



HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Technische Grundlagen der Informatik

Rechner / Mikroprozessoren

Prof. Dr.-Ing. Benjamin Weinert



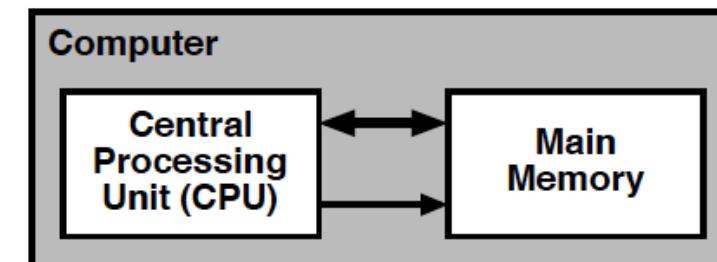
Gliederung

- Rechner/Rechnersysteme
- Rechnerarchitektur
- Von-Neumann-Rechner
- Mikroprozessor
- Befehlsformate und Befehlstypen
- Pipelining

Was ist ein Computer?

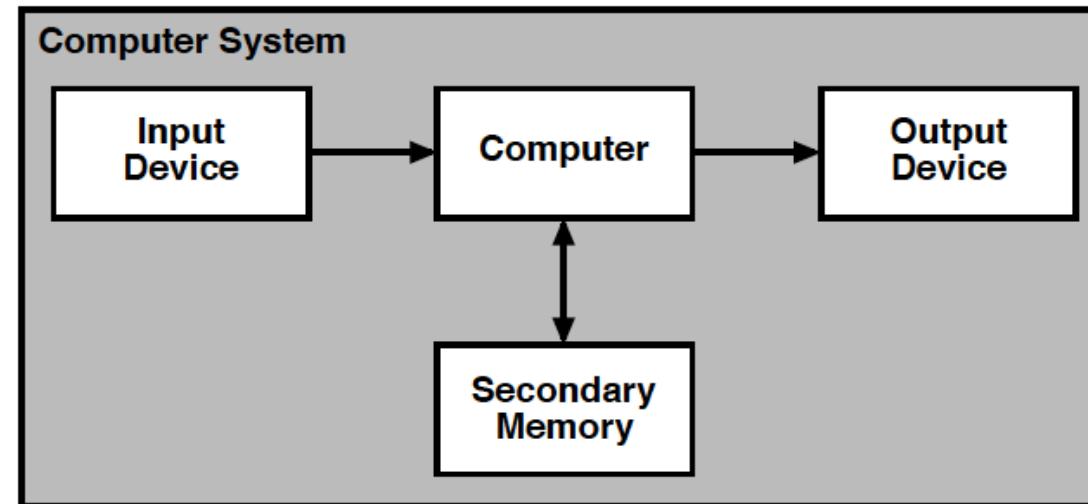
■ Definition

Computer („Rechner“, „Datenverarbeitungsanlage“) - elektronisch arbeitende Einrichtung, die Probleme dadurch löst, dass sie Daten nach einem vorgegebenen Algorithmus beziehungsweise Programm verarbeitet.



Was ist ein Computersystem?

- Computer und zugehörige Peripheriegeräte
 - Eingabegeräte
 - Ausgabegeräte
 - Sekundärspeicher



Was ist Rechnerarchitektur?

■ externe Architektur

- Eigenschaften die für Softwareentwickler sichtbar sind
 - Befehlssatz
 - Adressierungsarten
 - Datenbreite
 - Ein-/Ausgabemechanismen

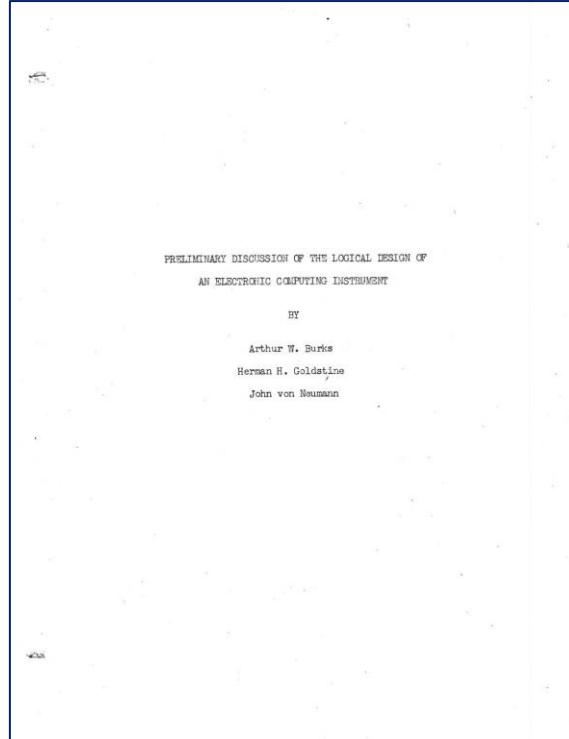
→ Rechnerarchitektur, Rechnerstrukturen

■ interne Architektur

- Implementierung der Eigenschaften
 - Kontrollsignale
 - Schnittstellen
 - Speicherorganisation

→ Rechnerorganisation

Das Von-Neumann-Konzept



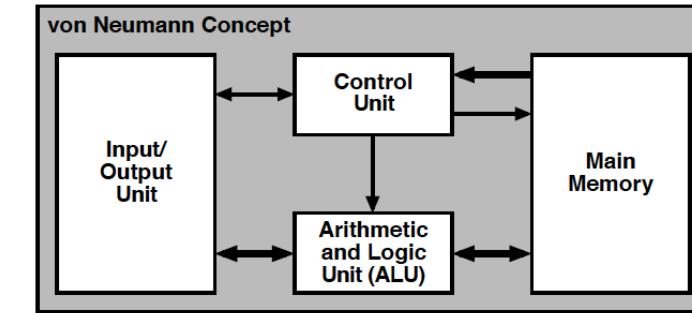
Arthur W. Burks, Hermann H. Goldstine, John von Neumann:
**“Preliminary Discussion of the Logical Design
of an Electronic Computing Instrument”**

Report to the U.S. Army Ordnance Department (1946)

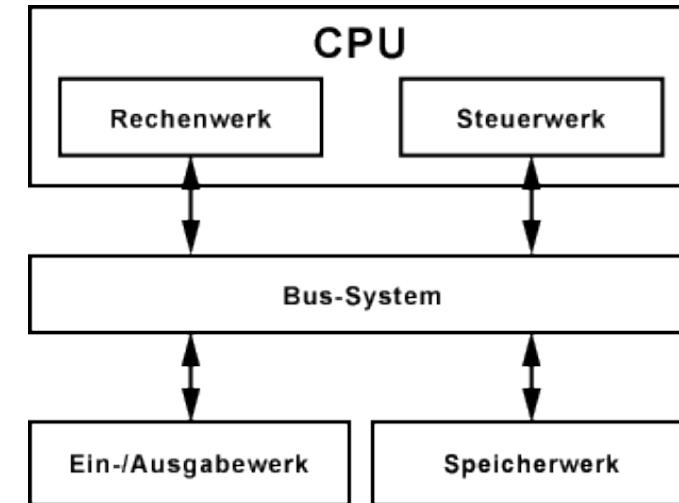
Das Von-Neumann-Konzept

Komponenten

- Stellt das Grundprinzip eines Computersystems dar
 - Prägt bis heute den Aufbau moderner Mikroprozessoren
 - Besteht aus vier wesentlichen Komponenten



