

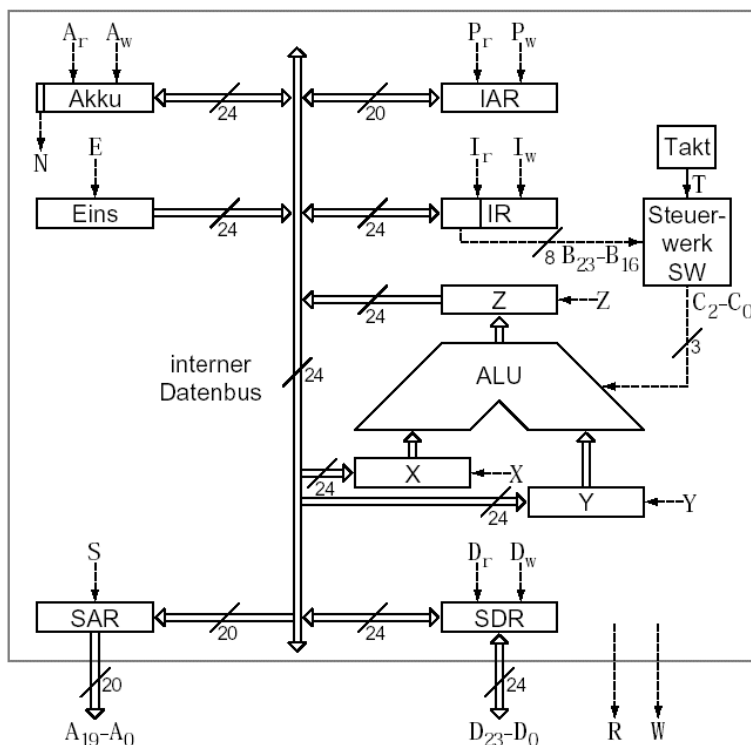
CISC & RISC

Zwei Techniken zur Implementierung von Befehlen im Rechner

- Direkt durch Hardware → aufwendige Schaltnetze und Schaltwerke bei größer Anzahl von Befehlen (200-300)
- Mikroprogramme als Befehlsinterpret → Mikroprogrammspeicher im Steuerwerk, der neu geladen werden kann → verschiedene Befehlssätze können implementiert werden



Mikroprogrammsteuerwerke



Architektur der MIMA
(1. Übungsblatt)



Mikroprogrammsteuerwerke

Mikrobefehlsformat:

A _r	A _w	X	Y	Z	E	P _r	P _w	I _r	I _w	D _r	D _w	S	C ₂	C ₁	C ₀	R	W	0 0	Folgeadresse F
27			24				20			16				12				9 8	0

Jedes Bit des Mikrobefehls entspricht einem Steuersignal

- **Horizontale Mikroprogrammierung:** Steuerungsfeld im Mikrobefehl für jedes Steuersignal im Rechner → Kein Dekoder
- **Vertikale Mikroprogrammierung:** Zusammenfassung von Steuersignalen zu Feldern.



Mikroprogrammsteuerwerke

Fetch-Phase bei der MIMA:

