Agenda

1. Grundlagen
- 1.1 Geschichtliche Entwicklung
- 1.2 Zahlen & Stellenwertsysteme
  - 1.2.1 Binär/Hexadezimal
  - 1.2.2 Komplementdarstellung
  - 1.2.3 Fließkommadarstellung
- 1.3 Arithmetische Operationen
- 1.4 Zeichensätze

2. Rechnerarchitektur
- 2.1 Von-Neumann-Architektur
- 2.2 Boolesche Algebra
- 2.3 Digitale Schaltungen
  - 2.3.1 Halbaddierer
  - 2.3.2 Volladdierer
  - 2.3.3 Ripple-Carry-Adder

3. Betriebssysteme
- 3.1 Grundlagen
- 3.2 Linux/Unix
- 3.3 VMs
- 3.4 Shell-Programmierung
- 3.5 Reguläre Ausdrücke

- Hinter dem Begriff der Rechnerarchitektur verbirgt sich die Beschreibung der technischen Realisierung einer Datenverarbeitungsanlage.
  Dies involviert insbesondere
  - die Gesamtheit der Bauprinzipien einer Rechenanlage. Dazu gehören:
    - die Festlegung der internen Darstellung der Daten
    - die Festlegung der auf den Daten ablaufenden Operationen, das heißt, die Festlegung des Umfangs der Maschinensprache der CPU.
    - der Aufbau der Maschinenbefehle, insbesondere das Format der Instruktionen
    - die Definition von Schnittstellen zwischen den Funktionseinheiten und zu externen Geräten
    - der ’Bauplan’ nach dem die einzelnen Funktionseinheiten zu einem Rechner zusammengefügt werden.
  - Zu beachten sind dabei die grundlegenden Aufgaben einer Rechenanlage: die Sammlung, Speicherung, Verarbeitung und Darstellung von Daten.

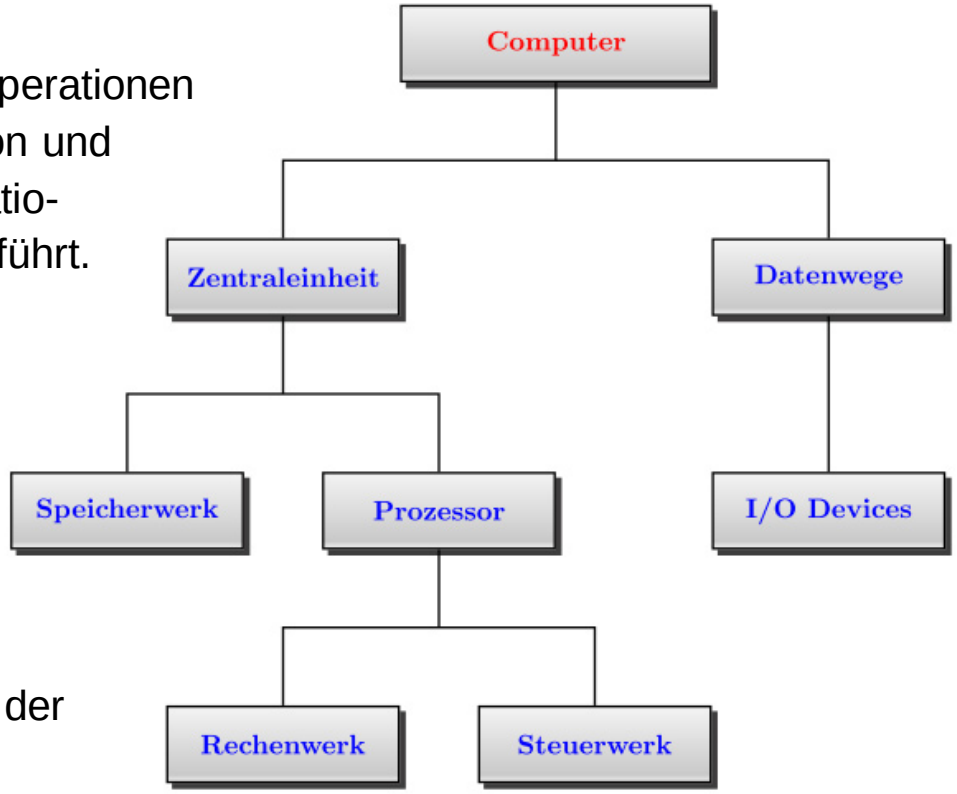
Die von-Neumann-Architektur

- Die meisten der heute in Gebrauch befindlichen Computer (diese realisieren das EVA-Prinzip) sind nach einem Konzept aufgebaut, das der Mathematiker John von Neumann in den vierziger Jahren des letzten Jahrhunderts entwickelt hat.
- Dieses Bauprinzip ist (nach EDV Maßstäben) uralt und heißt von-Neumann-Architektur.

