9. ReAl-Maschinen

ReAl-Systeme können mit herkömmlichen Prozessoren aufgebaut werden. Man kann z. B. einen Prozessor als Plattform einsetzen und durch weitere Prozessoren ergänzen, die als Verarbeitungsressourcen genutzt werden. Die Prozessoren werden üblicherweise über ein Bussystem oder über Punkt-zu-Punkt-Interfaces und Koppeleinrichtungen (Schaltverteiler, Hubs) miteinander verbunden (Abb.n 9.1, 9.2). Sie können auf einen gemeinsamen Speicheradreßraum zugreifen.

Die Plattform baut die Ressourcen-Arbeitsbereiche im Speicheradreßraum auf und veranlaßt die Prozessoren, die entsprechenden Funktionen auszuführen. Das kann z. B. durch Unterbrechungsauslösung bewirkt werden. Diese Lösung funktioniert mit herkömmlicher Hardware, hat aber den Nachteil vergleichsweise langer Latenzzeiten. Hierzu tragen vor allem bei:

- Die Interruptauslösung (mit der erforderlichen Kontextumschaltung),
- das Füllen der Caches (nach dem Schreiben in den Speicheradreßraum müssen die Caches der Prozessoren zunächst neu gefüllt werden).

Eine Abhilfe besteht darin, die Caches der Prozessoren für Schreibzugriffe von außen zugänglich zu machen. Das erfordert entsprechende Änderungen an den Prozessorschnittstellen. Es muß möglich sein, die Prozessoren als Targets anzusprechen, wobei die Zugriffsadressen die internen Caches und Pufferspeicher betreffen. Hierfür ist nur die Busanschlußsteuerung abzuwandeln (die anderen Schaltungen im Prozessor bleiben, wie sie sind).

Noch höhere Verarbeitungsleitungen lassen sich erreichen, wenn der Prozessor auch im Innern als ReAl-Maschine ausgelegt wird, die die ReAl-Befehle nicht mit herkömmlichen Maschinenbefehlen emuliert, sondern direkt ausführt. Hierzu können vorhandene Prozessorstrukturen ausgenutzt werden (vgl. die Abb.n 1.2 und 1.3). Befehlsdecodierung und Mikroprogrammsteuerung sind zu änderrn, Verarbeitungswerke, Anschlußsteuerungen, Caches, Puffer usw. bleiben im wesentlichen erhalten.

Die Alternative besteht darin, die Hardware von Grund auf aus elementaren Verarbeitungsressourcen aufzubauen. Im folgenden Beispiel ist die elementare Ressource eine Arithmetik-Logik-Einheit, die mit Registerspeichern und einem einfachen Steuerwerk (Sequencer) verbunden ist (Abb. 9.3). Der Funktionsumfang entspricht dem im Bereich der Hochleistungsprozessoren üblichen Stand der Technik (Tabelle 9.1). Die Datenstrukturen sind ganze Binärzahlen und Bitfelder. Die Unterstützung von Einzelbits, Zeichenketten, Gleitkommazahlen usw. ist aus Umfangsgründen nicht dargestellt.