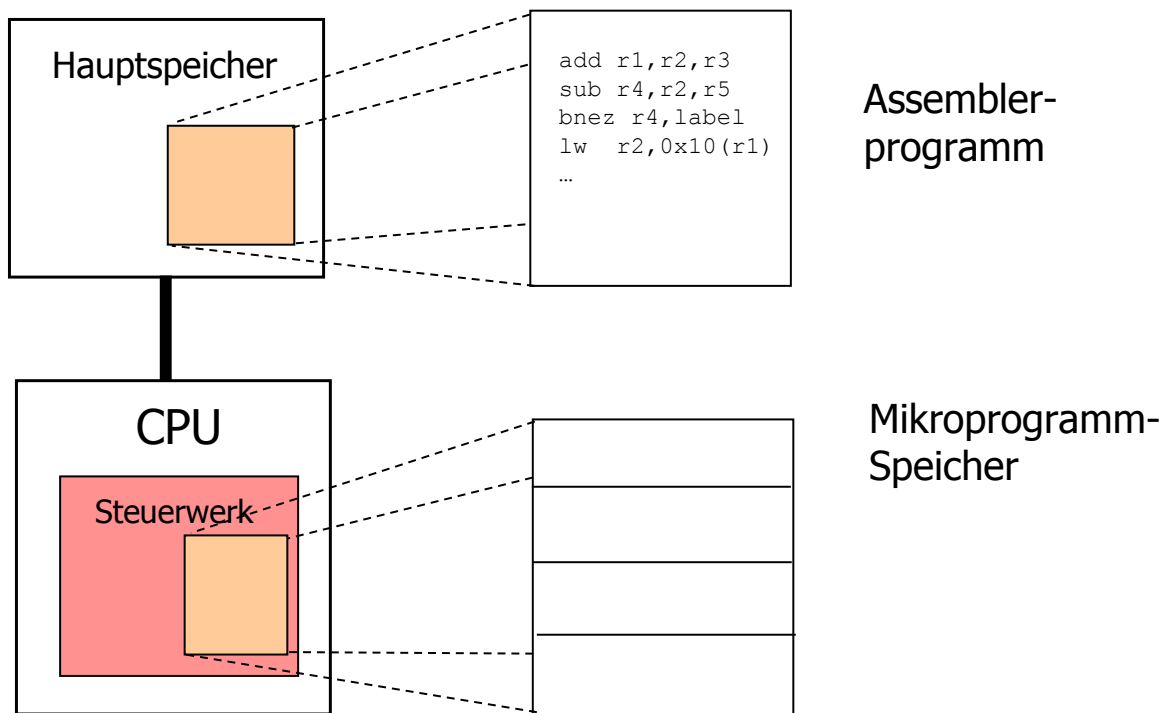
1. Takt: IAR -> SAR; IAR -> X; R = 1
2. Takt: Eins -> Y; ALU auf addieren; R = 1
3. Takt: ALU auf addieren; R = 1
4. Takt: Z -> IAR
5. Takt: SDR -> IR

Mikroprogramm der Fetch Phase

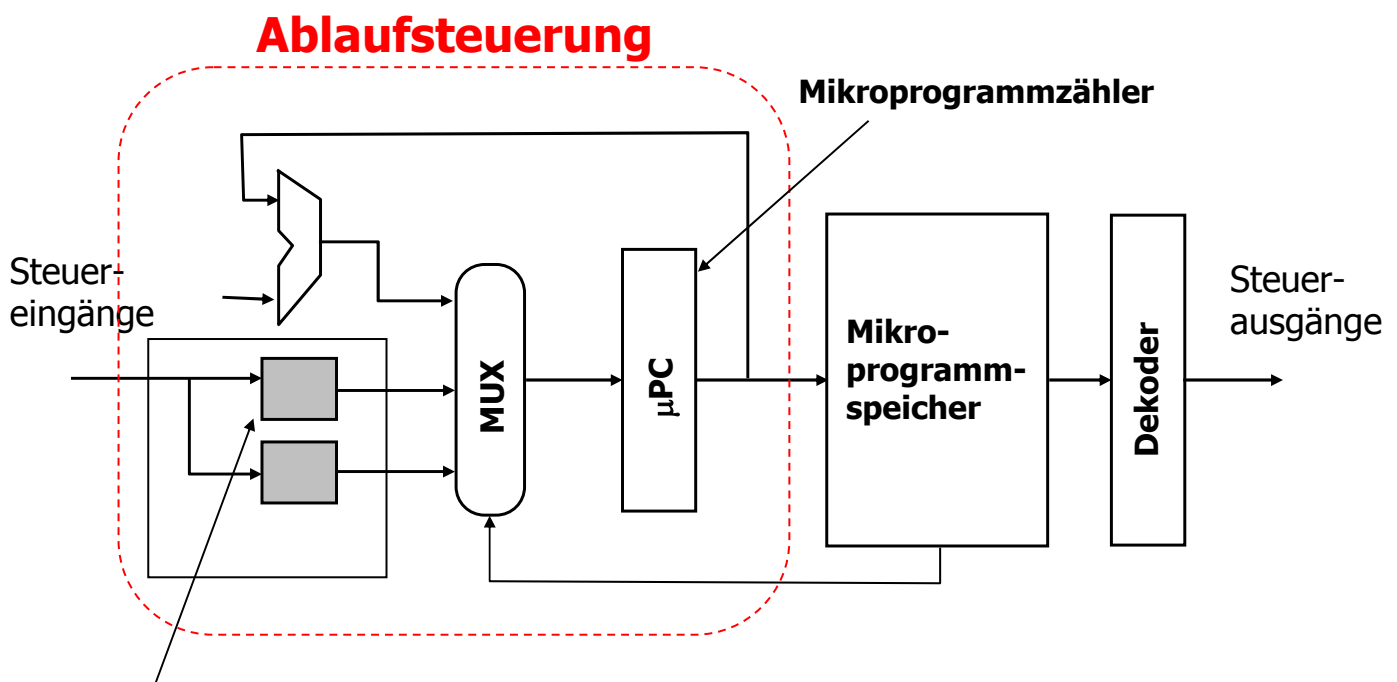
```
0010 0001 0000 1000 1000 0000 0001
0001 0100 0000 0000 1000 0000 0010
0000 0000 0000 0001 1000 0000 0011
0000 1010 0000 0000 0000 0000 0100
0000 0000 1001 0000 0000 0000 0101
```