- Ziel von Neumanns war dabei die Entwicklung genereller Konzepte, wie ein Rechner unabhängig von der Art der zu lösenden Aufgabe (universell) aufgebaut sein sollte. Diese generellen Konzepte nehmen daher auch keinerlei Bezug auf irgendeine technische Realisierung.
- Für die von-Neumann-Architektur gibt es keine präzise Definition. Daher ordnet man heute einen Rechner der von-Neumann-Architektur zu, wenn dieser bestimmte Charakteristiken erfüllt.

- Im Laufe der Zeit wurde die ursprüngliche Konzeption erweitert zum sogenannten klassischen Konzept eines Universalrechners.

- Klassisches Konzept eines Universalrechners:
  - Der Rechenautomat wird logisch und physisch in fünf Funktionseinheiten gegliedert:
    - 1. **Rechenwerk**
      Hier werden die arithmetischen Operationen (Addition Subtraktion, Multiplikation und Division) und die logischen Operationen (UND, ODER, NICHT) ausgeführt.
    - 2. **Leitwerk** oder **Steuerwerk**
      Das Steuerwerk veranlasst den Transfer der Instruktionen aus dem Hauptspeicher in den Prozessor, interpretiert (decodiert) die darin codierten Anweisungen und initialisiert über Steuersignale die zur Ausführung der Instruktion notwendigen Komponenten des Rechners.

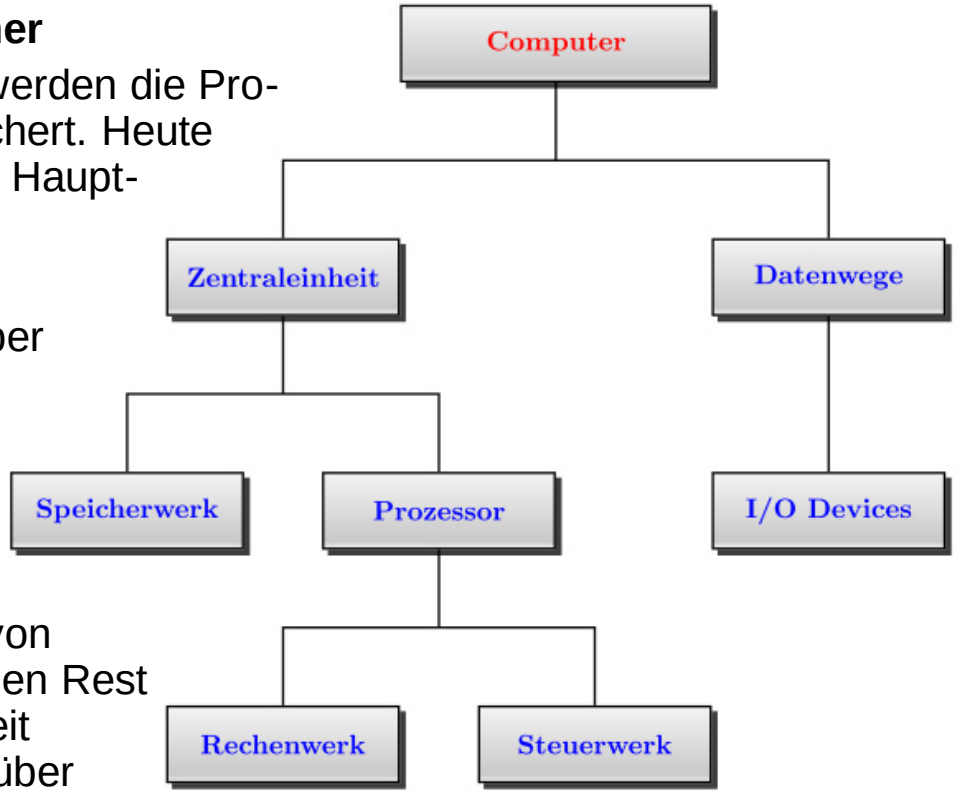


- Klassisches Konzept eines Universalrechners:
  - Der Rechenautomat wird logisch und physisch in fünf Funktionseinheiten gegliedert:
    - 3. **Speicherwerk** oder **Hauptspeicher**

