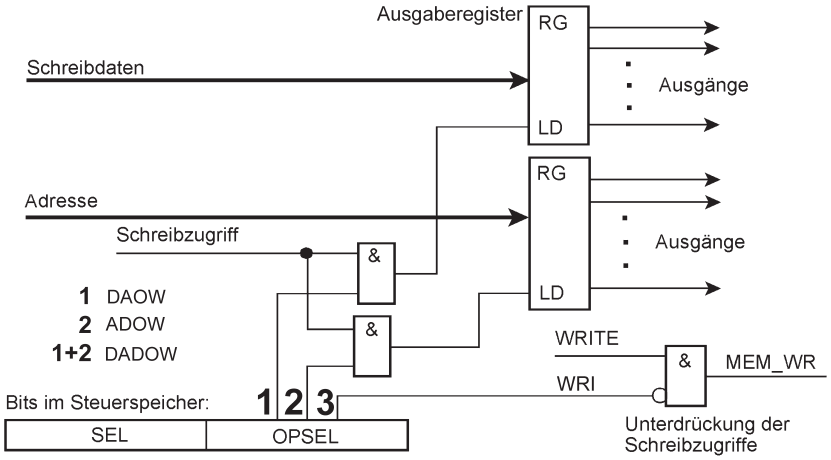


Abb. 7.20 Adreß- und Datenausgabe in Schreibzugriffen. Signalflüsse im Überblick. Die Adresse wird als Datenwort ausgegeben. Das vom Prozessor zum Schreiben bereitgestellte Datenwort kann zusätzlich ausgegeben werden (Varianten: Datenwort allein, Adresse allein, Adresse und Datenwort). In den Speicher darf dabei nichts geschrieben werden. SEL = Auswahl des Ausgaberegisters; WRITE = Schreibsteuersignal des Prozessors; MEM_WR = Schreibsteuersignal am Speicherbus; OPSEL = Erweiterungssteuersignale: 1 - Datenausgabe; 2 - Adreßausgabe; 3 - Schreibverhinderung (WRI = Write Inhibit); DAOW = Data Output while Writing; ADOW = Address Output while Writing; DADOW = Data and Address Output while Writing.

Adreßausgabe (2). Befehle mit Schreibzugriff
Es werden Befehle erweitert, die Schreibzugriffe zum Speicher ausführen (Abb. 7.18b und 7.20). Befehlsbeispiel:

```
ST Adresse, cpu_reg | ADOW out_reg
```

Man kann nicht nur die Adresse ausgeben, sondern auch die zu schreibenden Daten:

```
ST Adresse, cpu_reg | DADOW out_reg
```

Das Schreiben in den Speicher muß unterdrückt werden (wie in Abb. 7.20 gezeigt). Ein 8-Bit-Prozessor mit 16-Bit-Adressierung kann so 24 Bits auf einmal ausgeben, ein 32-Bit-Prozessor bis zu 64 Bits.

7.4 Dateneinspeisung

Die Dateneinspeisung erfordert ein Umschalten der Datenwege (Abb. 7.21). Beim Befehlslesen wird auch der Mikrobefehl aus dem Erweiterungsspeicher geholt. So verbleibt genügend Zeit, um die Datenwege umzuschalten, ehe der Datenzugriff des Befehls beginnt (vgl. Abb. 7.14). Hier wird nur die zusätzliche Durchlaufverzögerung der Auswahlschaltungen wirksam. Sie ist nicht allzu lang. Um sie besonders kurz zu halten, könnte man beispielsweise Transfer-Gates einsetzen. Im FPGA (Softcore-Implementierung) paßt die Auswahlfunktion womöglich in die Logikzellen der Speicherzugriffswege oder des Speicheradapters.

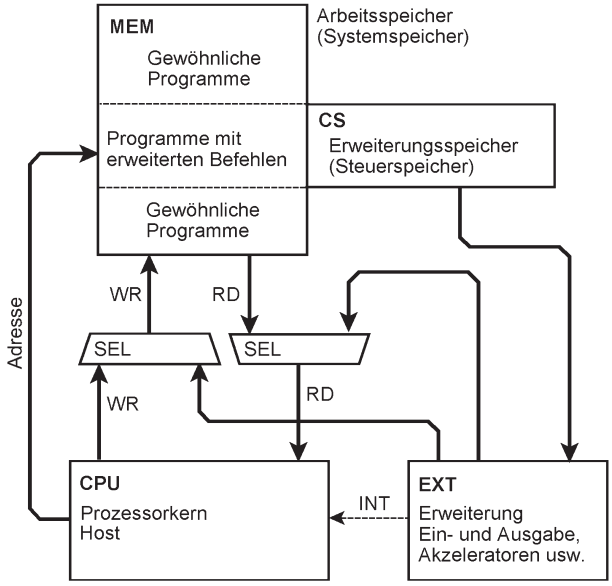


Abb. 7.21 Erweiterter Prozessor mit Dateneinspeisung. Hier sind nur die Datensignalwege der Speicherschnittstelle dargestellt.

7.4.1 Eingabe in die Prozessorregister

In Lade- oder Operationsbefehlen[6] wird anstelle des adressierten Speicherinhalts ein Bitmuster gelesen, das beim Datenzugriff in den Lesedatenweg zum Prozessor eingespeist wird (Abb. 7.22 und 7.24a). Befehlsbeispiel:

LD cpu_reg, Adresse | DAIR in_reg

Das Prozessorregister wird mit den ausgewählten Eingabedaten geladen. Die vom Befehl gelieferte Datenadresse kann ignoriert oder zur Auswahl der Lesedaten oder zur Datenausgabe genutzt werden (Adreßausgabe).

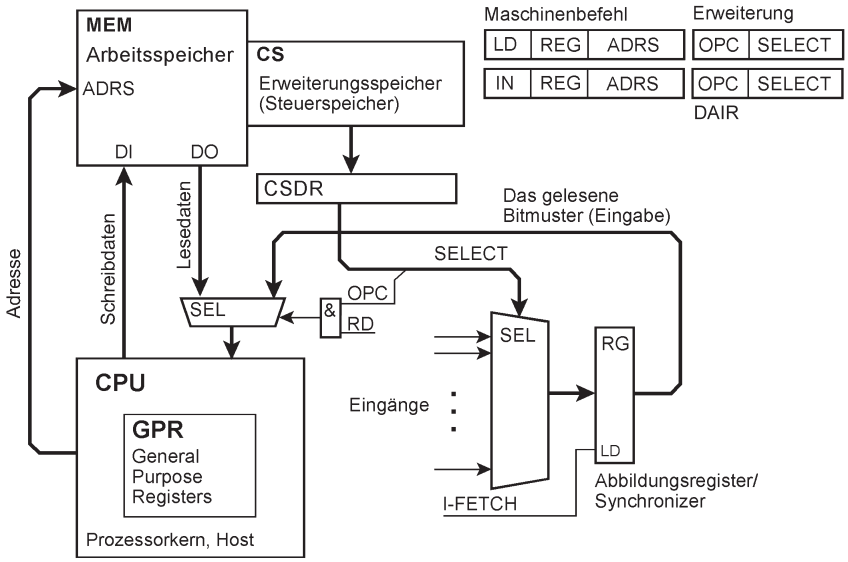


Abb. 7.22 Zusatzeingabe (1). Eingabe in die Prozessorregister. OPC = Operationscode der Erweiterung; SELECT = Auswahl der Eingänge; I-FETCH = Befehlslesezugriff; RD = Datenlesezugriff. Am Ende des Befehlslesens wird das ausgewählte Eingangsbitmuster in ein Abbildungsregister übernommen, damit es beim nachfolgenden Lesezugriff stabil anliegt. DAIR = Data Input while Reading.

7.4.2 Eingabe in den Arbeitsspeicher

In Speicherbefehlen[7] wird anstelle des Datenworts, das vom Prozessor kommt, ein Bitmuster geschrieben, dessen Quelle vom Steuerwort ausgewählt wird (Abb. 7.22 und 7.24b). Das Bitmuster wird beim Datenzugriff in den Schreibdatenweg eingespeist. Die vom Befehl gelieferten Schreibdaten können ignoriert oder zur Datenausgabe genutzt werden.

6. Befehlswirkung: <Register> := <Eingabeadresse> oder <Register> := <Register> **op** <Eingabeadresse>.
7. Befehlswirkung: <Arbeitsspeicheradresse> := <Register>.