Prinzip der Mikroprogrammierung



Implementierung des Steuerwerks



Sprungspeicher



Vorteile und Nachteile

➤ Vorteile der Mikroprogrammierung:

- Mehrere Befehlssätze auf einem Rechner → Anpassung des Befehlssatz an der Anwendung
- Mehrere Rechnertypen mit dem gleichen Befehlssatz

➤ Nachteile der Mikroprogrammierung:

- Aufwendig
- Langsam



CISC (complex instruction set computers)

Gründe dafür:

- Ausführung komplizierter Befehle ist immer noch schneller als die Ausführung von Programmen gleicher Funktion
- Mikroprogrammierung begünstigt komplizierte Befehle
- komplizierte Befehle führen zu kurzen Programmen
- Umfang des Befehlssatzes wird oft als Werbeargument verwendet
- Unterstützung höherer Programmiersprachen durch komplizierte Befehle (Direkte Abbildung: *Sprachkonstrukt* → *Befehl*)



CISC (complex instruction set computers)

Gründe dafür:

- Unterstützung von Compilern durch entsprechende Befehle
- Unterstützung spezieller Einsatzgebiete

Fazit:

**Entwicklung von Hardware,
Programmiersprachen und Einsatzgebieten
begünstigt „komplizierte“ Befehle.**



CISC (complex instruction set computers)

Gründe dagegen:

- Schnellere Hauptspeicher und die Verwendung von Cache-Speichern beschleunigen die Programmausführung
- Mikroprogramme wurden immer umfangreicher; Verlängerte Entwurfszeit, Komplexe Steuerwerke (> 50% der Chipfläche)
- Nur relativ kleine Teile des großen Befehlssatzes werden häufig benutzt
- Größere Fehlerhäufigkeit auf der Mikroprogrammebene
- Schwieriger Compilerbau



CISC (complex instruction set computers)

- **Systemprogramme in XPL auf IBM/360:**

90 % aller ausgeführten Befehle: 10 verschiedene Befehle

95 % aller ausgeführten Befehle: 21 verschiedene Befehle

99 % aller ausgeführten Befehle: 30 verschiedene Befehle

- **COBOL-Programme auf IBM/370:**

90,28 % aller ausgeführten Befehle: 26 verschiedene Befehle

99,08 % aller ausgeführten Befehle: 48 verschiedene Befehle

(nur 84 verschiedene Befehle wurden überhaupt benutzt)



Limitationen der CISC Architekturen

- **Befehlsausnutzung (80/20 Regel):**

viele mächtige Befehle, komplexes Befehlsformat,
Mikroprogrammierung, nur 20 % der Befehle werden
überwiegend benutzt

- **Kritisches Problem: Anzahl der Zyklen pro
Instruktion (CPI)**

bei allen heutigen CISC Architekturen ist $CPI \gg 2$



Prozentualer Anteil von Anweisungen in Hochsprachenprogrammen

Großteil der in Hochsprachen verwendeten Anweisungen ist sehr einfach:

Anweisung	Mittlerer zeitlicher Anteil
Zuweisung	47 %
if	23%
call	15 %
loop	6 %
goto	3 %
Andere	7 %



RISC (reduced instruction set computers)

Grundprinzipien:

- Viel benutzte einfache Befehle so schnell wie möglich machen (Ausführung möglichst in einer Taktphase. Keine Mikroprogrammierung mehr, Befehls-Pipeline)
- Der größte Teil der Arbeit soll durch optimierende Compiler zur Übersetzungszeit erledigt werden
- Operanden werden nach Möglichkeit in großen Registersätzen gehalten → schneller Zugriff → schnelle Verarbeitung
- Einheitliche Befehlsformate → schnelle Decodierung
- Pipelining anwenden, so gut es geht