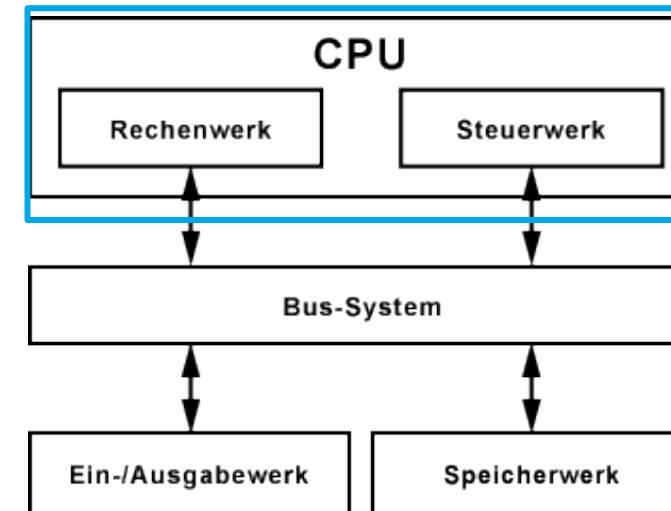
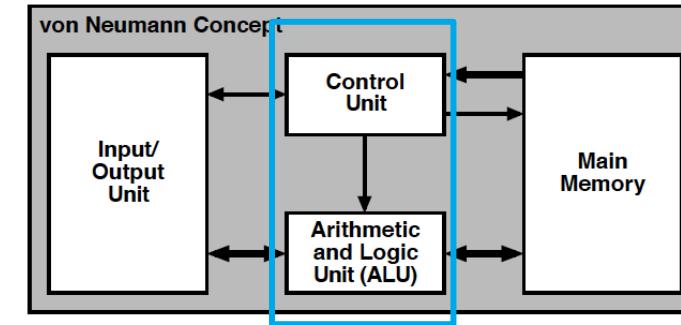
- Die „CPU“ ist das zentrale Steuerelement
 - Übernimmt die Koordination des Systems
 - Übernimmt die Befehlsausführung
 - Heute verfügen Computersysteme über mehrere, parallel arbeitende Prozessoren (Multiprozessorsystem)
 - Die Komponenten sind über ein Bussystem miteinander verbunden



Das Von-Neumann-Konzept

Komponenten

- Die CPU besteht aus Steuerwerk und Rechenwerk
- Steuerwerk
 - lädt Anweisungen aus dem Speicherwerk
 - decodiert diese
 - erzeugt eine Sequenz von Steuersignalen für das Rechenwerk und andere Einheiten
- Rechenwerk
 - ALU oder Akkumulatorregister
 - lädt Daten aus dem Hauptspeicher
 - kombiniert diese auf Basis der Steuersignale (Anweisungen)
 - schreibt Ergebnis in das Speicherwerk

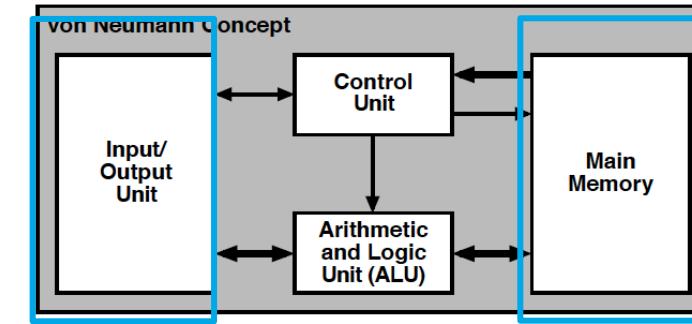


Das Von-Neumann-Konzept

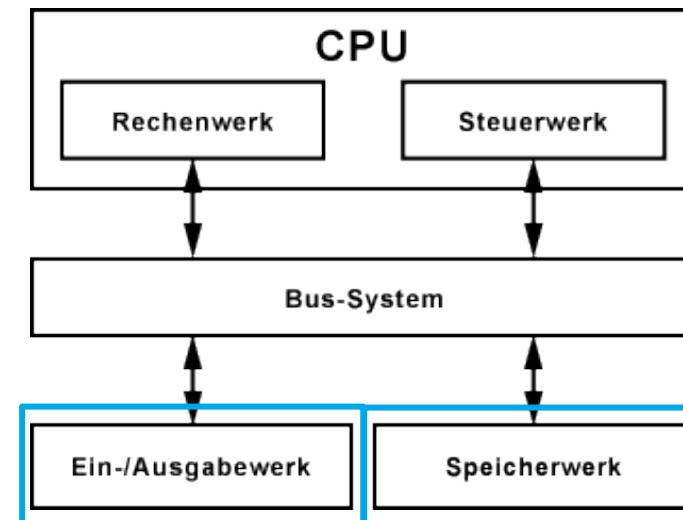


Komponenten

- **Hauptspeicher / Speicherwerk**
 - speichert Daten und Anweisungen
 - in Binärform
 - organisiert mit wahlfreiem Zugriff
 - Daten und Instruktionen können direkt über ihre Hauptspeicheradresse geladen werden



- **Ein-Ausgabewerk**
 - ermöglicht Kommunikation zwischen den Funktionseinheiten und peripheren Geräten
 - z.B. Tastatur, Monitor, Drucker, Festplatten, Netzwerk

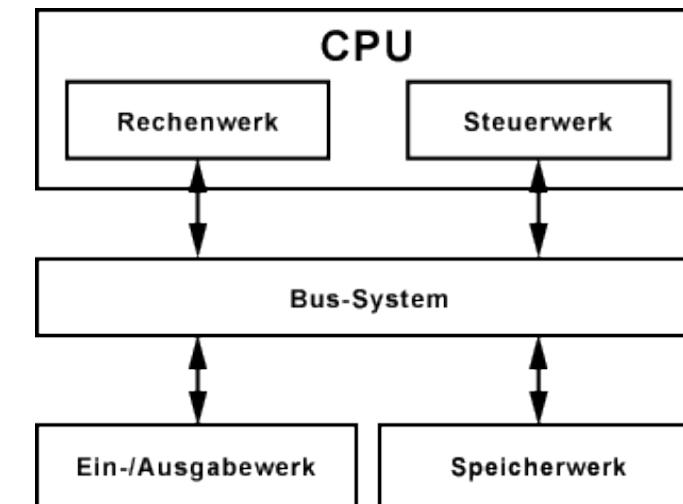
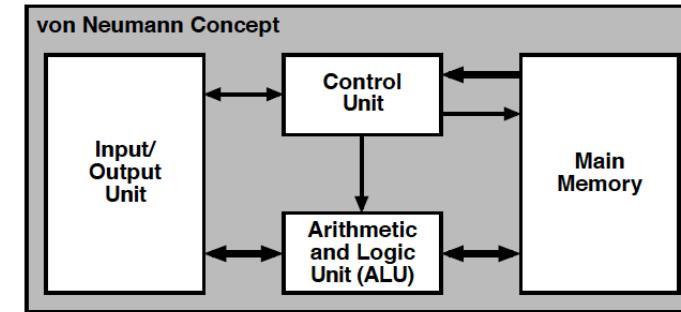