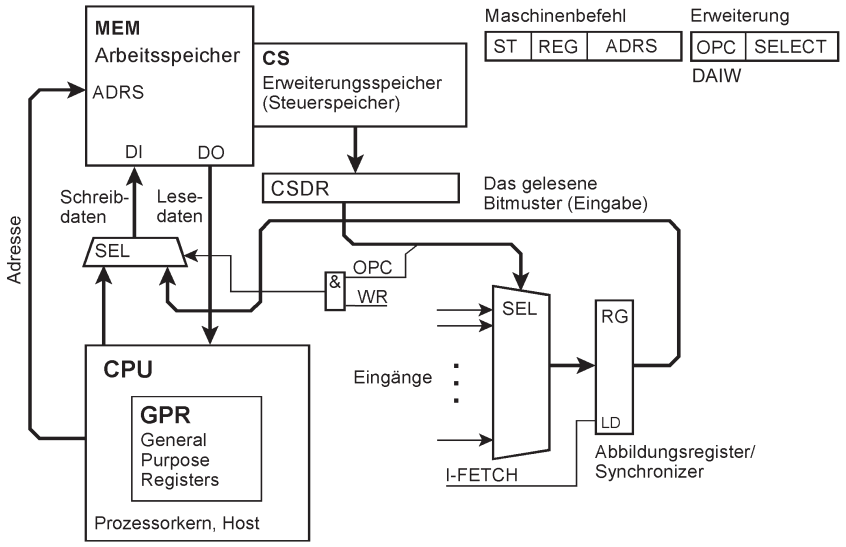


Abb. 7.23 Zusatzeingabe (2). Eingabe in den Arbeitsspeicher. OPC = Operationscode der Erweiterung; SELECT = Auswahl der Eingänge; I-FETCH = Befehlslesezugriff; WR = Datenschreibzugriff. Am Ende des Befehlslesens wird das ausgewählte Eingangsbitmuster in ein Abbildungsregister übernommen, damit es beim nachfolgenden Schreibzugriff stabil anliegt. DAIW = Data Input while Writing.

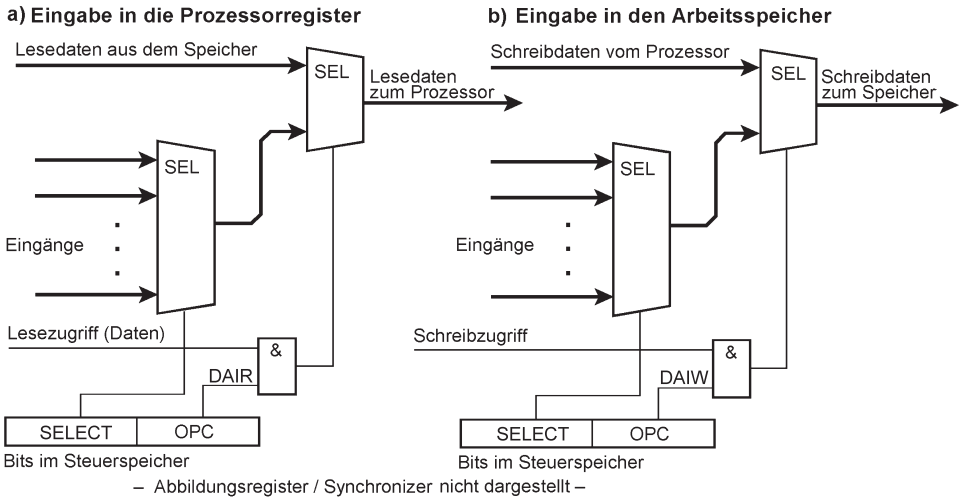


Abb. 7.24 Zusatzeingabe. Signalflüsse im Überblick. Synchronisierung und Abbildungsregister weggelassen.

7.4.3 Adressen einspeisen

Von außen veränderte oder eingespeiste Adressen können zur Funktionsverzweigung, zu Tabellenzugriffen usw. genutzt werden. Wir wollen aber nicht in die Speicheradressierung eingreifen, sondern nur in die Datenweg. Eine naheliegende Lösung wäre, beim Befehlslesen die Adresse zu beeinflussen, die der Befehl mitbringt (Befehlsmodifikation). Eine alternative Lösung besteht darin, Befehle mit indirekter Adressierung zu nutzen. Ein solcher Befehl führt zunächst einen Datenzugriff aus, um die Adresse zu holen. Während dieses Zugriffs wird der Lesedatenweg umgeschaltet, wie in den Abb. 7.22 und 7.24a gezeigt.

7.5 Befehlsmodifikation

Der Prozessorkern adressiert einen Befehl im Speicher, erhält aber ein Befehlsbitmuster, das – in Teilen oder als Ganzes – von außen kommt (Abb. 7.25 bis 7.27). Die Erweiterungsschaltungen können einzelne Bits, Bitfelder oder komplette Befehle einspeisen.

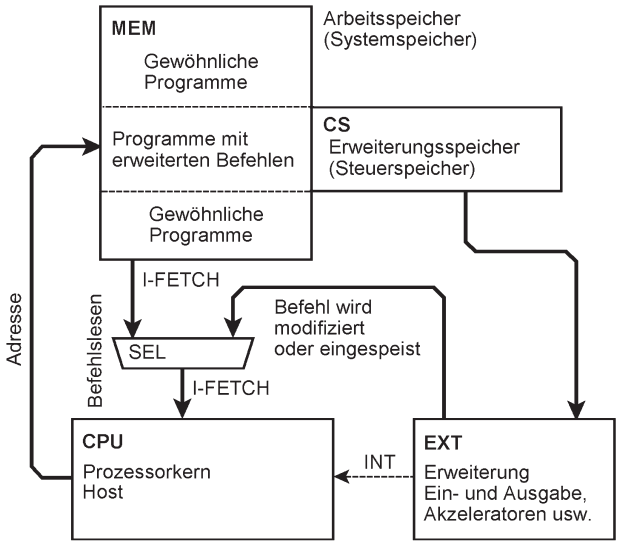


Abb. 7.25 Erweiterter Prozessor mit Befehlsmodifikation. Hier sind nur die Befehlslesewege der Speicherschnittstelle dargestellt.

Welcher Befehl steuert das Einspeisen – der aktuelle oder der vorhergehende?
Es liegt nahe, daß der aktuelle Befehl auch die Steueranweisungen mitbringt. Dann ist alles beieinander. Abb. 7.26 veranschaulicht den Ablauf. Erst ist das Steuerwort aus dem Steuerspeicher zu lesen, dann ist der Befehlsleseweg umzuschalten, dann zu warten, bis der Host den Befehl gelesen hat. Aus dem Steuerwort ergibt sich die Befehlsmodifikation bis hin zum Einspeisen eines kompletten anderen Befehls. Das kann dazu zwingen, die Taktfrequenz herabzusetzen oder Wartezustände einzufügen. Hierdurch dauert das Befehlslesen länger.

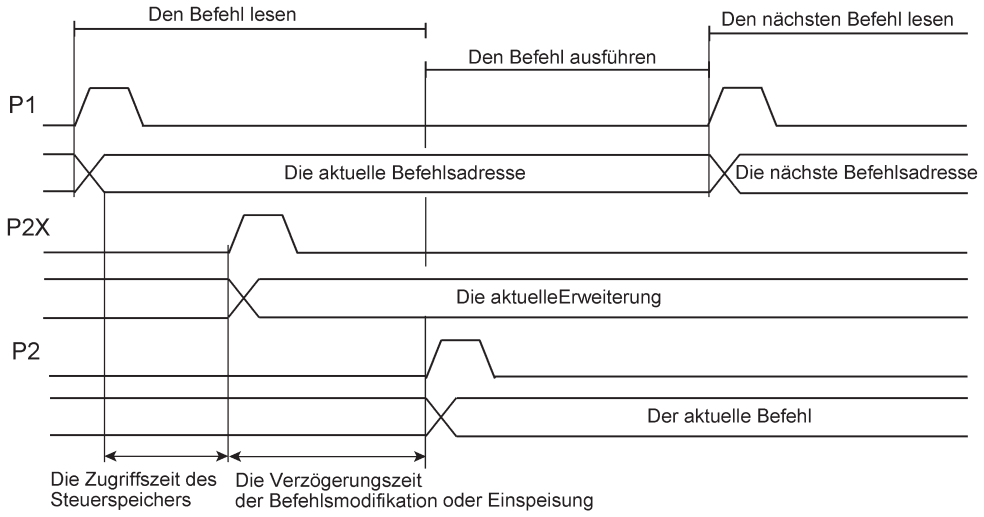


Abb. 7.26 Befehlslesen mit Befehlsmodifikation (1). Das Steuerwort aus dem Steuerspeicher entscheidet darüber, ob der Befehl verändert wird oder nicht. Soll er verändert werden, müssen die Änderungen eingespeist werden, bevor die nächste Phase (P2) beginnt.