RISC (reduced instruction set computers)

Entwurfsziele:

- Ausführung jedes Befehls in einem Taktzyklus
(Befehl \approx bisheriger Mikrobefehl bei CISC)
- Alle Befehle gleich lang:
Decodierschaltung wird einfacher
Programme länger, aber Ausführungszeit kürzer
- Nur Load-Store und Register-Register-Befehle:
weniger Adressierungsarten → schnelle Ausführung
- Koprozessorarchitektur für komplexe Befehle



RISC (reduced instruction set computers)

Keine Entwurfsziele sind z. B.:

- Unterstützung von Gleitkomma-Arithmetik
- Unterstützung von Betriebssystemfunktionen



Zielvorstellungen für RISC-Rechner

- Ein-Zyklus-Befehle
- Einheitliches Befehlsformat
- Wenige Maschinenbefehle
- Load/Store-Architektur
- Großer Registersatz
- Verzicht auf Mikroprogrammierung
- 32-Bit-Architektur
- Pipeline-gerechter Maschinenbefehlssatz (gleiche Befehlsausführzeiten)
- Keine Unterstützung für Betriebssystem und Gleitkomma-Arithmetik



Forderungen an RISC-Systeme

- Mindestens 75% aller Befehle sind Ein-Zyklus-Befehle
- Einheitliche Länge aller Befehle entsprechend der Datenbusbreite
- Nicht mehr als 128 Befehle
- Nicht mehr als 4 Befehlsformate
- Nicht mehr als 4 Adressierungsarten
- Load/Store-Architektur
- Festverdrahtete Steuereinheit, keine Mikroprogrammierung
- Mindestens 32 allgemein verwendbare Register



RISC-Rechner aus heutiger Sicht

Geblieben ist von der RISC-Idee im wesentlichen:

- das Befehlspipelining
- die Load/Store-Architektur
- ein großer Registersatz: 32 allgemeine und 32 Gleitpunkt-Register
- ein einheitliches Befehlsformat
- die Verwendung weniger Adressierungsarten
- der Verzicht auf Mikroprogrammierung



RISC & CISC

CISC	RISC
Komplexe Befehle, Ausführung in mehreren Taktzyklen	Einfache Befehle, Ausführung in einem Taktzyklen
Jeder Befehl kann auf den Speicher zugreifen	Nur Lade- und Speicherbefehle greifen auf den Speicher zu
Wenig Pipelining	Intensives Pipelining
Befehle werden von einem Mikroprogramm interpretiert	Befehle werden durch festverdrahtete Hardware ausgeführt
Befehlsformat variabler Länge	Alle Befehle mit fester Länge
Die Komplexität liegt im Mikroprogramm	Die Komplexität liegt im Compiler
Einfacher Registersatz	Mehrere Registersätze