Das Von-Neumann-Konzept

Funktionsweise & Eigenschaften



- Programmgesteuert
 - Im Speicher abgelegte Datenwörter werden von der CPU gemäß einem definierten Befehlssatz interpretiert und ausgeführt
- Universelles Berechnungsmodell
 - Frei programmierbar und nicht für eine spezielle Aufgabe konzipiert
 - Befehlssatz des Prozessors muss jeden möglichen Algorithmus ausdrücken können
 - Datenaustausch mit dem Speicher
 - Sprungfähigkeit
- Daten und Anweisungen werden im Hauptspeicher abgelegt
 - können geändert werden, da Schreib-Lese-Speicher
 - **Keine Trennung zwischen Daten und Programm!**
 - CPU entscheidet, ob das Datenwort einer Speicherstelle als Befehls- oder als Datenwort interpretiert wird
 - Alternativ: Harvard Mark I mit der Harvard-Architektur



Das Von-Neumann-Konzept

Funktionsweise & Eigenschaften



Abstraktionsebene	Daten	Kontrollfluss	Strukturelemente
Nicht-prozedurale Programmiersprachen	abstrakte Datentypen	Datenfluss Nachrichtenübertragung Inferenz	Funktionen Klassen Regeln
Höhere Programmiersprachen	zusammengesetzte Datentypen	Sequenz Verzweigung	Module
Einfache Programmiersprachen	einfache Datentypen	Iteration	Prozeduren Funktionen
Assembler	(interpretierte) Bitstrings	Sprünge	Makros
Maschinensprache	Bitstrings		

- Software wird üblicherweise in höheren Programmiersprachen entwickelt
 - unabhängig von der verwendeten Hardware
- Programme werden durch Compiler in Assembler übersetzt
 - abhängig von der verwendeten Hardware
- Assembler wird in binäre Maschinensprache übersetzt
 - ausführbar durch die verwendete Hardware

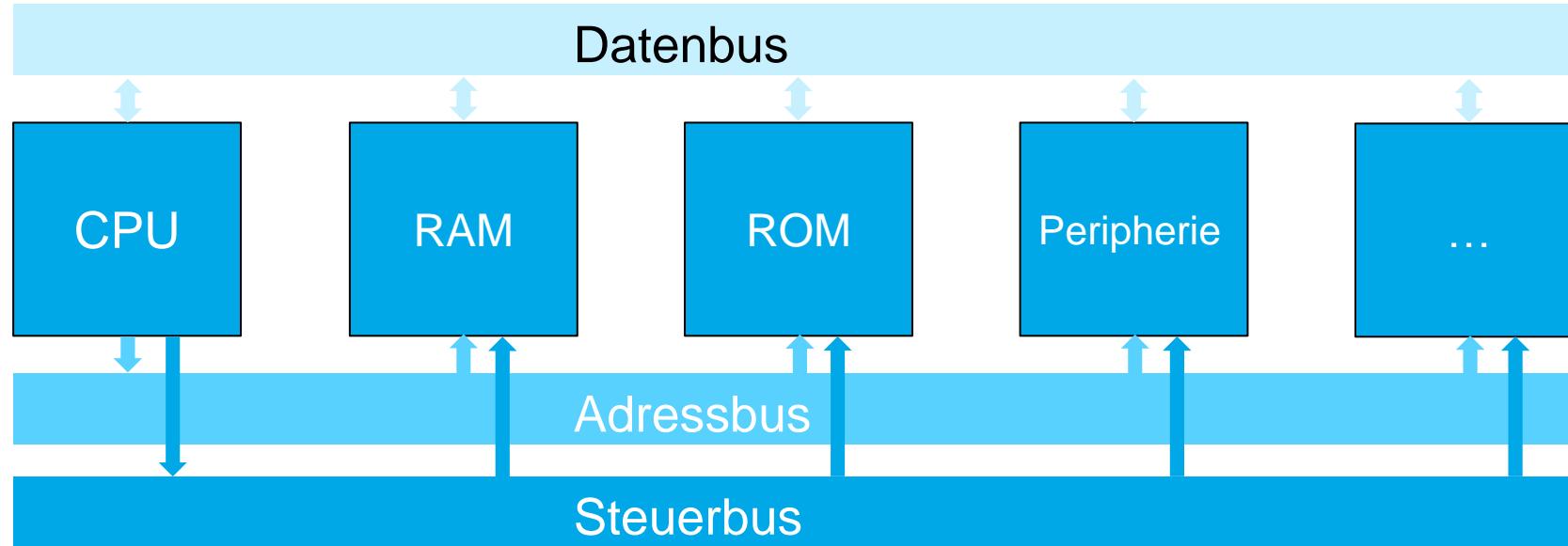
Das Von-Neumann-Konzept

Funktionsweise & Eigenschaften



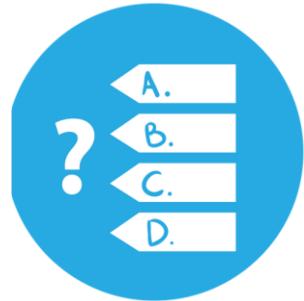
- Beispiel: Übertragung von Anweisungen einer höheren Programmiersprache in Befehle auf Ebene eines Mikroprozessors
 - **if (x != 0) y = y + 1**
- Das Kurzprogramm wird mittels Compiler in maschinennahe Assembler-Sprache übersetzt
 - **start: LDA 14** // Operand x aus Speicherstelle 14 laden
 - **BRZ weiter:** // Operand mit 0 vergleichen; bei !=0 weiterspringen
 - **LDA 15** // Operand y aus Speicherstelle 15 laden
 - **ADD #1** // Addition von 1
 - **STA 15** // Operand y (neu) in Speicherstelle 15 laden
- Für Interessierte: Exkurs zu Assembler: <https://www.csnewbs.com/assemblylanguage>