Eine Alternative besteht darin, die Befehlsmodifikation bereits in dem Steuerwort anzuweisen, das den vorhergehenden Befehl begleitet (Abb. 7.27). Dann steht genügend Zeit zur Verfügung, um über die Modifikation zu entscheiden und die Befehlslesewege entsprechend umzuschalten. Dabei sind aber grundsätzliche Besonderheiten zu beachten:

- Zwischen beiden Befehlen darf keine Unterbrechung wirksam werden. Ggf. ist eine zusätzliche zeitweilige Unterbrechungsverhinderung erforderlich (vgl. Abschnitt 7.2.1 (S. 410f), insbesondere Abb. 7.9).
- Es stellt sich die Frage, ob ein solcher zu modifizierender Befehl ein Sprungziel sein darf. Wenn ja, müßten die zugehörigen Steueranweisungen auch in den Erweiterungen der Verzweigungsbefehle enthalten sein. Die alternative Einfachlösung: grundsätzlich zwei Befehle nehmen – und wenn der erste ein NOP ist. Der eine bereitet das Umschalten vor, der nachfolgende wird vom Prozessorkern als modifizierter oder substituierter Befehl gelesen. Nur der erste Befehl darf ein Sprungziel sein.

Wenn man den Prozessorkern ändern darf oder selbst entwirft, könnte man auch daran denken, für die Befehlsmodifikation einen zusätzlichen Taktzyklus einzufügen. Auch ein so verlangsamter Befehl ist sicherlich schneller als das Programmstück, dessen Wirkungen er ersetzt.
Die nachfolgend beschriebenen Funktionen können mit einem Befehl oder mit zwei Befehlen implementiert werden. Die Wirkungsweise ist aber leichter zu verstehen, wenn das Steuerwort den zu modifizierenden Befehl begleitet. Dieses Prinzip liegt auch den Blockschaltbildern zugrunde.

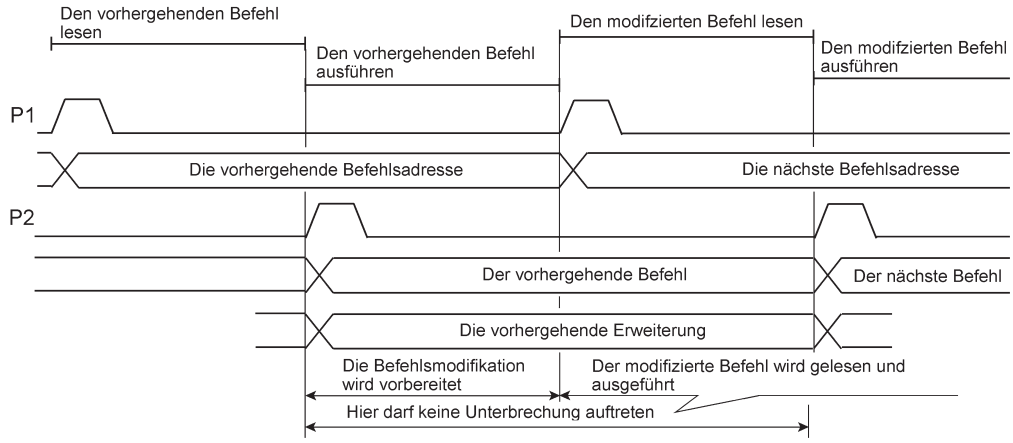


Abb. 7.27 Befehlslesen mit Befehlsmodifikation (2). Die Steueranweisungen stehen in dem Steuerwort, das zusammen mit dem vorhergehenden Befehl gelesen wird. Dann sind beim Lesen des zu modifizierenden Befehls die Lesedatenwege schon eingestellt.

7.5.1 Befehlssubstitution

Befehlssubstitution heißt, den Befehl, den der Prozessor im Arbeitsspeicher adressiert hat, durch einen anderen Befehl zu ersetzen (Abb. 7.28).

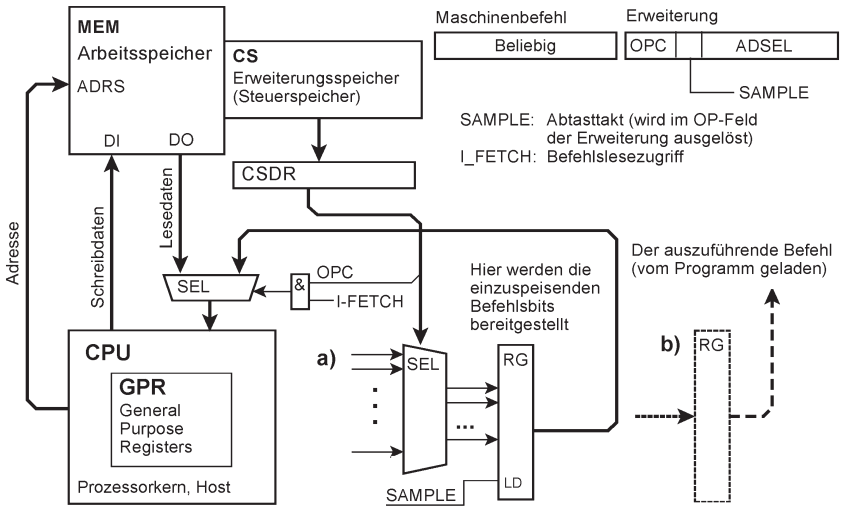


Abb. 7.28 Befehlssubstitution. a) Funktionsverzweigung durch Signaleinspeisung in bestimmte Bitpositionen des Befehls, z. B. in die Verzweigungsadresse; b) Ausführung eines zuvor geladenen Befehls. OPC = Operationscode der Erweiterung; ADSEL = Verzweigungsbitauswahl; SAMPLE = Verzweigungsbits abtasten; I-FETCH = Befehlslesezugriff.

7.5.2 Bedingte Befehlsausführung

Die Befehle werden mit Bedingungsauswahlsignalen erweitert. Die Bedingungen werden aus der Umgebung des Prozessors oder aus der Außenwelt entnommen (Eingangssignale). Sie beeinflussen den Befehlsleseweg des Prozessors. Ist die ausgewählte Bedingung erfüllt, wird der aus dem Speicher gelesene Befehl zum Prozessor weitergeleitet. Ist sie nicht erfüllt, wird ein NOP aufgeschaltet (Abb. 7.27 und 7.28). Diese Erweiterungsfunktion wirkt wie die Steuerung der Befehlsausführung mit Prädikaten (Predication)[8]. Hier ist man aber nicht auf den Inhalt eines Prädikatregisters oder auf wenige Bedingungsbits beschränkt, sondern kann mit beliebig vielen Prädikaten beliebiger Herkunft arbeiten. Soll ein Block aus mehreren Befehlen übergangen werden, ist es wichtig, daß die Bedingung nur am Anfang erfaßt und dann gehalten wird. In der Schaltung von Abb. 7.29 und 7.30 wird das Erfassen (Abtasten) der Bedingungen vom Steuerwort ausgelöst. Alle Bedingungen werden auf einmal abgetastet und dann gehalten (Prinzip des Abbildungsregisters).

Eine bedingte Verzweigung auf eine solche Bedingung wird mit einem unbedingten Verzweigungsbefehl programmiert, der bei Nichterfüllung übergangen wird. Ein Warten auf eine solche Bedingung ist eine unbedingte Verzweigung auf sich selbst, die bei Erfüllung übergangen und bei Nichterfüllung ausgeführt wird.

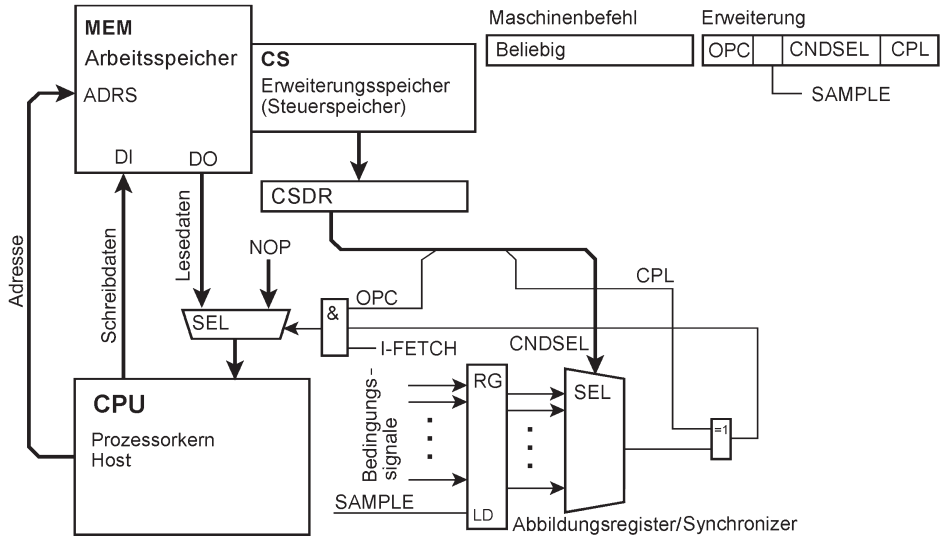


Abb. 7.29 Bedingte Befehlsausführung. OPC = Operationscode der Erweiterung; CNDSEL = Bedingungsauswahl; CPL = Invertierung (Complement); I-FETCH = Befehlslesezugriff; SAMPLE = Bedingungen abtasten.