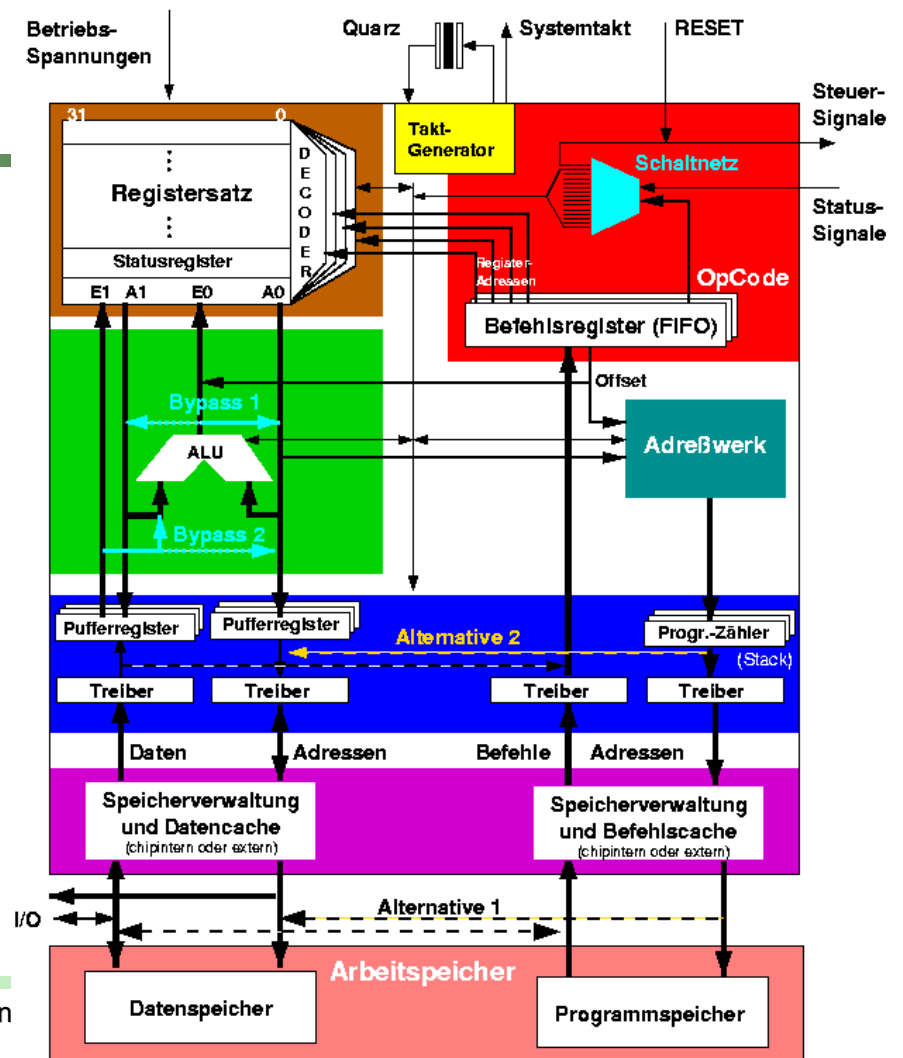
Vergleich CISC & RISC

Vergleich von drei typischen CISC-Rechnern mit den ersten drei RISC-Rechnern [Tanenbaum]:

	IBM 370/168	CISC VAX 11/780	Xerox Dorado	IBM 801	RISC Berkeley RISC I	Stanford MIPS
Fertigstellungsjahr	1973	1978	1978	1980	1981	1983
Instruktionen	208	303	270	120	3	55
Mikrocodegröße	54k	61k	17k	0	0	0
Instruktionsgröße	2-6 Bytes	2-57 Bytes	1-3 Bytes	4 Bytes	4 Bytes	4 Bytes
Operationsmodell	Reg-Reg Reg-Mem Mem-Mem	Reg-Reg Reg-Mem Mem-Mem	Stack	Reg-Reg	Reg-Reg	Reg-Reg



Aufbau eines RISC-Prozessors



Aufbau eines RISC-Prozessors

Havard Architektur:

getrennter Programm- und Datenspeicher, deshalb
zwei Adress- und Datenbusse

→ paralleles Holen von Operanden und Instruktionen

Vereinfachende Varianten:

1. zwei getrennte Bussysteme bis zu den Cache-Speichern, jedoch nur ein Arbeitsspeicher (niedrigere Kosten)
2. nur ein Bussystem wie bei Standard-Mikroprozessoren



Aufbau eines RISC-Prozessors

Systembusschnittstelle:

enthält Registerblocks sowohl für Daten als auch für
Adressen (gleichzeitiges Lesen eines Datums und
Zwischenspeichern eines Ergebnisses)

Befehlszähler:

ist manchmal als Hardware-Stack ausgebildet
(beschleunigt Unterprogrammaufrufe)



Aufbau eines RISC-Prozessors

Steuerwerk:

- festverdrahtet
- Das Befehlsregister als Warteschlange (FIFO) realisiert
- Für jede Pipeline-Stufe ist dort ein Register vorhanden
- Die OpCodes jeder Stufe können vom Schaltnetz des Steuerwerks ausgewertet werden

Registersatz:

- besteht aus einer großen Zahl von Registern
- erlaubt gleichzeitige Auswahl von 3 bis 4 Registern (z. B. 4 Port Registersatz, gleichzeitiges Schreiben (E0, E1) und Lesen (A0, A1) von jeweils 2 Registern)



Aufbau eines RISC-Prozessors

Rechenwerk:

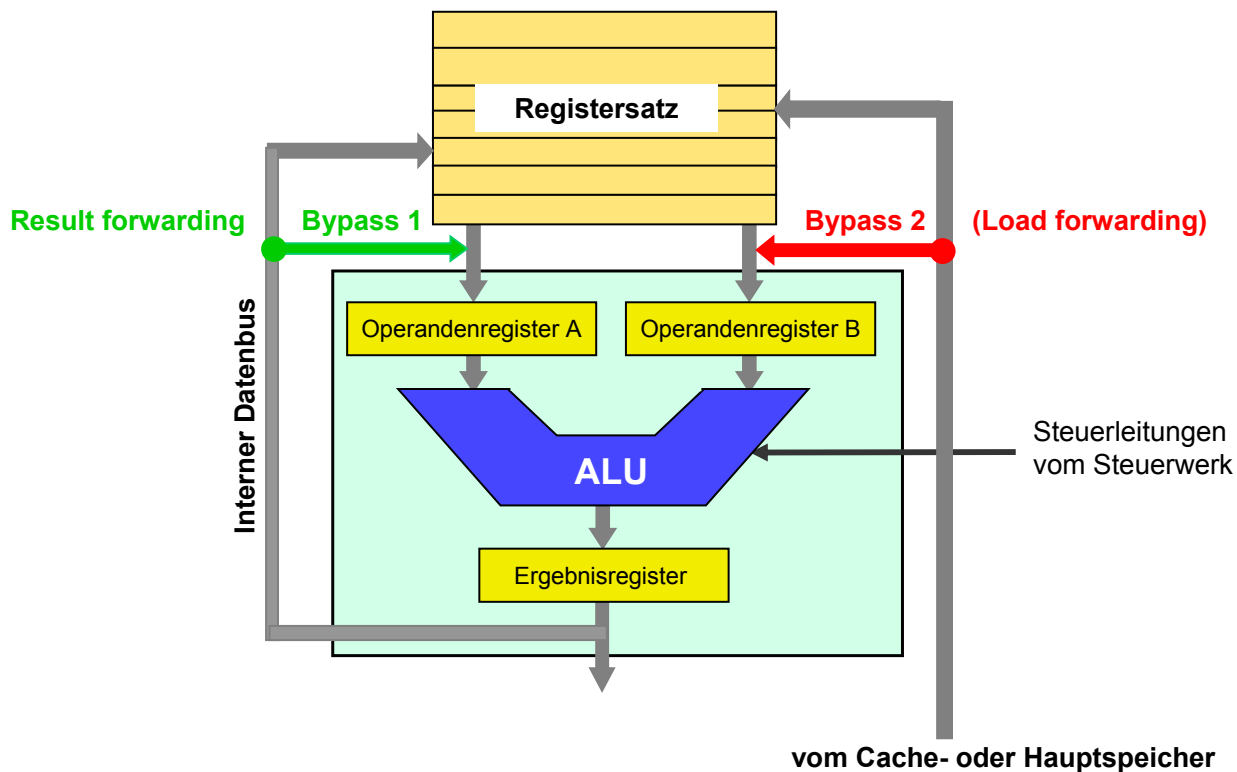
besitzt eine Load/Store-Architektur. Die Operanden werden über 2 Operandenbusse aus dem Registersatz herbeigeführt, das Ergebnis (noch im selben Taktzyklus) über den Ergebnisbus in den Registersatz geschrieben.

Normalerweise gibt es keine direkte Verbindung zwischen ALU und Systemdatenbus, Datentransfer läuft über die Register (Load/Store-Architektur)

Ausnahme: Register-Bypasses zur Vermeidung von Pipeline-Hemmnissen (forwarding techniques)



Bypass-Techniken



Unterschiede zwischen RISC- und CISC-Prozessoren

• Bypass 1

Folgen in der Pipeline zwei Befehle direkt hintereinander, bei denen das Ergebnis der Vorgänger-Operation als Operand der Nachfolger-Operation benötigt wird

- Zwischenspeichern im Registersatz würde den Ablauf verzögern
- Bypass 1 erlaubt gleichzeitig zum Abspeichern des Ergebnisses auch dessen direkte Rückkopplung zum Eingang der ALU (result forwarding)



• Bypass 2

Wurde ein Operand einer Pipeline-Operation erst im unmittelbar vorangehenden Taktzyklus vom Speicher in den Registersatz übertragen

- der Weg über den Registersatz würde ebenfalls eine Verzögerung bedeuten
- Bypass 2 erlaubt ein aus dem Speicher in ein Register geladenen Operanden gleichzeitig auch an den ALU-Eingang zu führen (load forwarding)