Verfeinerung von Von-Neumann: Das Systembus-Konzept

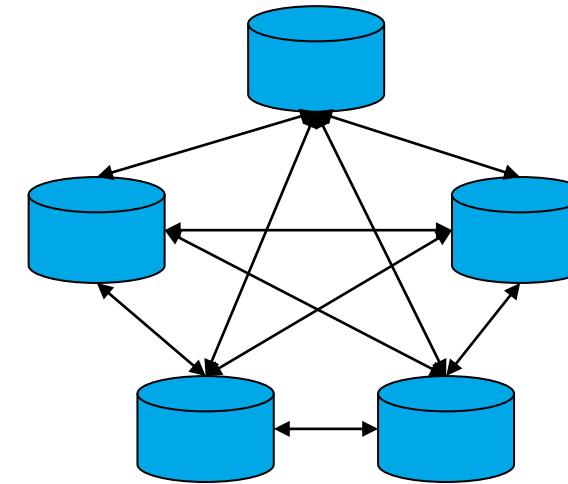
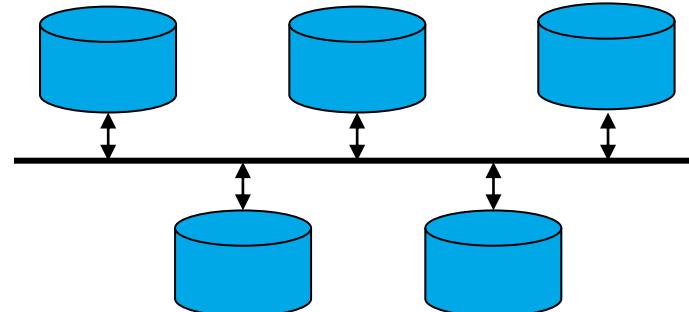


- Adressinformation (unidirektional): z.B. Übermittlung der Speicheradressen an den Speicher
- Dateninformation (bidirektional): Auslesen und Einlesen von Operanden vom Speicher in die CPU bzw. umgekehrt
- Kontrollinformation (unidirektional): Steuerung des Betriebsmodus, z.B. Lesezugriffe oder Schreibzugriffe

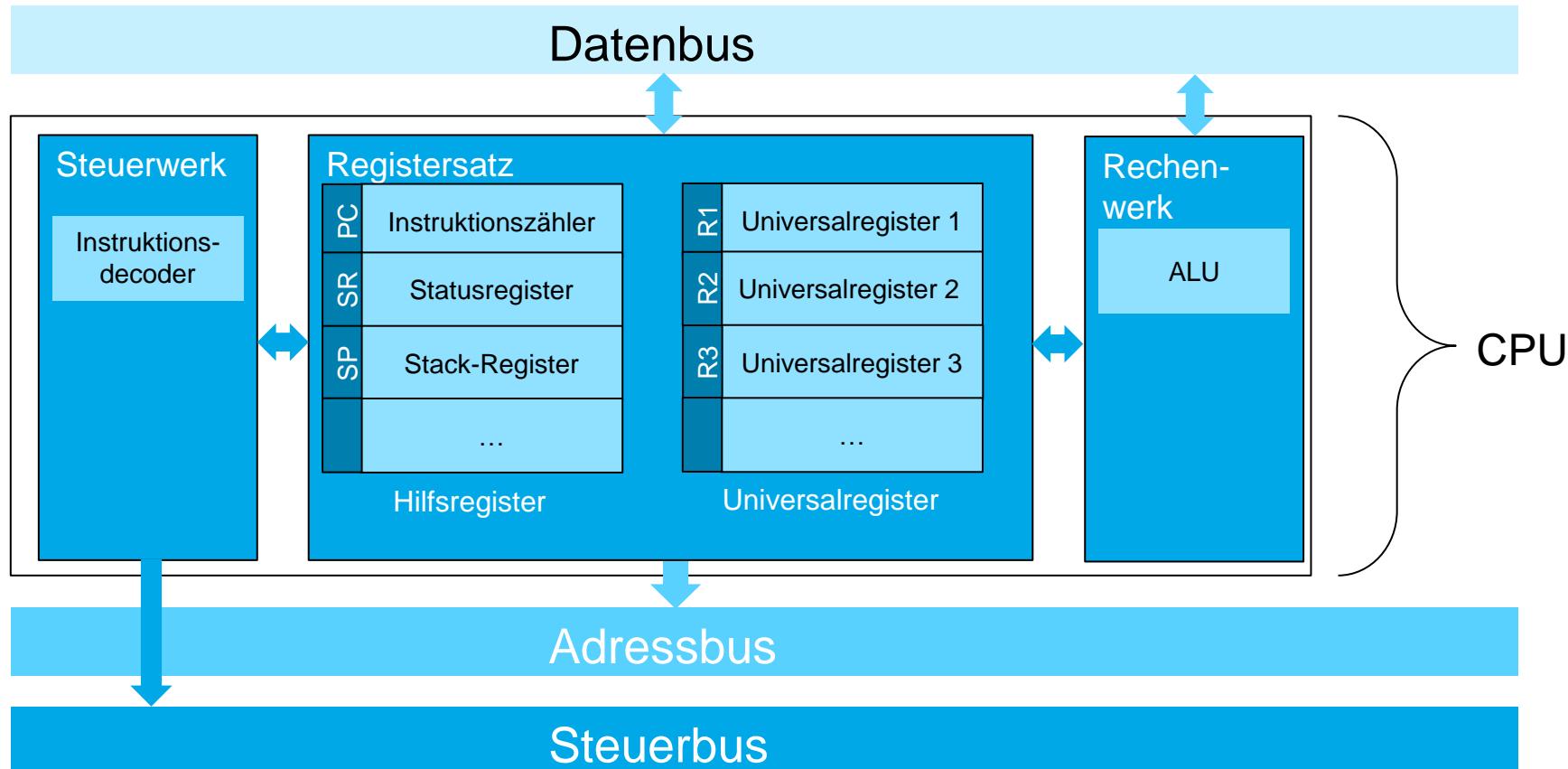
Diskussion



- Direkte Kommunikation vs. Indirekte Kommunikation
 - Vor- und Nachteile
 - Welchen Nachteil hat „von-Neumann“ zusätzlich?



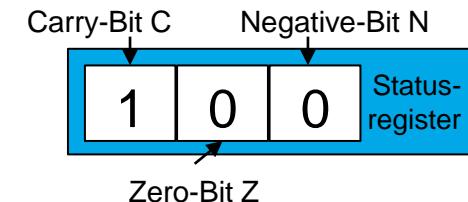
Generische Prozessorarchitektur



- Steuerwerk: Überwachung d. Kontroll- und Datenfluss und Datentransport zwischen Registern, Rechenwerk, Außenwelt
- Registersatz: Zwischenspeicher von Datenworten im Universalregister, Hilfsregister mit speziellen Aufgaben
- Rechenwerk: Eigentliche Datenverarbeitung via Akkumulator oder ALU

Generische Prozessorarchitektur - Registersatz

- Die Register sind kleine, jedoch sehr schnelle Speicherelemente, die im Prozessor-Kern untergebracht sind. Die Wortbreite der Register entspricht typischerweise der Breite der Datenworte, die vom Prozessor verarbeitet werden können.
- Universalregister
 - Zur Zwischenspeicherung von Datenworten
 - Können mit beliebigen Werten befüllt werden
 - Theoretisch ein Universalregister ausreichend
 - Dann müssten jedoch Zwischenergebnisse in den Speicher geschrieben werden
 - Mikroprozessoren verwenden eine Anzahl von Registern (ca. 20 bis 200) um Operanden und Ergebnisse von Operationen zu speichern
- Hilfsregister
 - Instruktionszähler / Befehlszähler (program counter, PC)
 - beinhaltet die Speicheradresse des nächsten auszuführenden Befehls.
 - Bei Laden des Befehls wird die Speicheradresse um +1 erhöht.
 - Sprünge durch Überschreiben der Speicheradresse im PC-Register
 - Statusregister (status register, SR);
 - vom Rechenwerk beschrieben,
 - beinhaltet Statusbits über zuletzt durchgeföhrte Operation.
 - Carry-Bit C – Überlaufprüfung
 - Zero-Bit Z – Prüfung, ob Ergebnis = 0
 - Negative-Flag N – Ergebnis negativ
 - Stapelregister (stack pointer, SP)
 - Für Aufruf von etwaigen Unterprogrammen
 - PC wird nicht überschrieben