8. Vgl. die Architekturen IA-64 (Itanium) und ARM. Bei einer Architektur wie ARM müßte man keine NOPs einspeisen. Es würde genügen, das Prädikatfeld des Befehls zu modifizieren.

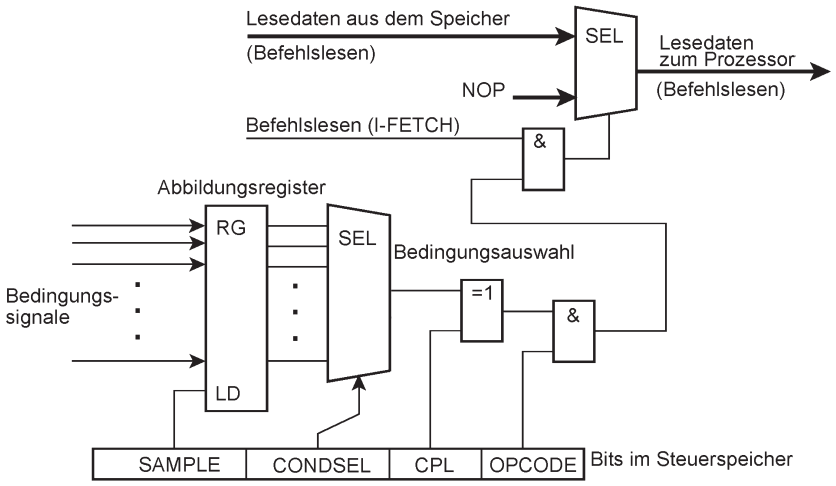


Abb. 7.30 Bedingte Befehlsausführung. Signalflüsse im Überblick.

7.5.3 Funktionsverzweigung

Der Befehl wird nicht komplett ausgewechselt. Vielmehr werden beim Befehlslesen bestimmte Bitpositionen des Befehlsformats auf ausgewählte Signale umgeschaltet. Die typische Nutzung besteht darin, in Verzweigungs- oder Unterprogrammrufbefehlen die Verzweigungsadresse abzuwandeln (Abb. 7.31). Ein solcher Befehl wird so zu einer Art Mikrobefehl, der zu einem von beispielsweise 8, 16, 32 usw. Nachfolgern verzweigen kann[9], bis hin zum Verzweigen mit Operationscodes oder Kommandobytes.

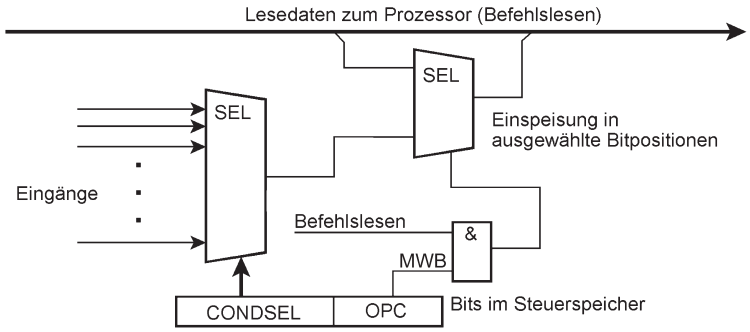


Abb. 7.31 Funktionsverzweigung durch Befehlsmodifikation. Signalflüsse im Überblick. COND-SEL = Bedingungsauswahl; MWB = Multiway Branch.

9. Wir speisen die Adreßbits beim Befehlslesen ein und nicht etwa in die Befehlsadresse beim Verzweigen, weil wir nicht in die Speicheradressierung der Host-Architektur eingreifen wollen.

7.5.4 Der EXECUTE-Befehl

Ein EXECUTE-Befehl bewirkt, daß ein Arbeitsspeicher- oder Registerinhalt als Befehl ausgeführt wird[10]. Wenn dieser Befehl eine Verzweigung auslöst, wird das Programm in Verzweigungsrichtung fortgesetzt. Ansonsten wird der Befehl ausgeführt, der dem EXECUTE-Befehl nachfolgt. EXECUTE ist dann praktisch ein Unterprogrammruf, wobei das Unterprogramm aus einem einzigen Befehl besteht. Diese Funktion kann implementiert werden, indem man die Auswahlschaltung im Datenweg an programmseitig ladbare Register (Befehlspuffer) anschließt (Abb. 7.28b und 7.32). Beim sozusagen klassischen EXECUTE-Befehl wird der Befehlspuffer vom Programm geladen. Man könnte aber auch daran denken, den Befehl von außen zuzuführen, beispielsweise von einem übergeordneten Steuerautomaten.

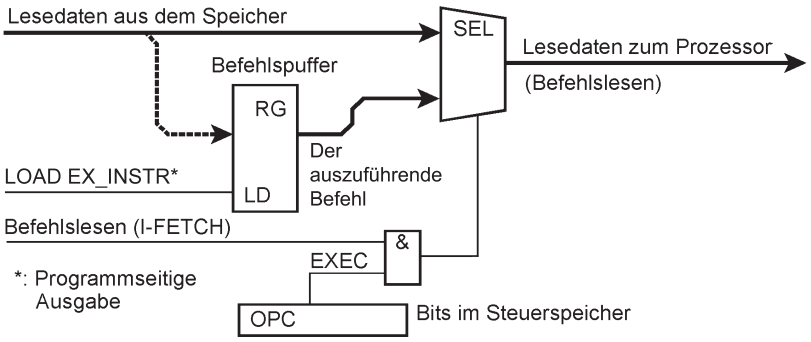


Abb. 7.32 Der EXECUTE-Befehl. Der Befehlspuffer wird vom Programm oder von einer externen Einrichtung geladen. LOAD_EX_INSTR = Laden des auszuführenden Befehls; EXEC = den geladenen Befehl ausführen.

7.5.5 Befehlsausgabe

Komplette Befehle[11] werden als Bitmuster ausgegeben. Sie können außen als spezielle Befehle, Kommandos usw. wirken, aber auch als einfache Ausgabedaten. Dem Prozessor hingegen werden NOP-Befehle zugeführt (Abb. 7.33 und 7.34). Diese Funktion ermöglicht es, den Befehlsleseablauf zu Ausgabezwecken auszunutzen[12]. Solche Ausgaben stehen sozusagen mitten im Maschinenprogramm. Man kann sie als Unterprogramme aufrufen oder zu ihnen verzweigen. Damit lassen sich Datenausgaben und externe Befehle (für periphere Steuereinheiten, Akzeleratoren usw.) so in die Programme einbauen, als würden sie von Grund auf zu Prozessorarchitektur gehören.

10. In der Anfangszeit der Computertechnik war es üblich, Befehle im laufenden Programm zu verändern. Dann ist man aber zu Programmen übergegangen, die sich während der Ausführung nicht ändern (reine Prozeduren). Der EXECUTE-Befehl ist eine Art Aushilfe. So kann man auch in reinen Prozeduren mit Befehlen arbeiten, die das Programm selbst aufgesetzt hat, um Abläufe zu beschleunigen oder Mängel der Architektur zu umgehen.
11. Im Gegensatz zur Direktwertausgabe (S. 413ff).
12. Das lückenlose Befehlslesen – ohne Datenzugriffe dazwischen – ist oftmals der schnellste Leseablauf.