      In diesem Bauteil des Rechners werden die Programme und die Daten abgespeichert. Heute nennt man das Speicherwerk den Hauptspeicher oder Arbeitsspeicher.
    - 4. **Eingabewerk**

      Daten und Programme werden über die Funktionseinheit Eingabewerk in die Rechenanlage eingegeben und laufen über Datenwege in die Speichereinheit.
    - 5. **Ausgabewerk**

      Diese Einheit dient der Ausgabe von Ergebnissen und/oder Daten an den Rest der Welt. Die von der Zentraleinheit berechneten Ergebnisse werden über Datenwege an die Ausgabeeinheiten transferiert.

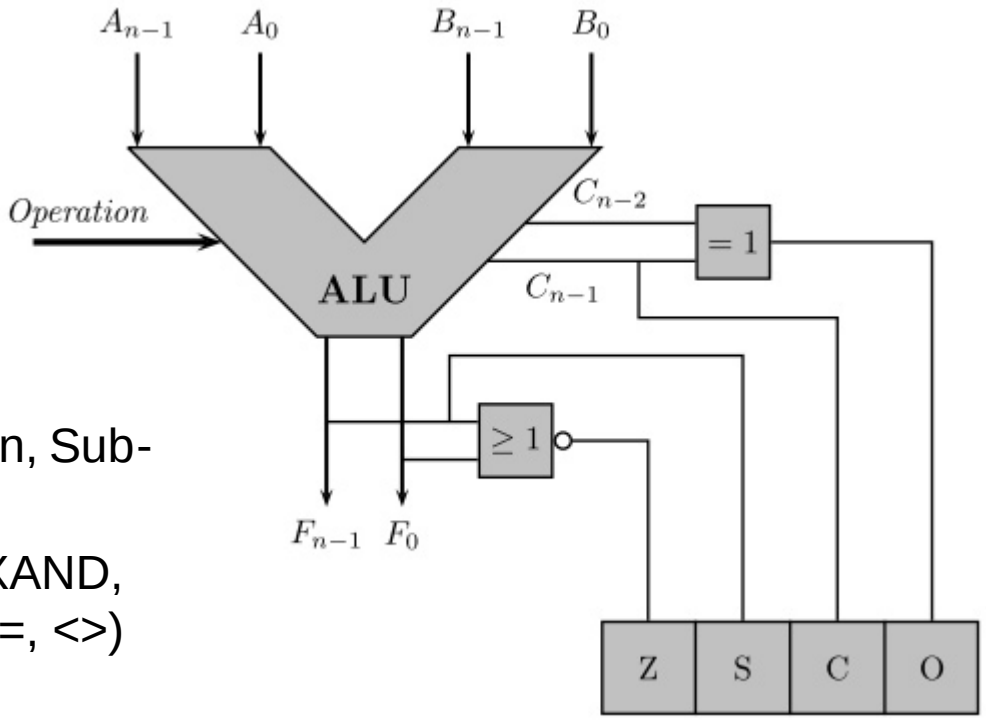


- Eigenschaften eines Universalrechners:
  - ist in seiner Struktur unabhängig von den zu bearbeitenden Problemen. Daher auch der Name Universalrechner.
  - Für jedes Problem eine spezielle Bearbeitungsvorschrift, die über Tastatur, Lochstreifen oder ein anderes Eingabegerät eingegeben und im Speicher abgelegt wird (genannt Programm; Daher sagt man dazu auch speicherprogrammierbare Rechenanlage).
  - Zur Ablage von Programmen und Daten dient derselbe Speicher.
  - Programme und Daten werden binär gespeichert.
  - Aufeinanderfolgende Instruktionen werden im Speicher in aufeinander folgenden Speicherzellen abgelegt. Durch Erhöhung der Befehlsadresse um eine Einheit wird dadurch der nächste Befehl angesprochen (Ausnahme: Sprungbefehle).
  - Der Rechner kann durch diese Architektur selbständig gemäß den Programm-instruktionen logische Entscheidungen treffen. Damit ist der wichtige Schritt vom starren Programmablauf zur flexiblen Programmsteuerung vollzogen, bzw. der Schritt von einer reinen Rechenanlage zur Datenverarbeitungsanlage.