Generische Prozessorarchitektur - Befehlsverarbeitung



■ Befehlsholphase (Fetch)

- Befehl wird via Befehlszähler (Instruktionszähler) aus dem Hauptspeicher geholt
- Einlesen des Befehls in die CPU
- Befehlszähler +1

■ Decodierphase (Decode)

- Dekodierung des Befehls
- Laden der für den Befehl erforderlichen Operanden

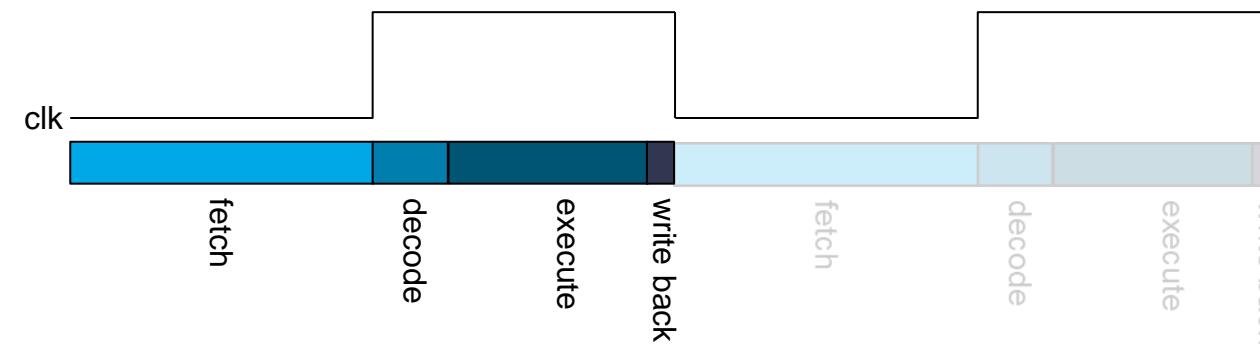
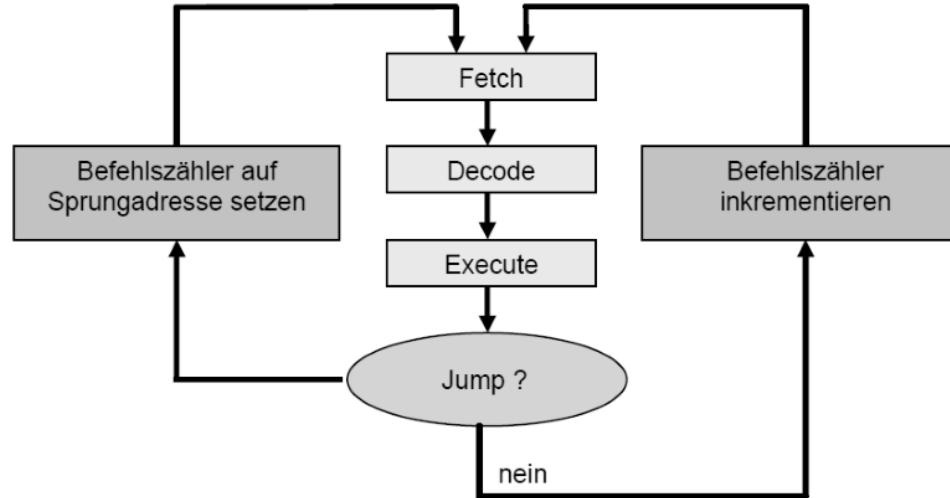
■ Ausführungsphase (Execute)

- Verknüpfung der Operanden im Rechenwerk

■ Speicherphase (Write back)

- Schreiben des Ergebnis in den Speicher
- Ablage in einem internen Register
- Verwendung als Sprungadresse

■ Mit jedem Takt wird ein Maschinenbefehl ausgeführt



Entwicklung von Prozessorarchitekturen

- 1940er: Akkumulator-Architekturen
 - auf Grund beschränkter Hardware-Ressourcen
- 1950er: Load/Store-Architekturen (alt. Register-Register-Architektur)
 - Leistungssteigerung durch schnelle Allzweckregister
- 1960er: Stack-Architekturen
 - angepasst an Programmiersprachen und Compiler
- 1970er: Storage/Storage-Architekturen
 - einfache Übersetzung durch komplexe Maschineninstruktionen
- 1980er: Reduced Instruction Set Computer (RISC)
 - einfachere Maschineninstruktionen wegen verbesserter Compilertechnik

Prozessorarchitektur / Befehlssatzarchitektur

- Eine Prozessor- oder Befehlssatzarchitektur umfasst die gesamte nach außen sichtbare Architektur eines Prozessors
- Man spricht von einer Prozessorfamilie, wenn alle Prozessoren die gleiche Basisarchitektur haben
 - Abwärtskompatibilität
- Beschreibt die Eigenschaften der nach außen sichtbaren Prozessorarchitektur und umfasst:
 - Befehlssatz
 - Registermodell
 - Datenformate
 - Adressierungsarten
 - Ein-/Ausgabeorganisation
- Beantwortet:
 - Wie werden Daten repräsentiert?
 - Wo werden die Daten gespeichert?
 - Welche Operationen können auf den Daten ausgeführt werden
 - Wie werden die Befehle codiert?
 - Wie wird auf die Daten zugegriffen?

Befehlssatzarchitektur - Befehlssatz & Befehlstypen

- Ein Befehlssatz ist die Gesamtheit aller Maschinenbefehle, die ein Mikroprozessor verarbeiten kann.
- **Maschinenbefehle** werden in verschiedene **Befehlstypen** gruppiert:
 - **Arithmetisch/Logische Befehle**
 - Fließkommaoperationen
 - Ganzzahlige Operationen
 - **Datentransport-Befehle**
 - **Steuerbefehle**
 - Jump
 - Branch
 - Procedure Call and Return
 - Interrupts, Traps
 - **Ein-/Ausgabebefehle**
 - **Spezialbefehle (Systemsteuerung)**

Befehlssatzarchitektur - Arithmetisch/Logische Befehle



■ Arithmetische Befehle

- Berechnungen mit vorzeichenbehafteten, ganzen Zahlen
- add, subtract, multiply, divide
- increment, decrement, negate