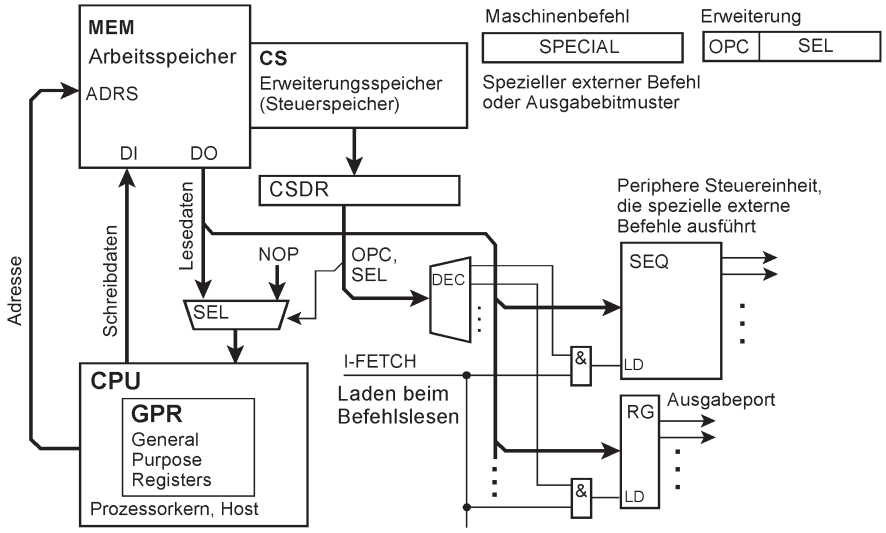


Abb. 7.33 Befehlsausgabe. Das Befehlsbitmuster wird ausgegeben. Es kann außen als externer Befehl o. dergl. wirken, aber auch als Datenwort ausgegeben werden. Der Prozessor bekommt statt dessen einen NOP angeboten. OPC = Operationscode der Erweiterung; SEL = Auswahl des Ausgabeports; I-FETCH = Befehlslesezugriff.

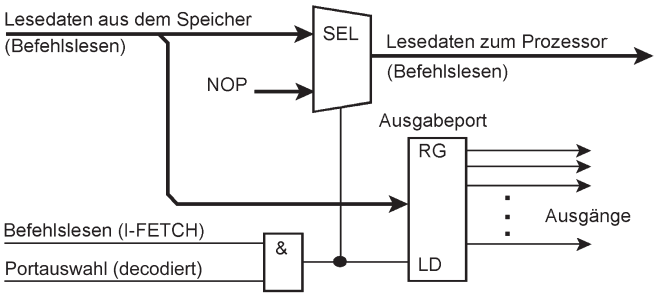


Abb. 7.34 Befehlsausgabe. Signalflüsse im Überblick.

7.6 Anwendungsbeispiele

7.6.1 Akzeleratoren anschließen

Der Akzelerator (Accelerator) ist ein zusätzliche Verarbeitungseinrichtung, die einem universellen Prozessor beigeordnet wird. Abb. 7.35 veranschaulicht, wie eine solche Einrichtung genutzt wird. Es ist ein autonomes Verarbeitungswerk mit programmseitig zugänglichen Registern an den Ein- und Ausgängen.

Zunächst lädt das Programm die Operanden in die eingangsseitigen Register, wählt die auszuführende Operation aus und startet sie. Der Prozessor wartet, bis der Akzelerator seine Arbeit beendet hat. Dann holt er die Ergebnisse aus den ausgangsseitigen Registern ab.

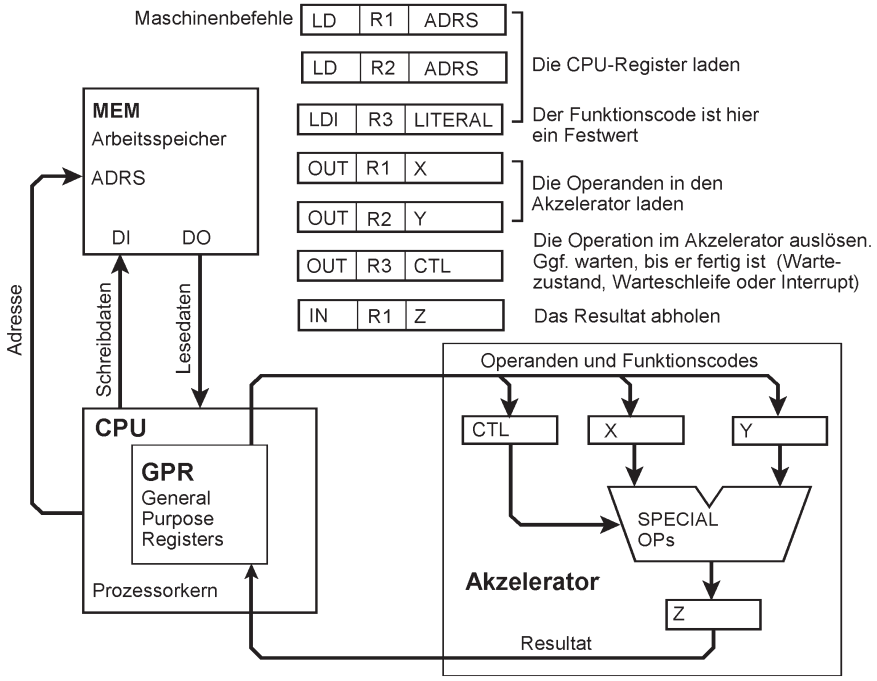


Abb. 7.35 So wirkt ein einfacher Akzelerator (z. B. im FPGA) mit dem Host-Prozessor zusammen[13]. Der eine Parameter der E-A-Befehle (IN, OUT) ist eine Registeradresse des Prozessors, der andere eine E-A-Adresse, die ein Register im Akzelerator auswählt.

Abb. 7.36 zeigt, wie der Akzelerator an einen erweiterten Prozessor angeschlossen werden kann. Die Register werden mit nebenläufigen Ausgaben geladen. Der Prozessor muß nur die Operanden aus dem Speicher lesen und in seine Register laden. Dabei gelangen sie gleichsam nebenher in das angeschlossene Rechenwerk. Die Ergebnisse werden über Zusatzeingaben abgeholt. In der Variante von Abb. 7.37 wird der Rechenablauf nicht von einem eigenen Befehl gestartet, der den Funktionscode lädt, sondern in der Erweiterung des Befehls angewiesen, der den letzten Operanden einträgt.

Es ist zudem nicht schwierig, den Prozessor so lange im Wartezustand zu halten, bis das Ergebnis vorliegt. Der Akzelerator verhält sich dann nicht wie ein nachträglicher Zusatz, dessen Nutzung mit einem gewissen Verwaltungs-Overhead verbunden ist, sondern so, als ob er von Grund auf zum Prozessor gehören würde.

13. Vgl. einschlägige Applikationsschriften, beispielsweise [73].

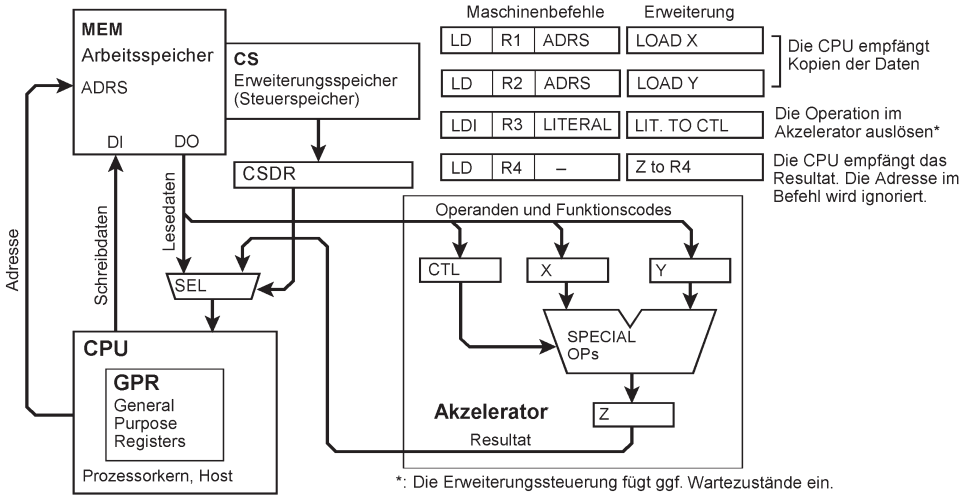


Abb. 7.36 So kann ein einfacher Akzelerator mit erweiterten Befehlen unterstützt werden.