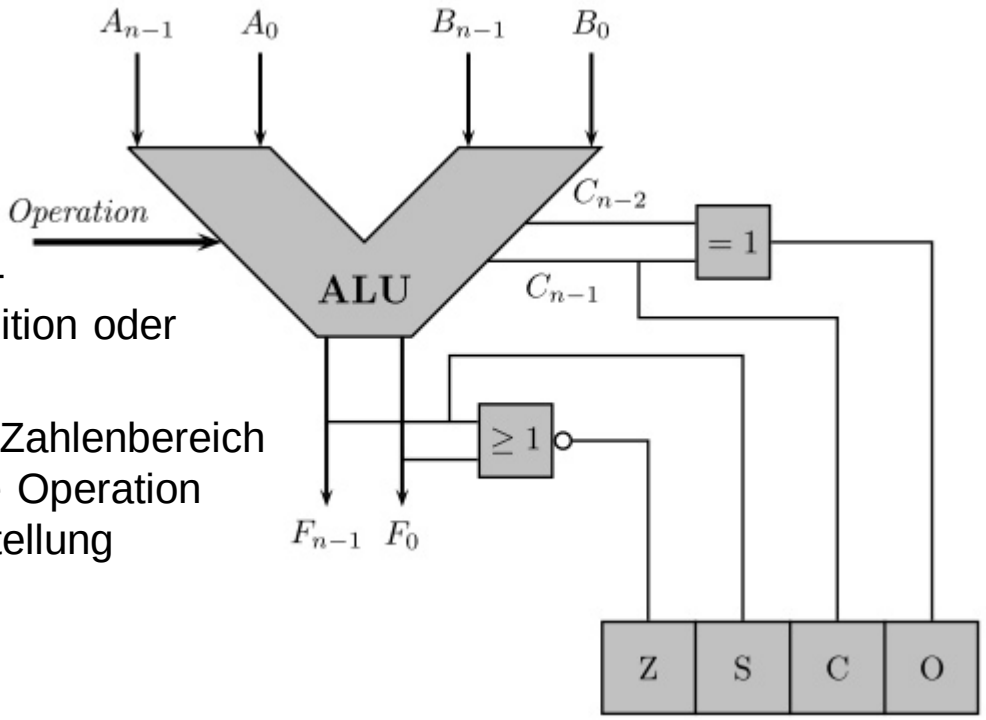
- Nachteile der von-Neumann-Architektur:
  - Im Speicher sind Daten und Instruktionen anhand des gespeicherten Bit-musters nicht unterscheidbar.
  - Konstante und variable Daten können im Speicher nicht unterschieden werden.
  - Bei falscher Adressierung können Speicherinhalte verändert werden, die nicht verändert werden dürfen, wie beispielsweise Instruktionen oder Konstanten. Die Änderung eines einzelnen Bits in einer Maschineninstruktion erzeugt u. U. einen völlig anderen Befehl.

- Der Prozessor:
  - zentrale Komponente des Rechnersystems in dem die eigentliche Verarbeitung der Daten stattfindet.
  - Maschineninstruktionen werden durch das Steuerwerk aus dem Hauptspeicher in den Prozessor geladen.
  - Abarbeitung der Instruktionen nennt man Fetch–Decode–Execute-Zyklus.
  - Ausführungseinheit für die arithmetischen Rechenoperationen sowie logische Operationen ist das Rechenwerk (ALU).
  - Registerspeicher dienen dem Zwischenspeichern von Daten aus dem Hauptspeicher, z. Bsp. Befehlszählerregister, das die Speicheradressen von auszuführenden Instruktionen enthält.
    - Vorteil von Registerspeichern: sehr schnelle Schreib- & Lesezugriffe
    - Größe des Befehlszählerregisters bestimmt den maximal adressierbaren Hauptspeicher: ein 32-Bit breites Register kommt auf 2^{32} Speicherplätze, was einer maximalen Hauptspeicherkapazität von 4 GB entspricht.
  - Unterscheidung zwischen CISC- & RISC-Prozessoren (Complex & Reduced Instruction Set Computing), was den Umfang der Maschinenbefehlssätze beschreibt.