■ Logische Befehle

- bitweise logische Operationen
- and, or, not
- Shift-Operationen

■ Typkonvertierung

- z.B. binär nach dezimal

■ Fließkommaarithmetik

- durch spezielle Hardware unterstützt
- früher häufig separate Prozessoren

Beispielhafter arithmetischer Befehl

- Seien a, b, c und d Integerzahlen
- Die vier Zahlen stehen wie folgt im Hauptspeicher
 - a = MEM [100 .. 103] b = MEM [104 .. 107]
 - c = MEM [108 ... 111] d = MEM [112 ... 115]
- Berechne: d := a + b + c

LW R1, 100 (R0)

LW R2, 104 (R0)

LW R3, 108 (R0)

ADD R4, R1, R2

ADD R5, R4, R3

SW 112 (R0), R5

Befehlssatzarchitektur - Datentransport-Befehle

- Datentransport
 - vom Hauptspeicher in Register
 - von Register in Hauptspeicher
 - innerhalb des Hauptspeichers
- Angabe von
 - Quelle
 - Ziel
 - Datenmenge
- Unterschiedliche Implementierungen
 - verschiedene Befehle für unterschiedliche Datentransporte
 - z.B. IBM /370
 - ein Befehl mit unterschiedlichen Adressierungsarten
 - z.B. DEC VAX

Befehlssatzarchitektur

- Steuerbefehle



- Jump - unbedingter Sprung
 - Zieladresse wird in Programmzähler geladen
- Branch – bedingter Sprung
 - wenn Bedingung erfüllt, wird Zieladresse in den Programmzähler geladen
- Procedure Call and Return
 - springt vom Hauptprogramm zum Unterprogramm und zurück
 - speichert aktuellen Prozessorzustand, mindestens Rücksprungadresse
- Interrupts/Traps (synchrone oder asynchrone Unterbrechung)
 - Programmausführung wird gestoppt
 - Prozessorzustand wird gesichert
 - Interrupt-Handler wird aufgerufen (üblicherweise Teil des Betriebssystems)

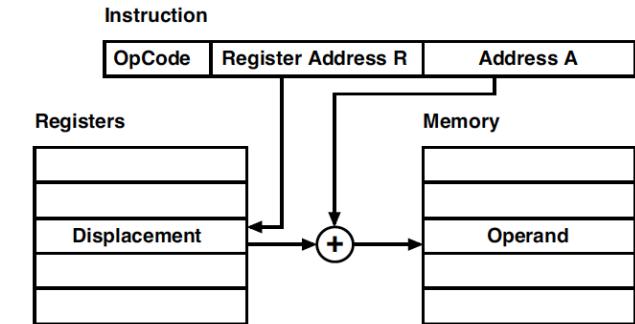
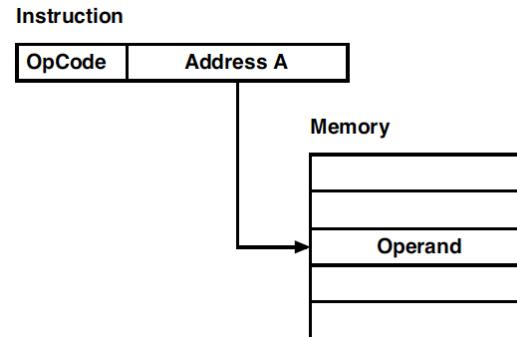
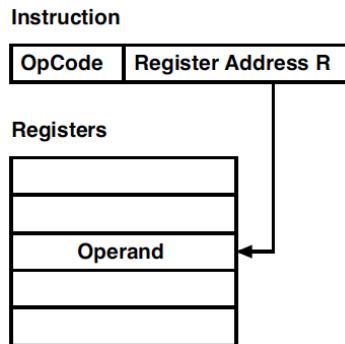
Befehlssatzarchitektur

- Zieladressen



- explizit
 - im Befehl enthalten
 - zur Übersetzungszeit bekannt
 - benutzt einen der zulässigen Adressierungsmodi
 - durch Angabe eines Displacement
 - wird zum Programmzähler addiert

- implizit
 - Adresse ist in einem Register gespeichert
 - Angabe einer Registeradresse in der Operation
 - Verwendung eines vordefinierten Registers (z.B. beim Unterprogrammrücksprung)



Befehlssatzarchitektur - Weitere Befehlstypen

- Ein-/Ausgabebefehle
 - Kommunikation erfolgt über den Hauptspeicher
 - durch Datentransferbefehle in spezielle Speicherbereiche
 - Memory-mapped
 - Zugriff von außen durch speziellen Controller
 - Direct Memory Access (DMA)
- Spezialbefehle (Systemsteuerung)
 - Operationen mit speziellen Rechten
 - Prozessor wird in speziellen Zustand versetzt
 - Kernel Mode
 - dürfen nur vom Betriebssystem genutzt werden
- Interesse geweckt? => Wahlpflichtmodul „Rechnernetze & Betriebssysteme“ im 4. Semester.

Befehlssatzarchitektur - Klassifikation von Befehlssätzen



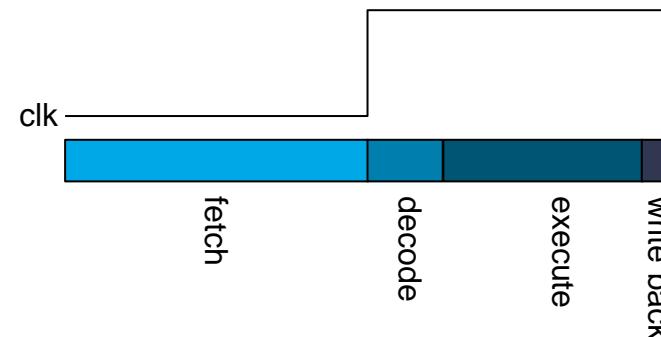
- Befehlssätze lassen sich in verschiedene Art und Weise klassifizieren
- Klassifikation nach Komplexität
 - **CISC** = Complex Instruction Set Architecture
 - **RISC** = Reduced Instruction Set Architecture
- Klassifikation nach Verortung der Operanden
 - Register-Speicher-Architektur
 - Bei einer Register-Speicher-Architektur können die Operanden der Befehle sowohl in Registern als auch im Speicher stehen
 - Load-Store-Architektur
 - Quell- und Zieloperand stehen im Register
 - Speicherzugriff via Lade- und Speicher-Befehle
- Klassifikation nach Anzahl der Operanden
 - Drei-Adress-Maschine: bis zu zwei Quell- und ein Zieloperand
 - Zwei-Adress-Maschine: zwei Quelloperanden, ein Quelloperand ist gleichzeitig Zieloperand
 - Ein-Adress-Maschine/Akkumulator-Maschine
 - Null-Adress-Maschine / Stack-Maschine