Befehlssatz und Adressierungsarten bei RISC-Prozessoren

RISC II der Universität Berkeley: 138 Register, 39 Befehle

6 Adressierungsarten:

- Unmittelbare Adressierung:
 $\langle \text{Mnemo} \rangle \# \langle \text{Operand} \rangle \quad \text{EA} = (\text{PC})$
- Explizite Register-Adressierung:
 $\langle \text{Mnemo} \rangle \text{ Ri} \quad \text{EA} = \text{i}$
- Register-indirekte Adressierung:
 $\langle \text{Mnemo} \rangle (\text{Ri}) \quad \text{EA} = (\text{Ri})$
- Register-relative Adressierung:
 $\langle \text{Mnemo} \rangle \langle \text{Offset} \rangle (\text{Ri}) \quad \text{EA} = (\text{Ri}) + \langle \text{Offset} \rangle$
- Register-relative Adressierung mit Index, ohne Offset:
 $\langle \text{Mnemo} \rangle (\text{Ri}) (\text{Rj}) \quad \text{EA} = (\text{Ri}) + (\text{Rj})$
- Programmzähler-relative Adressierung:
 $\langle \text{Mnemo} \rangle \langle \text{Offset} \rangle (\text{PC}) \quad \text{EA} = (\text{PC}) + \langle \text{Offset} \rangle$



Adressierungsarten bei RISC-Prozessoren

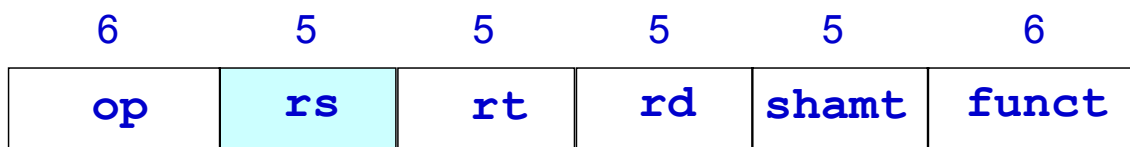
Vier Adressierungsarten:

- **Explizite Register-Adressierung:** Der Operand steht in einem Register
- **Direkte Adressierung:** Der Operand ist eine Konstante im Befehlsword
- **Basis-Adressierung:** Der Operand ist im Speicher an der Adresse, die sich durch die Addition eines Registerinhalts und einer Konstanten im Befehl ergibt.
- **Programmzähler-relative Adressierung:** Die Adresse ergibt sich aus dem Wert des Programmzählers und einer Konstanten im Befehl.

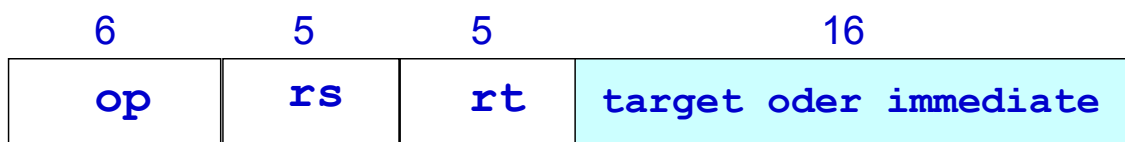


Adressierungsarten bei RISC-Prozessoren

Registeradressierung: (register)