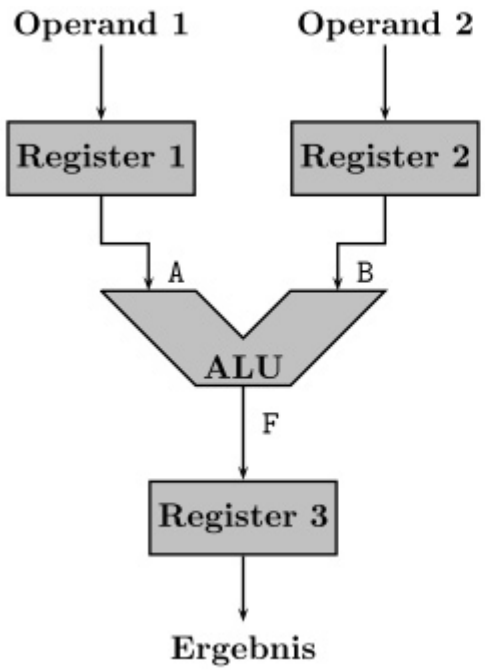
- Das Rechenwerk:
  - Dient der Ausführung arithmetischer und logischer Operationen.
  - Die ALU verknüpft die beiden Eingänge A und B gemäß der in der Instruktion angegebenen Operation und gibt das Resultat am Ausgang F aus.
  - Welche Operation auszuführen ist, wird durch den Opcode festgelegt, der Teil der Instruktion ist.
  - Ein Befehlsdekodierer, eine Komponente des Steuerwerks, wandelt den Opcode in Steuersignale um.
  - Diese Steuersignale wiederum initialisieren die entsprechenden Einheiten der ALU.
  - Arithmetische Operationen: Addition, Subtraktion, Multiplikation, Division
  - Logische Operationen: AND, OR, XAND, XOR, Negation, Vergleiche (<, >, ==, <>)

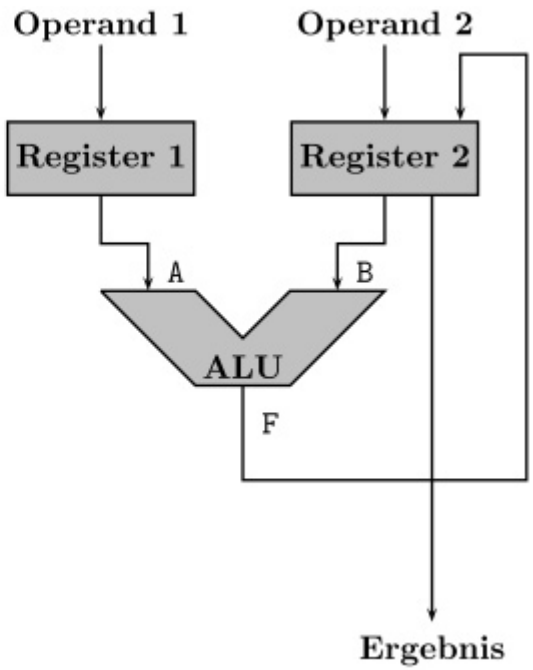


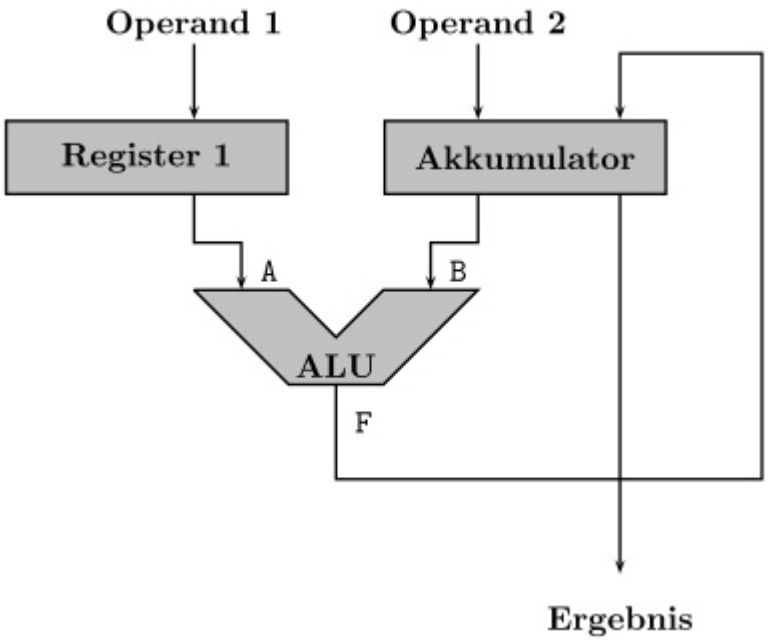
- Das Rechenwerk:
  - Zur Auswertung der Ergebnisse von Operationen besitzt eine CPU das sogenannte Flag-Register.
  - Gesetzte, bzw. nicht gesetzte Bits signalisieren die folgenden Angaben:
    - Das **Zero Bit** zeigt an, ob das Resultat einer ALU Operation Null ist.
    - Das **Sign Bit** zeigt an, ob das Ergebnis (bei vorzeichenbehafteten Zahlen) negativ ist.
    - Das **Carry Bit** zeigt an, ob ein Übertrag in der höchsten Ergebnisstelle bei der Ausführung einer Addition oder Subtraktion aufgetreten ist.
    - Das **Overflow Bit** zeigt an, ob der Zahlenbereich (falls es sich um eine arithmetische Operation mit Zahlen in der Komplementdarstellung handelt) überschritten wurde.