Befehlssatzarchitektur - Befehlsformate

- Wesentliche Eigenschaft einer CPU ist die Anzahl von Operanden, die ein einzelner Befehl maximal entgegennimmt.
- Das Befehlsformat definiert, wie die Befehle codiert sind
- Befehscodierung beginnt mit dem OpCode, der den Befehl selbst festlegt
 - Je nach OpCode werden weitere Felder im Befehlsformat benötigt
 - z.B. für arithmetisch-logische Befehle die Adressfelder für benötigte Operanden (OPx)
 - oder für Lade-/Speicherbefehle die Quell- und Zieladressangaben
- Je nach Art, wie die arithmetisch-logischen Befehle ihre Operanden adressieren, unterscheidet man vier Klassen von Befehlssätzen:
 - 3-Adress-Format OpCode, OP1, OP2, DEST
 - explizite Form binärer Operationen
 - 2-Adress-Format OpCode, DEST/OP1, OP2
 - Resultat überschreibt einen der beiden Operanden
 - 1-Adress-Format OpCode, OP1
 - zweiter Operand steht im Akkumulator(-register)
 - Ergebnis wird ebenfalls im Akkumulator gespeichert
 - 0-Adress-Format OpCode
 - beide Operanden werden vom Stack geladen
 - Ergebnis wird auf dem Stack abgelegt

Befehlszyklus

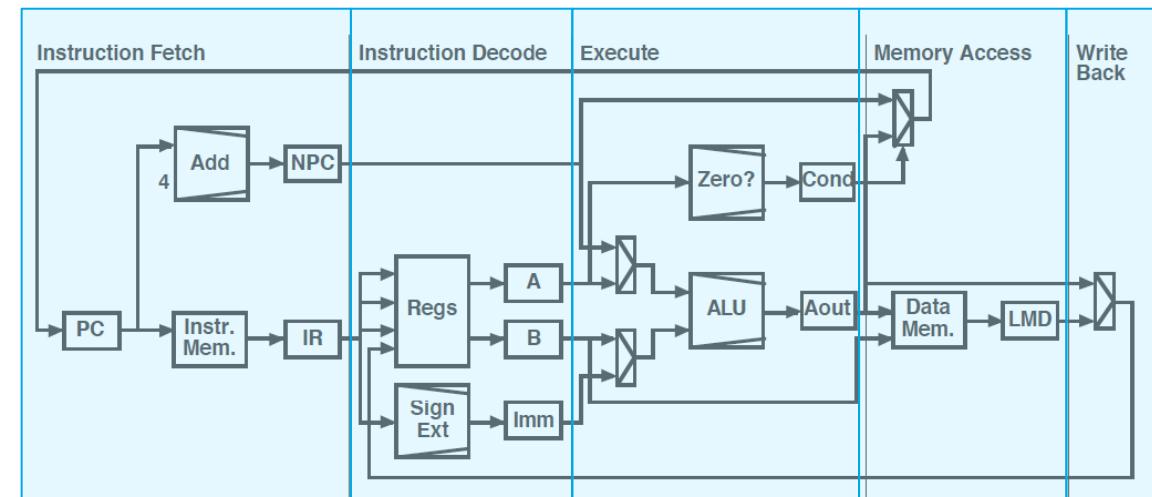
- Der Befehlszyklus eines von-Neumann-Rechners besteht prinzipiell aus zwei Phasen
 - Instruction Fetch
 - Hauptspeicher wird mit dem Inhalt des Programmzählers oder einer expliziten Sprungadresse adressiert
 - Die gelesene Information wird ans Steuerwerk übergeben und als Befehl interpretiert
 - Instruction Execution
 - Hauptspeicher wird mit den Operandenadressen, die im Befehl enthalten sind, adressiert
 - Die gelesene Information wird der arithmetisch-logischen Einheit zugeführt
 - Das Resultat der Operation wird in den Hauptspeicher zurückgeschrieben
 - an die Adresse, die implizit oder explizit im Befehl enthalten ist



Detaillierter Befehlszyklus einer DLX-Prozessorarchitektur

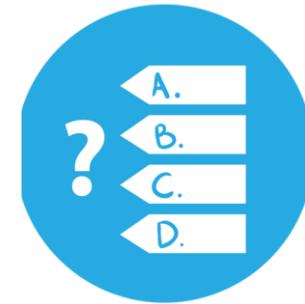
- Für Rechner mit einer Load/Store-Architektur lässt sich der Befehlszyklus weiter unterteilen
 - Instruction Fetch IF
 - liest Befehl aus dem Hauptspeicher, der durch Befehlszähler oder Sprungadresse adressiert wird
 - Instruction Decode and Source Register Fetch ID
 - Dekodieren des Befehls, auslesen des Quellregisters
 - Instruction Execution oder Effective Adress Calculation EX
 - Adressberechnung für Load/Store-Befehle
 - Zieladressberechnung für Sprünge
 - Ausführung von ALU-Befehlen
 - Memory Access oder Branch/Jump MEM
 - Ausführung von Load/Store-Befehlen
 - Ausführung von Sprüngen
 - Write Back to Destination Registers

Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.



Pipelining

- Wir haben:
 - Eine Waschmaschine (30 Minuten)
 - Ein Trockner (40 Minuten)
 - Eine Person, die falten (20 Minuten)

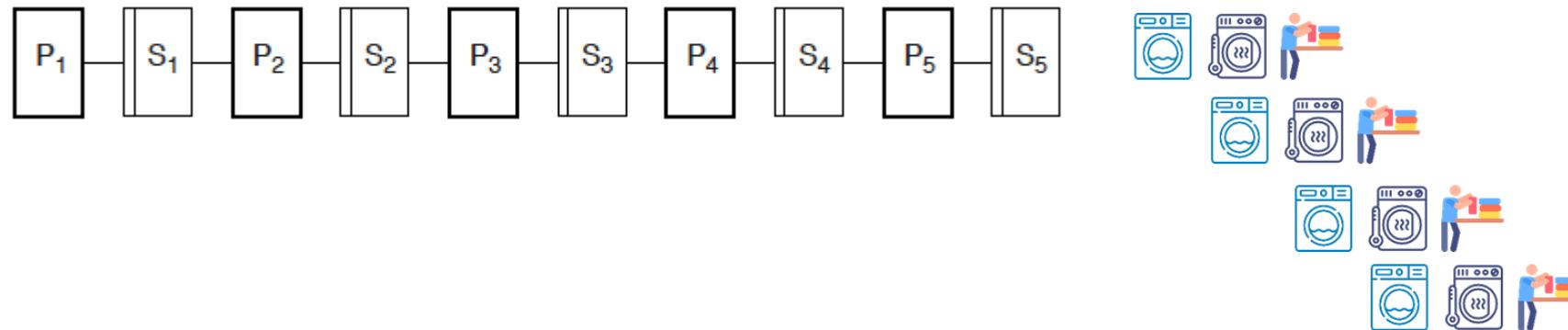


- Es benötigt 90 Minuten für eine Ladung Wäsche waschen, trocknen und falten
 - Wie lang benötigen Sie für 4 Ladungen Wäsche?