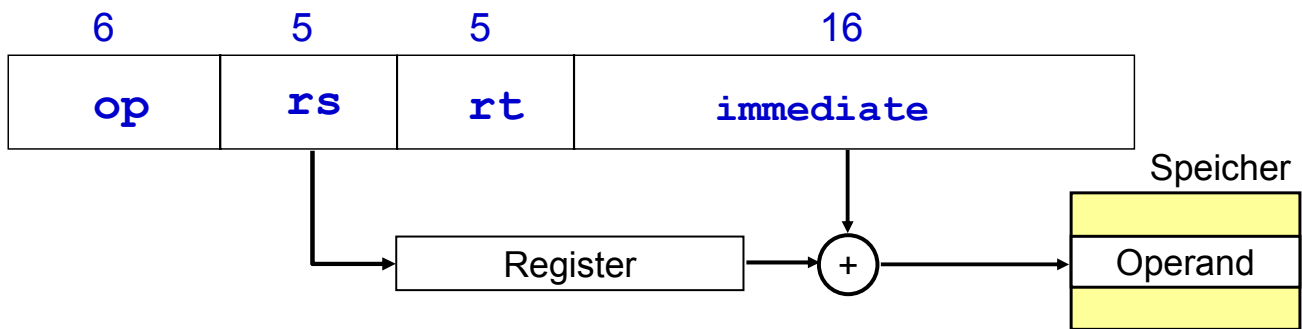


Direkte Adressierung: imm

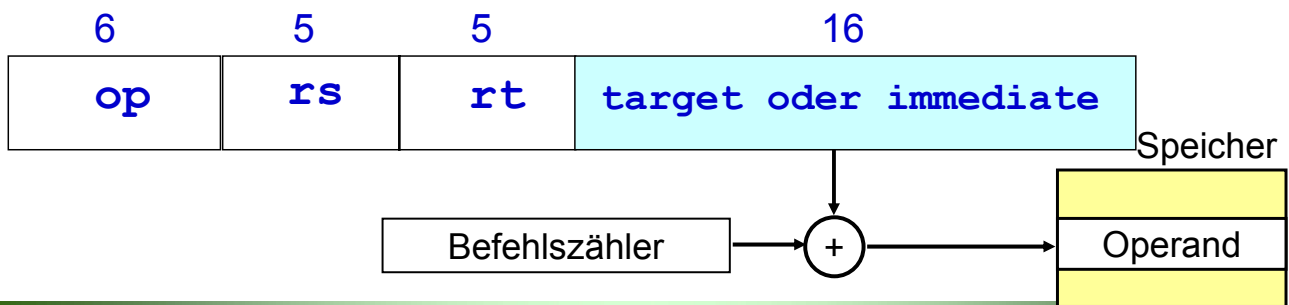


Adressierungsarten bei RISC-Prozessoren

Basisadressierung: imm(register)

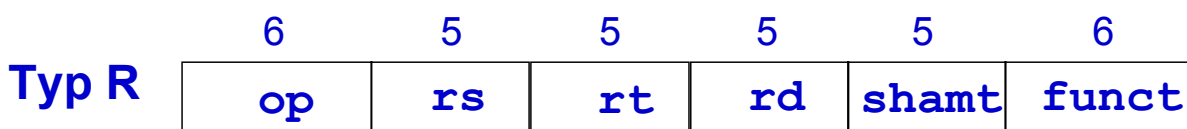
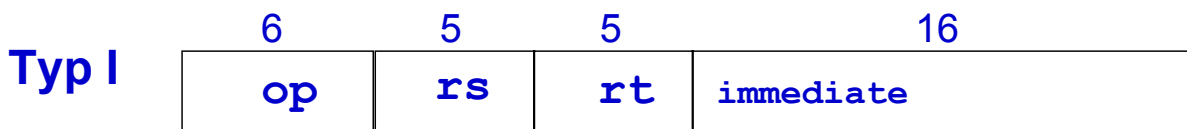


Befehlszähler-relative Adressierung: imm(PC)



Befehlsformate

Der DLX-Prozessor hat ausschließlich Befehle feste Länge (32-Bit, davon 6 Bit als OpCode). Die Befehle werden in Typ I, J und R unterteilt:



Befehlsverarbeitung in RISC-Prozessoren

Einfacher Befehlssatz von RISC-Prozessoren

➔ Maschinenprogramme sind länger als bei CISC Prozessoren

(komplexe Befehle und Adressierungsarten müssen aus den einfachen RISC-Befehlen zusammengesetzt werden)

Trotzdem arbeitet ein RISC-Prozessor meist schneller als ein CISC-Prozessor. Der Grund liegt in der nahezu **vollständigen Parallelarbeit aller Komponenten** eines RISC-Prozessors (extreme Pipeline-Verarbeitung)

Es wird mit großer Wahrscheinlichkeit **in jedem Taktzyklus ein Befehl** beendet



RISC - superskalar

- RISC-Prozessoren, die das Entwurfsziel von durchschnittlich einer Befehlsausführung pro Takt (CPI – *cycles per instruction* oder IPC – *instructions per cycle* von eins) erreichen, werden als **skalare RISC-Prozessoren** bezeichnet.
- Die Superskalar-Technik ermöglicht es, pro Takt mehrere Befehle den Ausführungseinheiten zuzuordnen und eine gleiche Anzahl von Befehlsausführungen pro Takt zu beenden.
- Solche Prozessoren werden als **superskalare (RISC)-Prozessoren** bezeichnet, da die oben definierten RISC-Charakteristika auch heute noch weitgehend beibehalten werden.
- Heutige Mikroprozessoren nutzen Befehlsebenenparallelität durch die Pipelining- und Superskalartechnik.