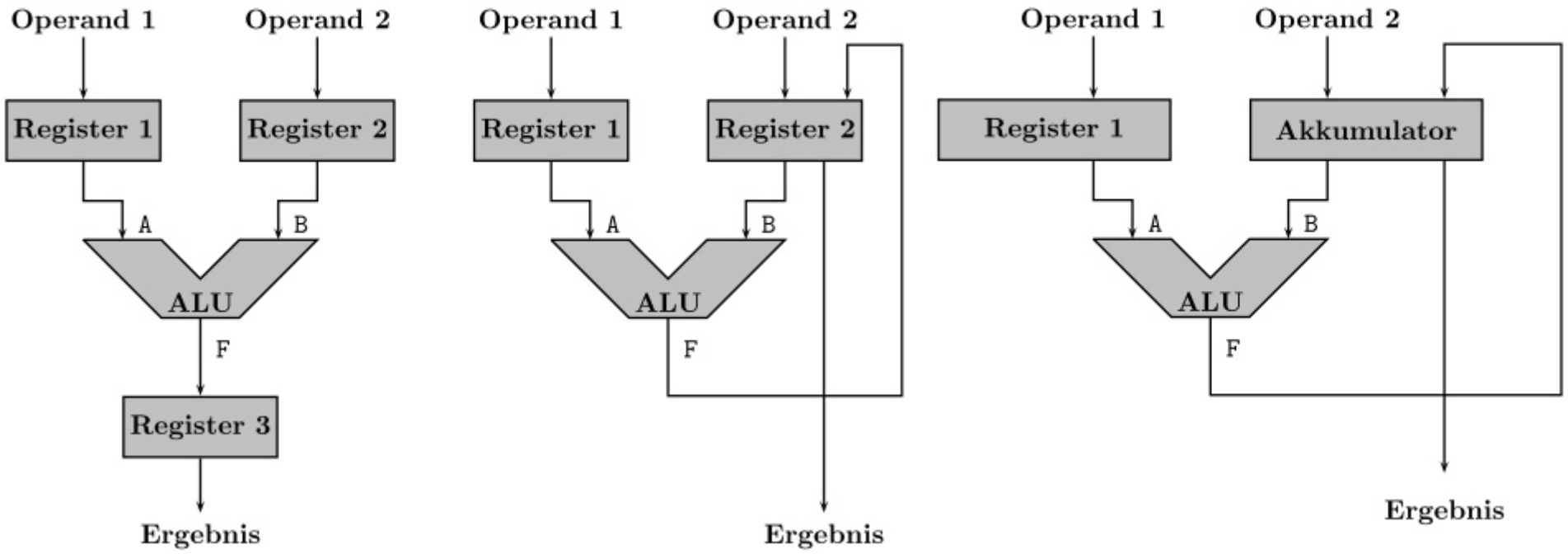
- Das Rechenwerk
  - Unterscheidung zwischen Drei-, Zwei- & Einadressmaschine.
  - Zweiadressmaschine: Schaltungsaufwand reduziert sich bei Einsparung des Ergebnisregisters:
    - Ergebnis überschreibt den Inhalt von Register 2