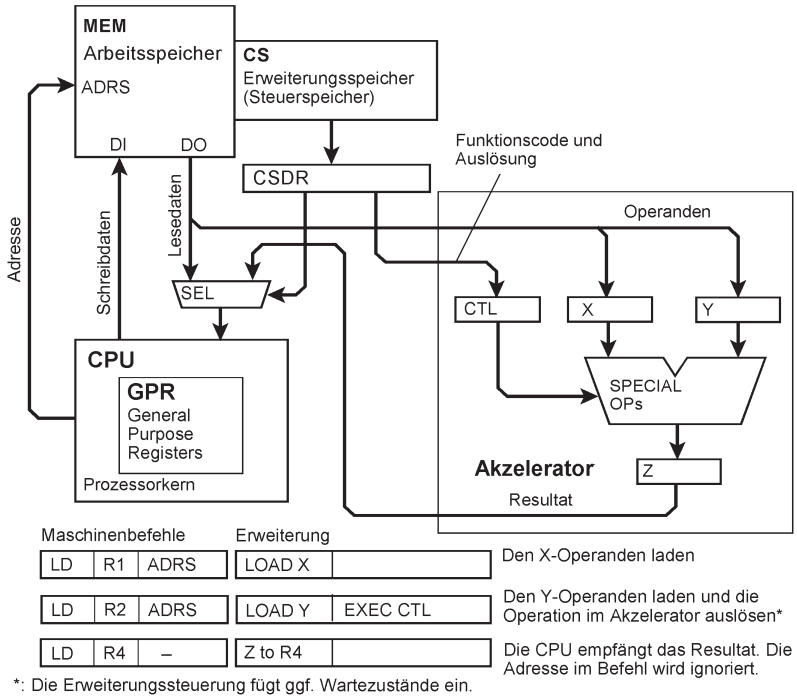


Abb. 7.37 Hier können die erweiterten Befehle auch die Operation im Akzelerator auswählen und auslösen. Er startet automatisch, nachdem der letzte Operand geladen wurde.

7.6.2 Zustandsautomaten unterstützen

Die Entwurfsaufgabe: die Zustandsübergänge eines einfachen universellen Branch Sequencers unterstützen, und zwar mit einem einzigen erweiterten Maschinenbefehl (Abb. 7.38).

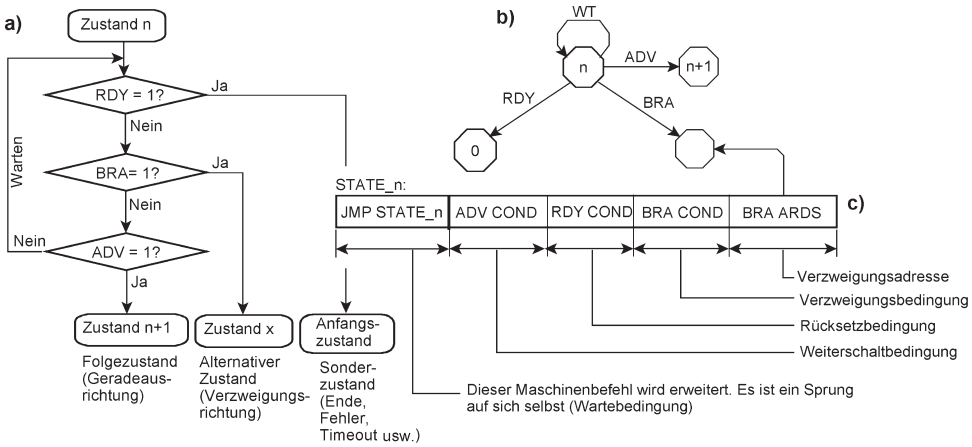


Abb. 7.38 Ein vielseitig nutzbares Schema von Zustandsübergängen[14]. a) Programmablauf (Flußdiagramm); b) Zustandsdiagramm; c) eine naheliegende Befehlserweiterung.

Was übliche Maschinenbefehle nicht können

In mehrere Richtungen verzweigen und warten. Die Entwurfsaufgabe besteht also darin, einen Maschinenbefehl auf solche Funktionen zu erweitern. Dem stehen aber grundsätzliche Schwierigkeiten gegenüber:

1) Verzweigungsadressen im Erweiterungsspeicher

Wenn es beliebige Adressen sein sollen und keine Festwerte, müßten die Maschinenprogrammfertigung und die Laufzeitorganisation darauf abgestellt sein. Die Compiler, Linker, Lader usw. müßten erweitert werden, um solche Adressen zu erzeugen. Deshalb kann es zweckmäßig sein, in den Erweiterungsspeicher keine Adressen aufzunehmen, die sich erst während der Maschinenprogrammfertigung oder beim Laden zwecks Ausführung ergeben (verschiebliche (relocatable) Programme). Was man einbringen könnte, wären Festadressen, Anweisungen zum Einspeisen vorgegebener Festadressen, oder Relativadressen, die sich auf die aktuelle Befehlsadresse beziehen[15].

2) Verzweigen auf Bedingungen

Die meisten Prozessoren können nicht auf beliebige Bedingungen von außen verzweigen, sondern nur auf interne Bedingungsbits (Flagbits).

14. Abb. 7.38 beruht auf den Abbildungen 2.14 und 6.18 (S. 38 und 335).
15. Vgl. die typischen relativen Verzweigungsbefehle, z. B. x86 Branch on Condition (Abb. 5.46).

3) Warten
Wir wollen harte Wartezustände vermeiden, also nicht etwa den Takt anhalten oder an der Speicherschnittstelle einen beliebig lange dauernden Wartezustand erzwingen oder dem Prozessor die Busherrschaft entziehen[16]. All das kann man tun, es sind aber harte Eingriffe, und man sollte gründlich überlegen, ob man deren Nachteile in Kauf nimmt. Es verbleibt also nur, durch Verzweigen auf sich selbst zu warten. Der Wartezustand wird beendet, indem man einen NOP-Befehl oder einen Verzweigungsbefehl mit der entsprechenden Zieladresse einspeist.

Das Übergangsmuster von Abb. 7.38 mit einem einzigen Befehl unterstützen
Die in Abb. 7.38c angedeutete Lösung – alles, was fehlt, im Steuerwort unterzubringen – sieht eine Verzweigungsadresse im Erweiterungsspeicher vor. Die Frage ist, ob man sich das leisten kann, vor allem in Hinblick auf die grundsätzliche Absicht, außerhalb der Erweiterungen alles so zu lassen wie es ist. Das Problem ist also ohne Verzweigungsadressen im Erweiterungsspeicher und ohne bedingte Verzweigungen im Prozessor zu lösen.

Ein Lösungsvorschlag:

- Das Warten ist eine unbedingte Verzweigung auf sich selbst. Dieser Befehl wird erweitert.
- Um zum Folgebefehl weiterzuschalten (Folgezustand), wird ein NOP eingespeist. Wenn die Maschine eine bedingte Befehlsausführung aufweist, die über Prädikatbits gesteuert wird (Predication; vgl. ARM), genügt anstelle des NOP ein passendes Prädikatbitmuster.
- Als Verzweigungsadresse im Steuerwort (BRANCH ADRS in Abb. 7.38) speichern wir eine relative Adresse (Offset, Displacement). Wird der erweiterte Befehl gelesen, so bringt er seine eigene Adresse mit. Soll bedingt verzweigt werden, wird die relative Adresse hinzuaddiert und das Ergebnis in den Befehlsleseweg eingespeist. Das bedeutet aber einen vergleichsweise beträchtlichen Aufwand (Addierer, Umschaltung im Befehlsleseweg). Wenn der Prozessor relative Verzweigungen unterstützt, kann man darauf verzichten. Der zu erweiternde Befehl ist dann eine Verzweigung zur alternativen Befehlsadresse. Um zu warten, wird eine unbedingte relative Verzweigung auf sich selbst eingespeist[17], um zum Folgebefehl weiterzuschalten, ein NOP.
- Alternativ dazu könnte man eine Adresse aus einem Register einspeisen oder auch einen kompletten Verzweigungsbefehl (Prinzip des EXECUTE-Befehls). Entwicklungssoftware und Prozessorarchitektur unterstützen typischerweise das Gewinnen effektiver Adressen, die man programmseitig verarbeiten und ausgeben kann[18].
- Um zur Behandlung eines Sonderzustands zu verzweigen, wird ein Verzweigungsbefehl mit einer Festadresse eingespeist. Ob man statt dessen einfach ein Rücksetzen auslöst, ist Sache der Entwurfsoptimierung.

16. Vgl. Abschnitt 8.7 (S. 483).
17. Das ist ein Festwert. Vgl. Abb. 5.46. In diesem Beispiel wäre der Offset = –2.
18. Beispielsweise mit Befehlen *Load Effective Address* (LEA).

7.6.3 Den Prozessor auf Kommandosteuerung umbauen

Der Start-Stop-Betrieb kann programmseitig implementiert werden. Um den Overhead zu verringen, kann man den Stopzustand als Verzweigung auf sich selbst programmieren und über ein Ausgabefeld im Erweiterungsspeicher an den Steuerautomaten melden. Der erneute Start erfolgt durch Einspeisen eines NOP oder eines EXECUTE-Befehls, der vom Steuerautomaten geladen wird. Es ist typischerweise eine unbedingte Verzweigung zur jeweils auszuführenden Verarbeitungsfunktion.