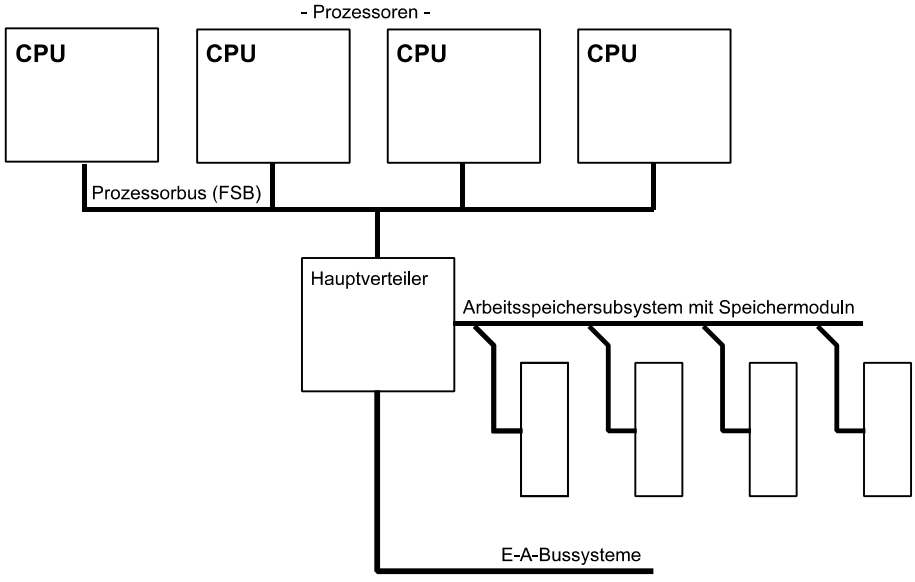

Abb. 9.1 Über ein Bussystem verbundene Prozessoren

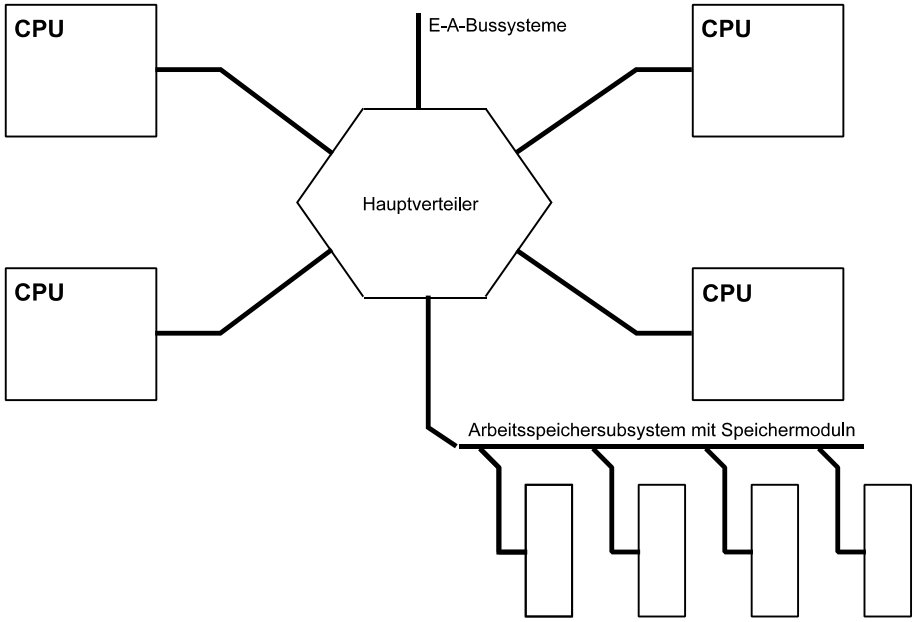

Abb. 9.2 Über Schaltverteiler verbundene Prozessoren

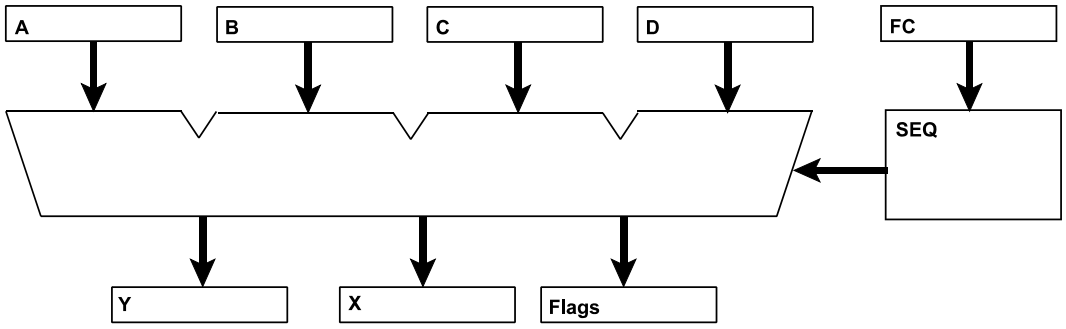

Abb. 9.3 Eine elementare Verarbeitungsressource

Ressource	Eingangsparameter		Ausgangsparameter		Funktion
ADD	2	A, B	2	X, Flags	X := A + B
SUB	2	A, B	2	X, Flags	X := A - B
MUL	2	A, B	3	X, Y, Flags	Y|X := A * B
DIV	3	A, B, C	3	X, Y, Flags	X := A|B : C, Y := A|B mod C (Rest)
NEG	1	A	2	X, Flags	X := A * (-1) (Zweierkomplement)
AND	2	A, B	2	X, Flags	X := A and B
OR	2	A, B	2	X, Flags	X := A or B
XOR	2	A, B	2	X, Flags	X := A xor B
NOT	1	A	2	X, Flags	X := not A
SHL	3	A, B, C	3	X, Y, Flags	Y|X := A nach links verschoben um C Bits, aufgefüllt mit B
SHR	3	A, B, C	3	X, Y, Flags	Y|X := A nach rechts verschoben um C Bits, aufgefüllt mit B