Pipelining

- Eine Verkettung von Verarbeitungseinheiten, die wie ein Fließband arbeiten, wird Pipeline genannt



- Befehle „betreten“ die Pipeline, werden durch die einzelnen Stufen abgearbeitet und verlassen die Pipeline am anderen Ende
- Jede Stufe bearbeitet einen Teil des Befehls parallel zu den anderen Stufen
- Zwischen den Pipeline-Stufen P_i werden Pufferspeicher S_i benötigt
- Pipelining ist eine Technik, um mehrere Befehle zeitlich überlappend zu bearbeiten
- Die Bearbeitungszeit für einen einzelnen Befehl wird dabei nicht verkürzt

Pipelining

- Der Durchsatz einer Befehlspipeline wird dadurch bestimmt, wie häufig ein Befehl die Pipeline betritt
- Die Pipeline-Stufen sind miteinander gekoppelt
 - Alle Stufen müssen die gleiche Bearbeitungszeit aufweisen
- Die Zeit, um einen Befehl zur nächsten Stufe weiterzureichen wird Maschinenzyklus genannt
- Die Länge des Maschinenzyklus wird durch die langsamste Stufe bestimmt
- Ziel des Pipeline-Entwurfs ist daher, die Durchlaufzeit der Pipeline-Stufen möglichst gleich zu halten
- Bei optimaler Auslegung der Pipeline-Stufen ergibt sich die durchschnittliche Zeit pro Befehlsausführung als

$$\frac{\text{Befehlsausführungszeit ohne Pipelining}}{\text{Anzahl der Pipeline – Stufen}}$$

Grundlegende DLX-Pipeline

- Der DLX-Datenpfad kann ohne größere Veränderungen als Pipeline realisiert werden
 - pro Taktzyklus wird die Bearbeitung eines Befehls gestartet
 - jeder Taktzyklus des DLX-Datenpfads wird zur Pipeline-Stufe

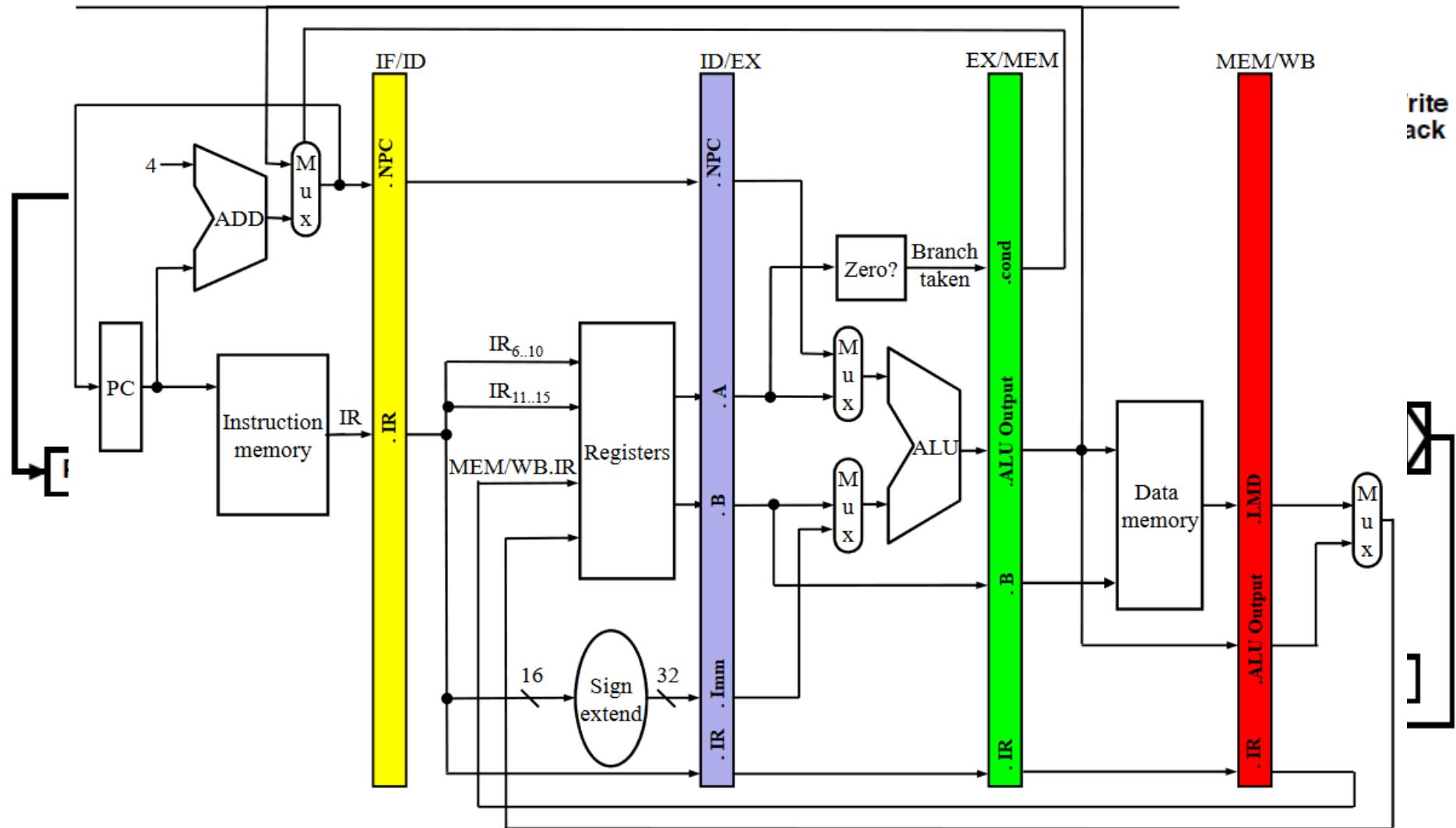
Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

instr / clock cycle	1	2	3	4	5	6	7	8	9
instr i	IF	ID	EX	MEM	WB				
instr i + 1		IF	ID	EX	MEM	WB			
instr i + 2			IF	ID	EX	MEM	WB		
instr i + 3				IF	ID	EX	MEM	WB	
instr i + 4					IF	ID	EX	MEM	WB

Grundlegende DLX-Pipeline

- Jede Stufe ist in jedem Takt aktiv
 - Jede Kombination von gleichzeitig aktiven Stufen muss möglich sein.
- Pipelining erfordert, dass Werte von einer Stufe zur nächsten weitergereicht werden
 - Die Pipeline-Stufen werden durch sogenannte Pipeline-Register abgegrenzt
 - beinhalten alle Register, die Werte zwischen den Taktzyklen eines Befehls puffern
- Ein Befehl ist pro Taktzyklus in genau einer Pipeline-Stufe aktiv
- Jede Berechnung findet zwischen zwei Pipeline-Registern statt

DLX-Datenpfad mit Pipelining



Pipeline-Hazards

- Instruktionen sind nicht immer unabhängig voneinander
- Situationen, in denen die korrekte Ausführung des nächsten Befehls nicht möglich ist, werden als Hazard bezeichnet
 - Erfordern üblicherweise Wartezyklen (sog. stalls)
- Man unterscheidet drei Arten von Hazards
 - Strukturelle Hazards
 - Die gleiche Hardware-Einheit wird von unterschiedlichen Pipeline-Stufen benötigt
 - Daten-Hazards
 - Ein Befehl benötigt das Ergebnis eines vorangegangenen Befehls, das noch nicht vorliegt
 - Steuerungs-Hazards
 - Ein Befehl verändert den Inhalt des PC (branch/jump), aber der nächste Befehl wurde bereits geladen