7.6.4 In den Prozessorkern eingreifen

Diese Gelegenheit ergibt sich, wenn man einen IP Core verändern darf oder von Grund auf neu entwirft. Die Maschine soll grundsätzlich voll kompatibel bleiben. Deshalb werden alle neuen, besonderen, anwendungsspezifischen usw. Funktionen im Erweiterungsspeicher codiert. Bei voller Wahrung der Kompatibilität könnte man u. a.

- auf alternative Register umschalten (alternative Flagbits, Adreßzeiger usw.),
- alternative Registersätze adressieren,
- zusätzliche Operationen auslösen,
- das Setzen von Flagbits selektiv steuern[19],
- externe Verzweigungsbedingungen einspeisen (Abb. 7.39),
- externe Signale in die Verzweigungsadressen einspeisen (Mehrwege- und Funktionsverzweigung; Abb. 7.40 und 7.41),
- im Start-Stop-Betrieb hart anhalten (Taktstop),
- die Ein- und Ausgabe auch in Verarbeitungsbefehlen unterstützen.

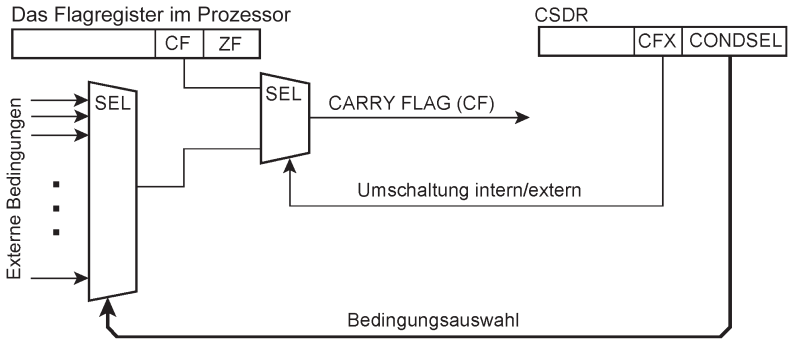


Abb. 7.39 Flagbiteinspeisung. In das Carry Flag (CF) wird eine externe Bedingung eingespeist. CFX = CF-Erweiterung; CONDSEL = Bedingungsauswahl.

19. Es ist in der Architektur fest vorgegeben, welche Flagbits von den einzelnen Befehlen gestellt werden. Das zwingt nicht selten dazu, Flags zu retten (z. B. auf den Stack) oder einen unübersichtlichen Maschinencode mit verwirrenden Verzweigungen ("Spaghetti-Code") zu erzeugen. Bits im Steuerwort können veranlassen, die Architekturvorgaben zu umgehen, z. B. bestimmte Flagbits nicht zu stellen, oder auf alternative Flagregister umzuschalten.

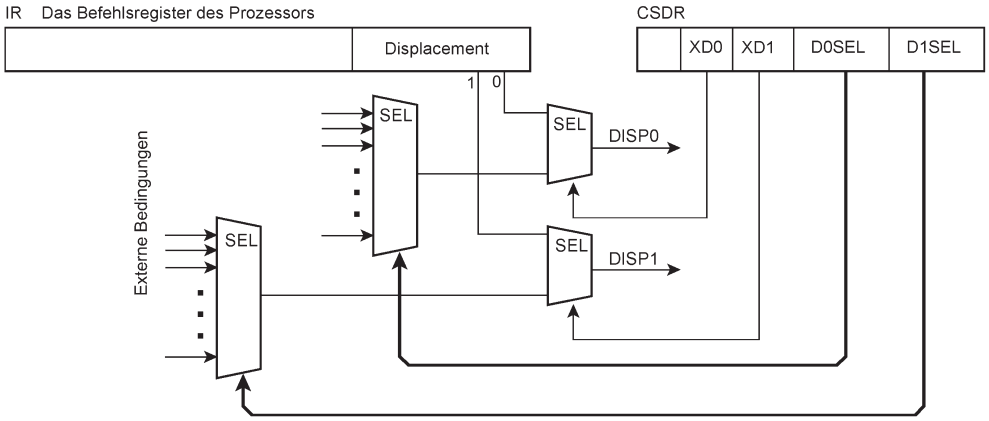


Abb. 7.40 Externe Erweiterung von Verzweigungsbefehlen auf Vierwegeverzweigung[20]. Der Verzweigungsbefehl liefert ein Adreß-Displacement. Die zwei niedrigstwertigen Bitpositionen können auf externe Bedingungssignale umgeschaltet werden.

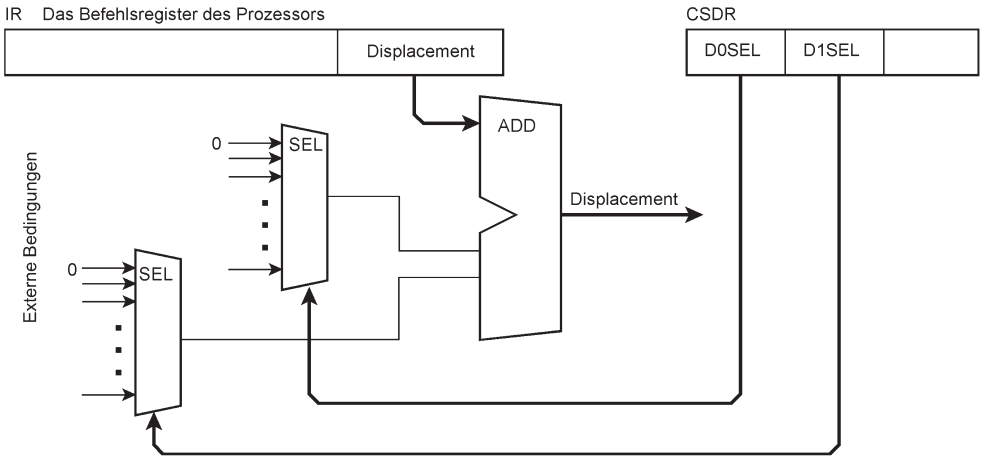


Abb. 7.41 Werden Bedingungen in die Adresse eingespeist, müssen die Sprungziele an entsprechenden integralen Adressen placiert werden. Wenn man die Bedingungsbits zum Displacement im Befehl addiert, entfällt diese Einschränkung.

20. Die Abbildungen sind nur Beispiele. Sie zeigen die einfache Vierwegeverzweigung, wie sie in Kapitel 5 anhand der Abb. 5.69 (S. 273) erläutert wird. Grundsätzlich können alle Lösungen der Mehrwege- und Funktionsverzweigung sinngemäß implementiert werden.

7.7 Der Erweiterungs- oder Steuerspeicher

Es ist ein besonderer Speicher, den den Arbeitsspeicher ergänzt. Die Zugriffsbreite richtet sich nach den jeweiligen Anforderungen. Es können nur wenige, aber auch sehr viele Bits sein. Der Adreßraum kann dem des Arbeits- oder Programmspeichers entsprechen. Es liegt aber auch nahe, den Steuerspeicher kleiner zu dimensionieren.
Speicherbereiche, denen ein Steuerspeicher zugeordnet ist, können auch Host-Befehle ohne Erweiterungswirkung enthalten. In unseren Darstellungen gilt, daß es keine Erweiterungswirkungen gibt, wenn das betreffende Wort im Steuerspeicher nur Nullen enthält.
Der Steuerspeicher kann im gesamten Adreßraum oder in einzelnen Bereichen angeordnet sein. Die erweiterten Steuerwirkungen werden ausgelöst, wenn die Adresse des Befehlslesens eine solche Bereichsadresse ist.

Mehrere Adreßbereiche
Abb. 7.42 veranschaulicht, daß man mehrere Steuerspeicher vorsehen kann, auch mit unterschiedlicher Zugriffsbreite und Speicherkapazität. Wenn beispielsweise verschiedenartige periphere Einrichtungen zu unterstützen sind und jeweils andere Funktionen benötigt werden, ist es eine Frage der Entwurfsoptimierung, ob man unterschiedliche Steuerspeicher und Erweiterungsschaltungen vorsieht oder eine einzige entsprechend großzügig ausgelegte Erweiterung, die alles kann.