Ressource	Eingangsparameter		Ausgangsparameter		Funktion
SHRA	2	A, B	3	X, Y, Flags	Y|X := A arithmetiesch nach rechts verschoben um B Bits
ROTL	2	A, B	2	X, Flags	X := A nach rechts rotiert um B Bits
ROTR	2	A, B	2	X, Flags	X := A nach links rotiert um B Bits
EXTRACT	3	A, B, C	2	X, Flags	X := Bitfeld aus A, beginnend an Bit B, Länge C
DEPOSIT	4	A, B, C, D	2	X, Flags	X := A mit eingefügtem Bitfeld B. Einfügung beginnend an Bit C, Länge D
FIRSTOCC	1	A	3	X, Y, Flags	X := Bitvektor mit höchstwertiger 1 aus A (sonst Nullen), Y := Bitadresse der höchstwertigen Eins
LASTOCC	1	A	3	X, Y, Flags	X := Bitvektor mit niedrigstwertiger 1 aus A (sonst Nullen), Y := Bitadresse der niedrigstwertigen Eins
NOOCCS	1	A	2	X, Flags	X := Anzahl der Einsen in A
LOAD_A	3	A, B, C	2	X, Flags	Laden X gemäß <A + B)>; B := B + C (Autoincrement)
STORE_A	4	A, B,C, D	1	Flags	Speichern D gemäß <A + B)>; B := B + C (Autoincrement)
LOAD_X	2	A, B, C	2	X, Flags	Laden X gemäß <A + (B * C)>; B := B + 1 (Autoincrement)
STORE_X	2	A, B, C, D	1	Flags	Speichern D gemäß <A + (B * C)>; B := B + 1 (Autoincrement)
FORLOOP	3	A, B, C	2	X, Flags	Anfänglich: X := A; dann X := X + B, solange X $\leq$ C

Tabelle 9.1 Der Funktionsumfang einer elementaren Verarbeitungsressource

Anmerkungen:

1. Y|X und A|B = höherwertiges Wort in Y oder A, niederwertiges Wort in X oder B.
2. Ladefunktionen (LOAD-A/X) stellen ein Ergebnis in X bereit, das an die eigentlichen Verarbeitungsressourcen geliefert wird (Verkettung).
3. Speicherfunktionen (STORE-A/X) speichern einen in D abgelegten Wert, der von den eigentlichen Verarbeitungsressourcen dort abgeliefert wurde (Verkettung).
4. LOAD_A, STORE_A = Laden und Speichern gemäß Adreßrechnung Basis + Displacement mit nachträglicher Erhöhung der Displacementangabe. Basis in A, Displacement in B, Erhöhung in C. Unterstützt Zugriffe zu aufeinanderfolgenden gleich langen Datenstrukturen der Länge C.
5. LOAD_X, STORE_X= Laden und Speichern gemäß Indexangabe in B mit nachträglicher Erhöhung des Indexwertes um 1. Unterstützt Zugriffe zu aufeinanderfolgenden gleich langen Datenstrukturen der Länge C auf Grundlage einer Indexangabe (= laufende Nummer 0, 1, 2, 3 usw.). Basisadresse in A, Länge der Datenstruktur in C.
6. FORLOOP = FOR X := A TO C STEP B.

Ersichtlicherweise kommt man mit maximal 4 Operanden und 3 Ergebnissen aus. Es genügt somit eine Parameteradresse von 3 Bits. Hiermit kann man auch die weiteren in modernen Hochleistungsprozessoren üblichen Operationen (über Gleitkommazahlen, Einzelbits usw.) unterstützen.

Jede Verarbeitungsfunktion in Tabelle 9.1 entspricht – aus Sicht der Systemarchitektur – wenigstens einem Ressourcentyp (es gibt Abwandlungen in Hinsicht auf die Operandenlängen usw.). Ein s-Operator stellt eine Hardware gemäß Abb. 9.2 auf eine bestimmte Funktion ein.

Verarbeitungs- und Speicherzugriffsfunktionen
Im Beispiel sind hierfür jeweils besondere Ressourcen vorzusehen, die ggf. entsprechend zu verketten sind. Die universellen Verarbeitungseinrichtungen gemäß Abb. 9.3 sind so auslegt, daß sie beide Aufgaben übernehmen können. Anwendungsbeispiel: eine Vektoroperation C[i] := A[i] *op* B[i] (komponentenweise Verknüpfung) erfordert fünf Ressourcen: drei zur Adressierung der Vektoren A, B, C, eine für die Operandenverknüpfung und eine zum Erkennen des Ablaufendes (FOR-Schleife).

Parallelisierung durch Aufrollen von Schleifen
Es ist bekannt, das mehrfache Durchlaufen einer Schleife durch entsprechend mehrfache Parallelausführung der Operationen des Schleifenkörpers zu ersetzen (Loop Unrolling). Ist die Durchlaufzahl (1) von Anfang an (zur Compilierzeit) bekannt und (2) nicht allzu groß, so bereitet dies in einer ReAl-Maschine keine besonderen Schwierigkeiten.

Beispiel:

FOR n = 1 to 20

– Schleifenkörper –

NEXT N

wird parallelisiert (aufgerollt), indem die zur Implementierung des Schleifenkörpers erforderliche Ressourcenausstattung 20mal angefordert und entsprechend mit Parametern versorgt wird.

Stehen nicht genügend Ressouren zur Verfügung, so ist nur eine abschnittsweise Parallelausführung möglich.

Beispiel: Im Ausdruck

FOR n = 1 TO 20

– Schleifenkörper –

NEXT N

können nur vier Schleifenkörperfunktionen parallel unterstützt werden. Hierzu ist die Schleife umzuformen:

FOR n = 1 TO 20 STEP 5

1. Schleifenkörper | 2. Schleifenkörper | 3. Schleifenkörper | 4. Schleifenkörper

NEXT N

Die Ressourcenausstattung des Schleifenkörpers wird viermal angefordert: die Schleife wird in 5 Durchläufen ausgeführt.

Ist die Durchlaufzahl zur Compilierzeit nicht bekannt (Beispiel: FOR n = 1 TO x), so ist diese einfache Form des Aufrollens nicht möglich. Ein Ausweg besteht darin, eine gewisse Anzahl an Verarbeitungsressourcen zwecks Parallelausführung pauschal bereitzustellen und deren Nutzung über entsprechend ausgebildete Iterator- und Speicherzugriffsressourcen zu steuern (Abb.n 9.4, 9.5).

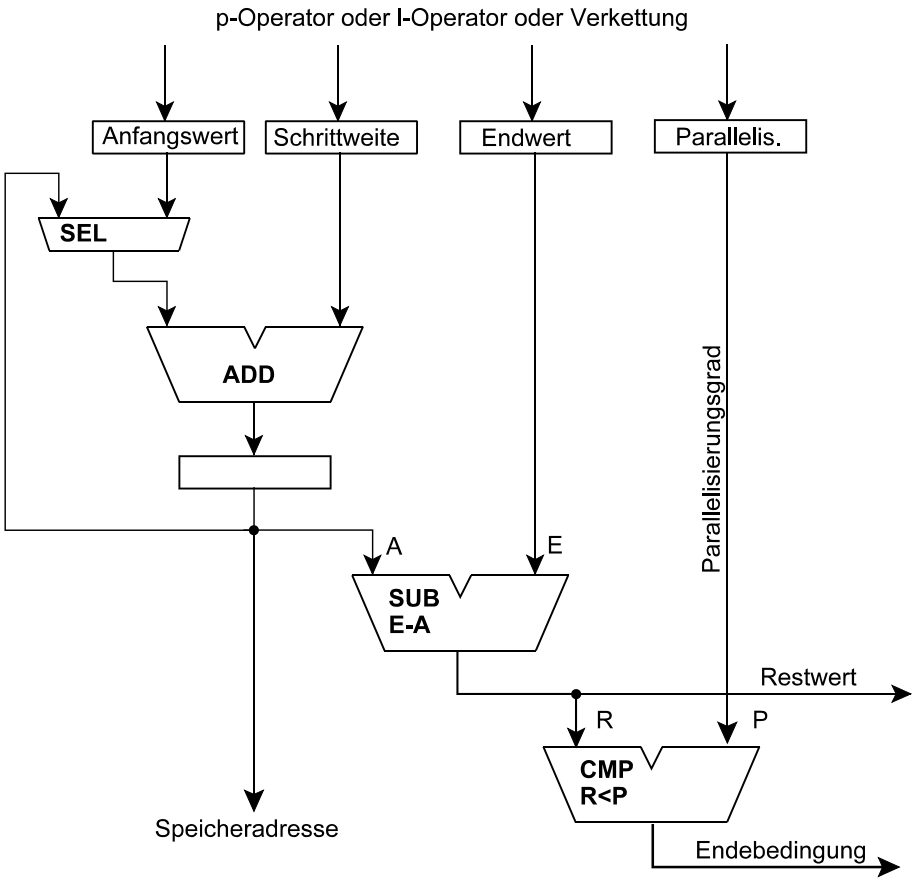

Abb. 9.4 Iterator zur Unterstützung der Parallelverarbeitung

Der Iterator entspricht im wesentlichen der Ausführung gemäß Abb. 2.16. Die Anzahl der parallel unterstützten Verarbeitungsressourcen wird als Parallelisierungsgrad P bezeichnet (P = 1: 1 Ressource, P = 2: 2 Ressourcen usw.). Die Nutzung wird üblicherweise vom Compiler gesteuert. Er ordnet z. B. einer bestimmten Schleife 4 Verarbeitungsressourcen zu (P = 4) und korrigiert die Schrittweite entsprechend:

Aus FOR n = 1 TO x wird FOR n = 1 TO x STEP P · x.

Beispiel:

Aus FOR n = 1 TO 14 wird mit P = 4 FOR n = 1 TO 14 STEP 4.

Werte für n (jede Verarbeitungsresource erledigt jeweils einen dieser Werte):

im 1. Durchlauf: 1, 2, 3, 4
im 2. Durchlauf: 5, 6, 7, 8
im 3. Durchlauf: 9, 10, 11, 12
im 4. Durchlauf: 13, 14

Ersichtlicherweise sind im letzten Durchlauf nicht alle vier Verarbeitungsressourcen beschäftigt. Um den letzten Durchlauf zu erkennen, wird der aktuelle Wert vom Endwert subtrahiert: Restwert R = E - A. Ist der Restwert kleiner als der Parallelisierungsgrad (R < P), so ist das Schleifenende erreicht. Ist $R \leq 0$, so wird dei Schleife verlassen. Ist R > 0, so handelt es sich um einen letzten Durchlauf, in dem nicht alle Verarbeitungsressourcen beschäftigt sind.