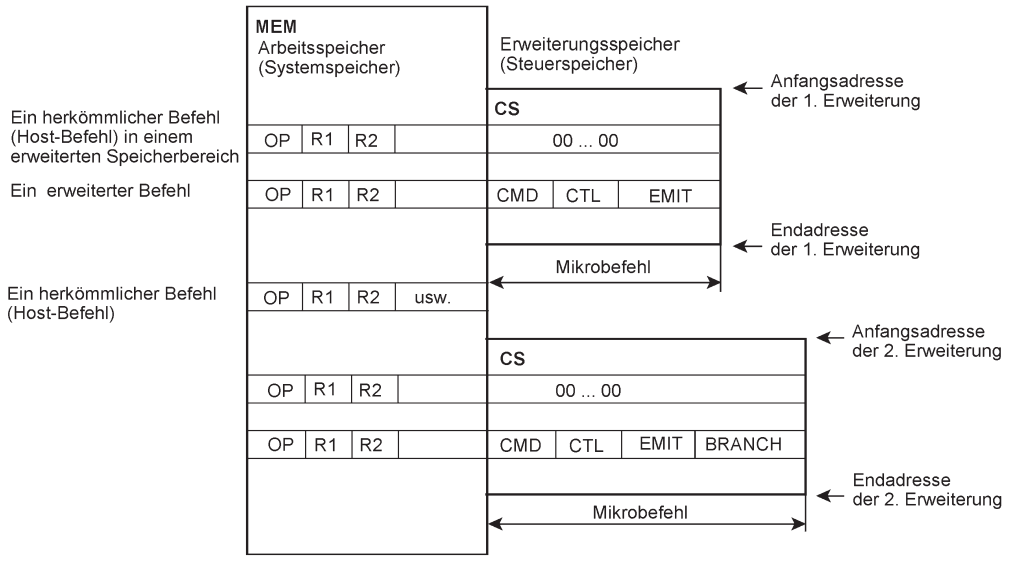


Abb. 7.42 Der Host wird mit mehreren Steuerspeichern erweitert. Jede Erweiterung unterstützt jeweils bestimmte Funktionen.

Kürzere Mikrobefehle
Nicht alle Erweiterungen passen zu allen Maschinenbefehlen. Der Mikrobefehl, der einen Maschinenbefehl erweitert, muß nur die Steuerbits, Anweisungsfelder und Direktwerte enthalten, die für diese Erweiterung von Bedeutung sind. Diese Überlegung führt auf kürzere, unterschiedlich formatierte Mikrobefehle, in deren Decodierung der Operationscode des zugehörigen Maschinenbefehls einfließt (Abb. 7.43).

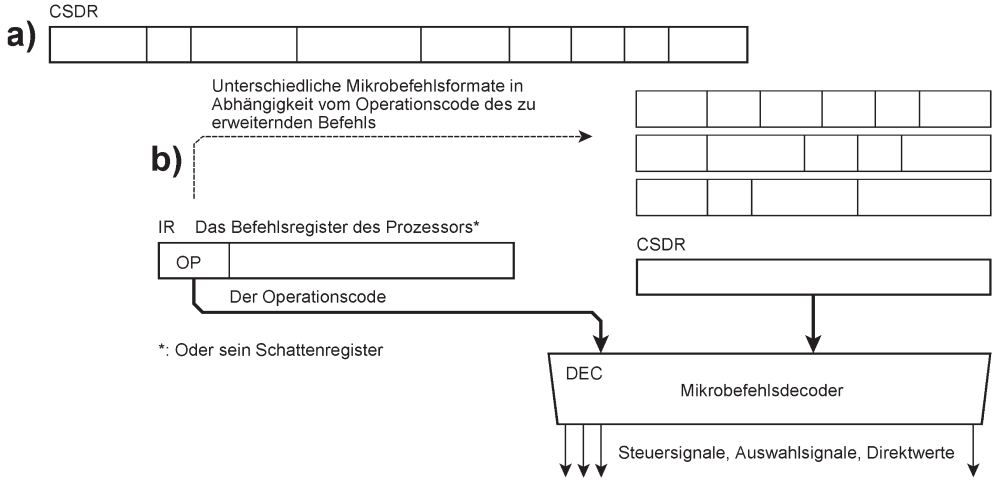


Abb. 7.43 Die Mikrobefehle verkürzen. a) Herkömmlicherweise werden alle Mikrobefehlsbits aneinandergereiht, die zur Steuerung und Parameterversorgung der Erweiterungsschaltungen benötigt werden. Das ergibt vergleichsweise lange Mikrobefehle. b) Die Mikrobefehle enthalten nur die Bits, die für die jeweilige Erweiterung benötigt werden. Da die Erweiterung zum Maschinenbefehl passen muß, kann man dessen Operationscode ausnutzen, um den Mikrobefehl im Register CSDR zu decodieren und die Signale zu gewinnen, die den Erweiterungsschaltungen zugeführt werden.

Implementierung
Es liegt nahe, solche Speicher mit Block-RAMs zu implementieren.

ROM oder RAM
Der Zusatzspeicher wird während des normalen Betriebs nur gelesen, auch bei Schreibzugriffen. Somit liegt es nahe, ihn als ROM auszuführen oder – als Block-RAM – entsprechend zu konfigurieren. Manchmal ist aber ein RAM zweckmäßiger. Ob man nur ein anfängliches Laden oder auch das Laden im laufenden Betrieb unterstützt, ist Sache der Entwurfsoptimierung.

Betriebsarten
Die Betriebsart wird in einem programmseitig ladbaren Register eingestellt. Zweckmäßige Betriebsarten sind:

- eine allgemeine Erlaubnissteuerung (Erweiterungswirkungen ein/aus),
- programmseitige Lesezugriffe zum Steuerspeicher (zu Diagnosezwecken),
- programmseitige Schreibzugriffe (zum Laden, wenn der Steuerspeicher ein RAM ist).

Abb. 7.44 veranschaulicht die Zugänglichkeit des Steuerspeichers im Speicheradreßraum. Wir brauchen zwei Zugriffswege. Abb. 7.45 zeigt, das man dieses Problem mit einem als Dual-Port Memory konfigurierten Block-RAM auf einfache Weise lösen kann.

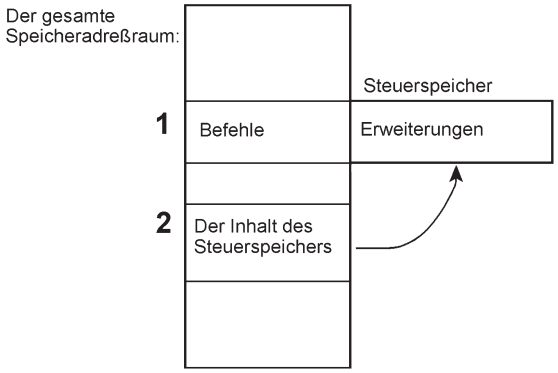


Abb. 7.44 Der Steuerspeicher im Speicheradreßraum. 1 - Befehle aus diesem Adreßbereich werden mit zusätzlichen Steuerwirkungen erweitert. 2 - in diesem Adreßbereich ist der Inhalt des Steuerspeichers programmseitig zugänglich.

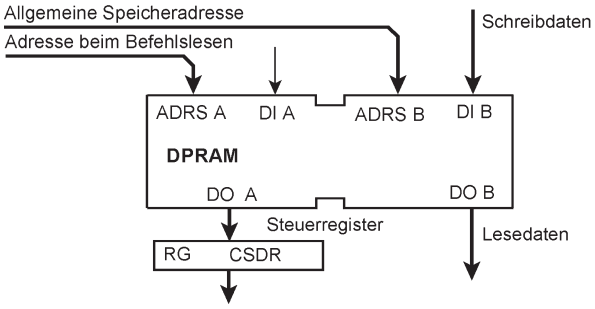


Abb. 7.45 Der Steuerspeicher als Block-RAM in Dual-Port-Konfiguration. So sind die beiden Zugriffswege auf einfache Weise zu implementieren.

Den Befehls-Cache erweitern

Auch diese Lösung liegt nahe. Voraussetzung ist, daß man die betreffenden Funktionseinheiten als IP Cores entsprechend abwandeln und erweitern kann. Gemäß Abb. 7.46 wird der Bereich des Cache, in dem sich die zu erweiternden Befehle befinden, als Non-Cacheable Region konfiguriert. Er wird dann als direkt adressierter Speicher und nicht als transparenter Cache betrieben. Beim Befehlslesen aus diesem Bereich wird parallel dazu der Steuerspeicher gelesen.

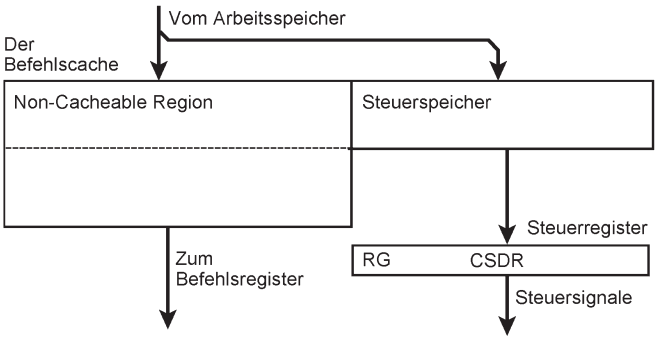


Abb. 7.46 So kann ein Befehls-Cache erweitert werden.