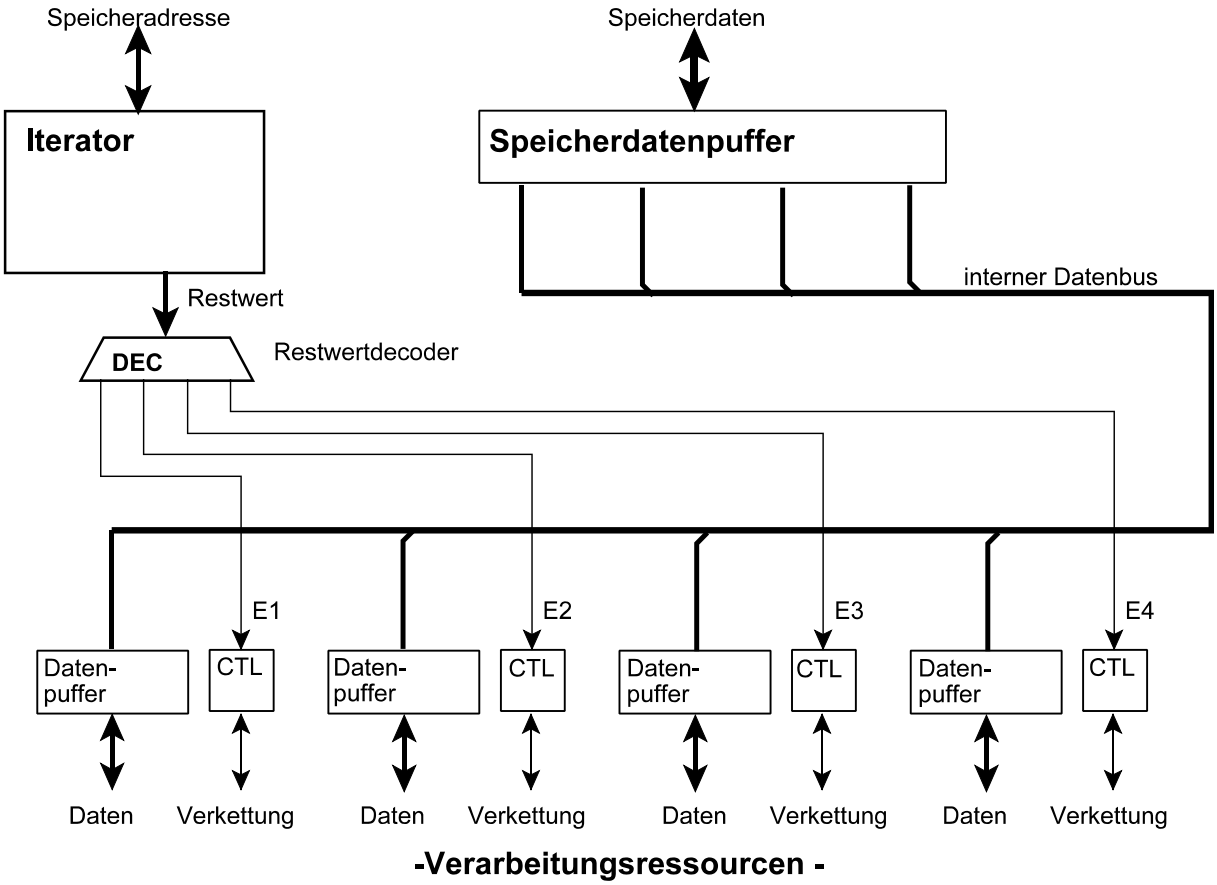

Abb. 9.5 Speicherzugriffsressource zur Unterstützung der Parallelverarbeitung

Da die Aktivierung und Versorgung der parallel arbeitenden Verarbeitungsressourcen koordiniert werden muß, ist es zweckmäßig, alle betreffenden Verarbeitungsressourcen einer einzigen entsprechend ausgelegten Speicherzugriffsressource nachzuschalten. Die Speicherzugriffsressource muß – wie auch das gesamte Speichersubsystem – die Speicherbandbreite unterstützen können, die nötig ist, um die parallel arbeitenden Verarbeitungsressourcen mit Daten zu versorgen und die anfallenden Ergebnisse abzutransportieren. Beispiel: wenn vier Ressourcen mit einer Zugriffsbreite von 64 Bits angeschlossen sind, werden die Speicherzugriffe mit einer Zugriffsbreite von 256 Bits ausgeführt. Datenpuffer, die Zugriffe mit verschiedener Breite auf verschiedenen Adressen sammeln und in Zugriffe mit größerer Breite umsetzen können, sind in modenen Hochleistungsprozessoren enthalten. Sie entsprechen somit dem Stand der Technik und müssen nicht näher beschrieben werden. Die Speicherzugriffsressource enthält einen Iterator gemäß Abb. 9.4. Jeder Verarbeitungsressource ist ein Datenpuffer und eine Verkettungssteuerung zugeordnet. Ein Verkettungsvorgang wird ausgelöst, wenn Daten bereitstehen (Lesen) oder abzuholen sind (Schreiben). Eine Verkettungssteuerung wird nur dann wirksam, wenn ihr Erlaubniseingang aktiv ist. Die Erlaubniseingänge werden vom Restwertdecoder erregt. Es handelt sich um eine kombinatorische Schaltung, die Erlaubnissignale (E1...E4) für die Verkettungssteuerungen abgibt (Tabelle 9.2). Handelt es sich nicht um den letzten Schleifendurchlauf, so sind alle vier Verkettungssteuerungen aktiv. Im letzten Schleifendurchlauf richtet sich die Aktivierung nach dem Restwert.

Restwert	E1	E2	E3	E4
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
≤ 4	1	1	1	1

Tabelle 9.2 Zur Wirkungswweise des Restwertdecoders

ReAl-Maschinen
Eine ReAl-Maschine besteht aus:

- Der Plattform,
- Verarbeitungsressourcen,
- gemeinsamen Speichermitteln und E-A-Einrichtungen.

Diese Einrichtungen können z. B. über einen Universalbus gekoppelt sein (Abb. 9.6). In einer abgewandelten Konfiguration (Abb. 9.7) sind die gemeinsamen Speichermittel an die Plattfom angeschlossen. Der Universalbus kann in mehrere Bussysteme aufgelöst werden, z. B. in einen Operandenbus und einen Ergebnisbus.

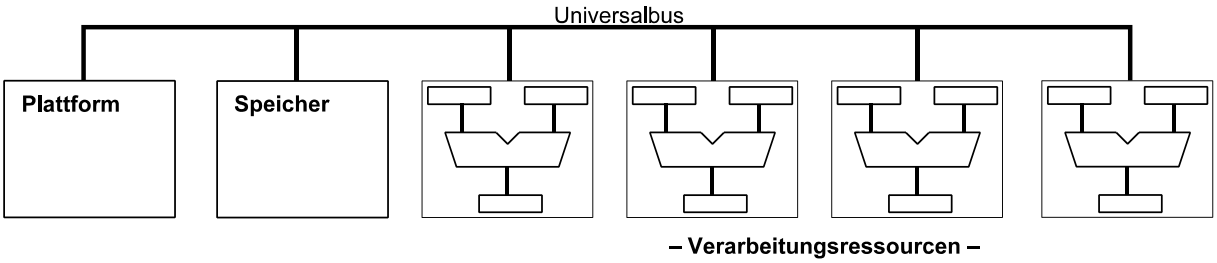

Abb. 9.6 ReAl-Machine mit Universalbus

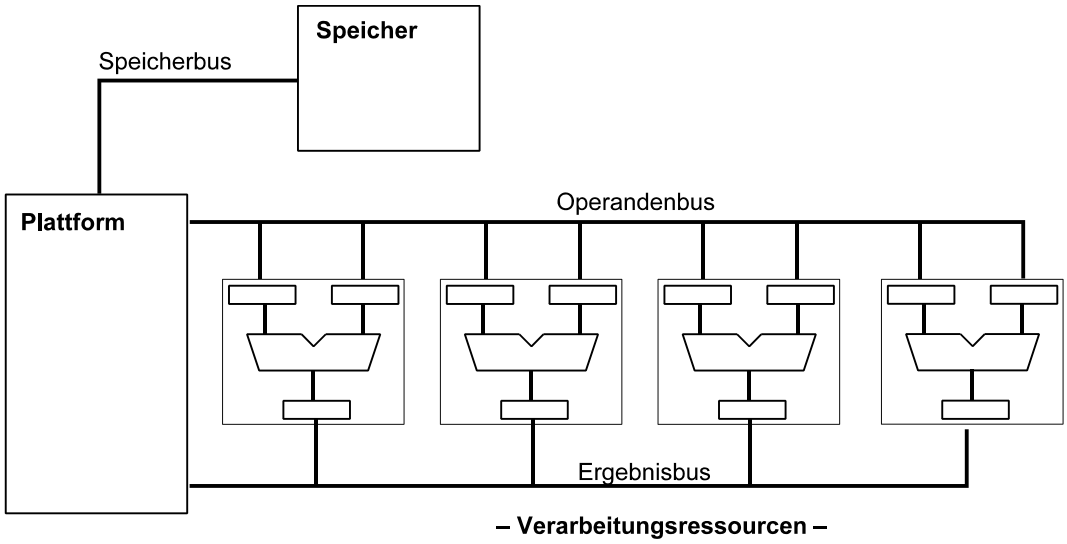

Abb. 9.7 ReAl-Maschine mit zwei Bussystemen

Ein Verarbeitungswerk kann zu einer Zeit mehr als eine Ressource verkörpern. Hierzu sind entsprechend adressierbare Speicheranordungen für die Parameter und Ergebnisse vorzusehen, die über eine Ressoucenadreßangabe adressiert werden (Abb. 9.8).

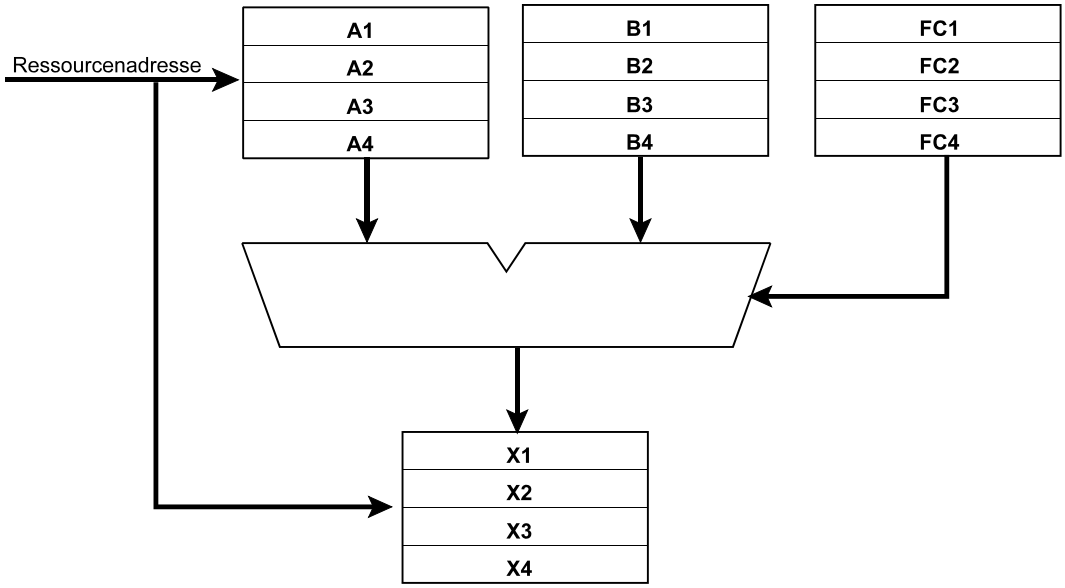

Abb. 9.8 Ein Verarbeitungswerk mit adressierbaren Speicheranordnungen kann zu einer Zeit mehr als eine Ressource verkörpern

Zentrale und autonome Steuerung
Kleinere ReAl-Maschinen können zentral gesteuert werden. Die Plattform steuert hierbei alle Speicherzugriffe und Informationstransporte. Nur die y-Operatoren werden autonom in den Verarbeitungsressourcen erledigt. Die Verkettung wird von der Plattform emuliert. Solche Maschinen können z.B. auf Grundlage von Bussystemen aufgebaut werden, an die alle Verarbeitungsressourcen als Slaves (Targets) angeschlossen sind.

Hochleistungsmaschinen erfordern eine autonome Steuerung der Speicherzugriffs- und Verkettungsfunktionen. Die Verarbeitungsressourcen gemäß Abb. 9.3 sind hierzu mit entsprechenden Anschlußsteuerschaltungen auszurüsten (Abb. 9.9) und über universelle Multimaster-Bussysteme, Schaltverteiler o. dergl. miteinander zu verbinden (vgl. die Abb.n 9.1 und 9.2).

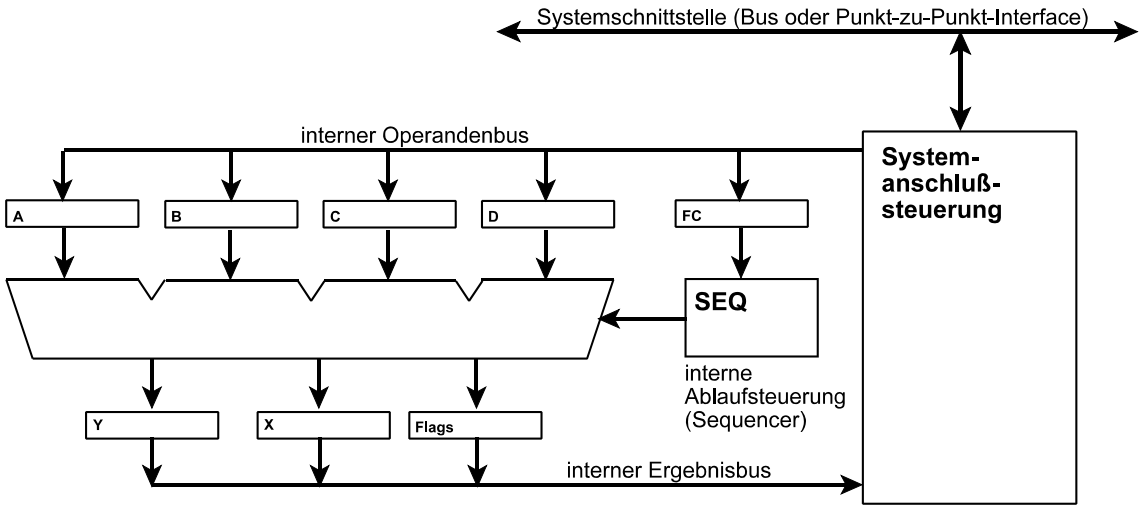

Abb. 9.9 Universelle Verarbeitungsressouce mit Systemanschlußsteuerung

Die Plattform
Es gibt verschiedene Möglichkeiten, eine Plattform aufzubauen:

- Nutzung eines herkömmlichen Prozessors,
- Auslegung als mikroprogrammiertes Steuerwerk,
- Auslegung als Sammlung elementarer Ressourcen (vgl. Abb. 5.2),
- Aufbau aus universellen Verarbeitungsressourcen mit einer anfänglichen (nach dem Einschalten bzw. Rücksetzen) festen Zusammenschaltung.

Demgemäß können die Funktionen der Plattform gesteuert werden durch:

1. Herkömmliche Maschinenbefehle,
2. Mikrobefehle,
3. elementare Maschinenbefehle für Verwaltungsaufgaben (Regiebefehle), z. B. Laden, Speichern, Verzweigen, Unterprogrammruf, Rückkehr, Unterbrechungssteuerung,
4. ReAl-Operatoren,
5. beliebige Kombinationen von 1. bis 4.

ReAl-Systeme können mehrere Plattformen enthalten.

Eine elementare Plattform als Ressourcensammlung
Hier soll untersucht werden, wieviele Resourcenadressen für eine Plattform beiseite zu setzen sind, die man sich als Zusammenfassung der in Kapitel 5 veranschaulichten Auslegungen vorstellen kann. Abb. 9.10 gibt einen Überblick über die Parameter.

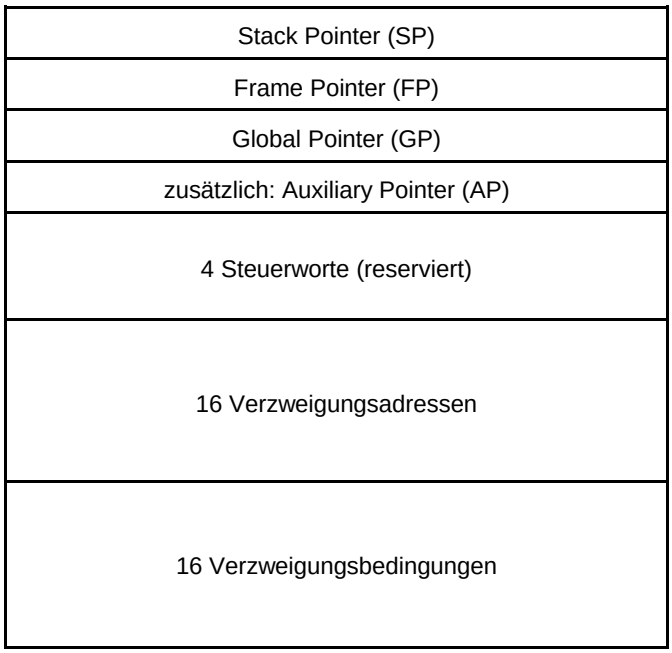

Abb. 9.10 Parameter einer elementaren Plattform. Die Verzweigungsvorkehrungen werden auch zum Unterprogrammruf verwendet

Aus Abb. 9.10 ergeben sich 40 Ziele (Registeradressen) für p-Operatoren. Mit 40 Adressen entspricht die Plattform 5 Resourcen mit je 8 Parametern oder einer flachen Ressourcenadresse von 6 Bits.

In y-Operatoren können die 40 Adressen z. B. wie folgt belegt werden:

- 16 Ziele für y-Operatoren zwecks Verzweigung,
- 16 Ziele für y-Operatoren zwecks Unterprogrammruf,
- 8 Ziele für y-Operatoren mit sonstigen Wirkungen (Rückkehr aus Unterprogramm usw.).

Aus den dargestellten Grundsatzlösungen und Varianten können ReAl-Maschinen gleichsam nach dem Baukastenprinzip konfiguriert werden. Die Auswahl ergibt sich – wie in der Rechnerarchitektur üblich – auf Grundlage von Kosten-Nutzen-Betrachtungen und Nutzungshäufigkeiten.

Beispiel:

- Eine Auslegung X bringt Vorteile bei Anwendungen der Art A,
- eine Auslegung Y bringt Vorteile bei Anwendungen der Art B.

Kommen Anwendungen der Art A häufiger vor als solche der Art B, wird man die Auslegung X wählen und umgekehrt.