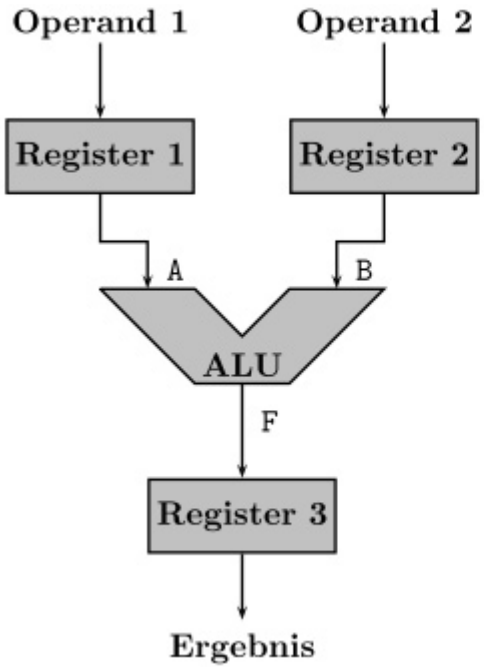


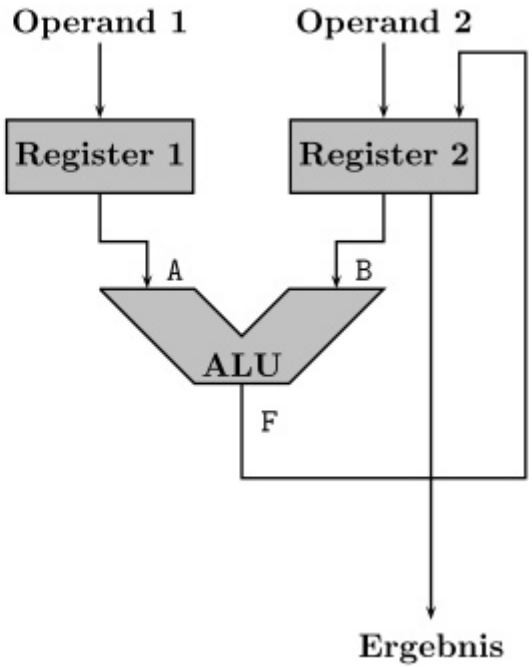


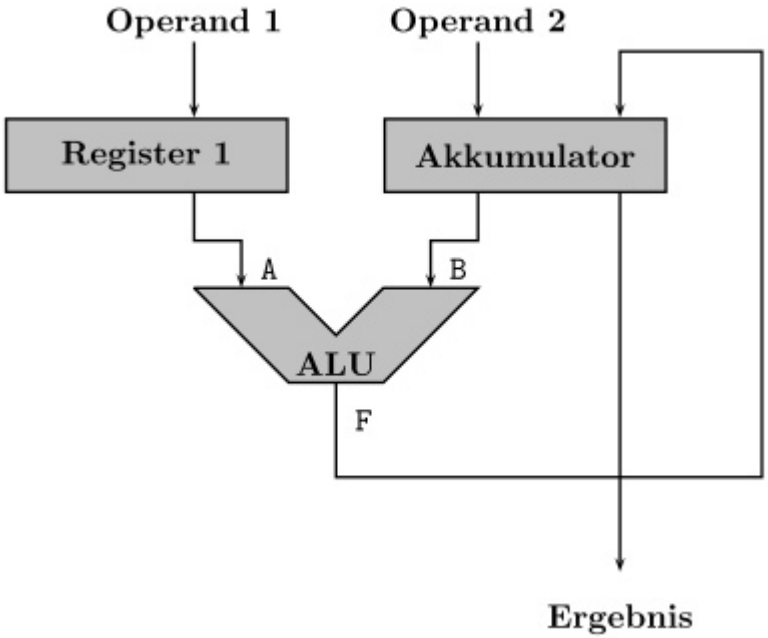
- Das Rechenwerk
  - Einadressmaschine: Es gibt nur noch 1 adressierbares Register. Der zweite Operand einer Berechnung sowie das anschließende Ergebnis werden automatisch in den Akkumulator geladen. Vorteil: kurze Befehlsformate für die Maschineninstruktionen



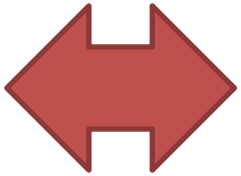
- Das Rechenwerk
  - Die wichtigsten heutigen Prozessoren sind Zweiadressmaschinen.





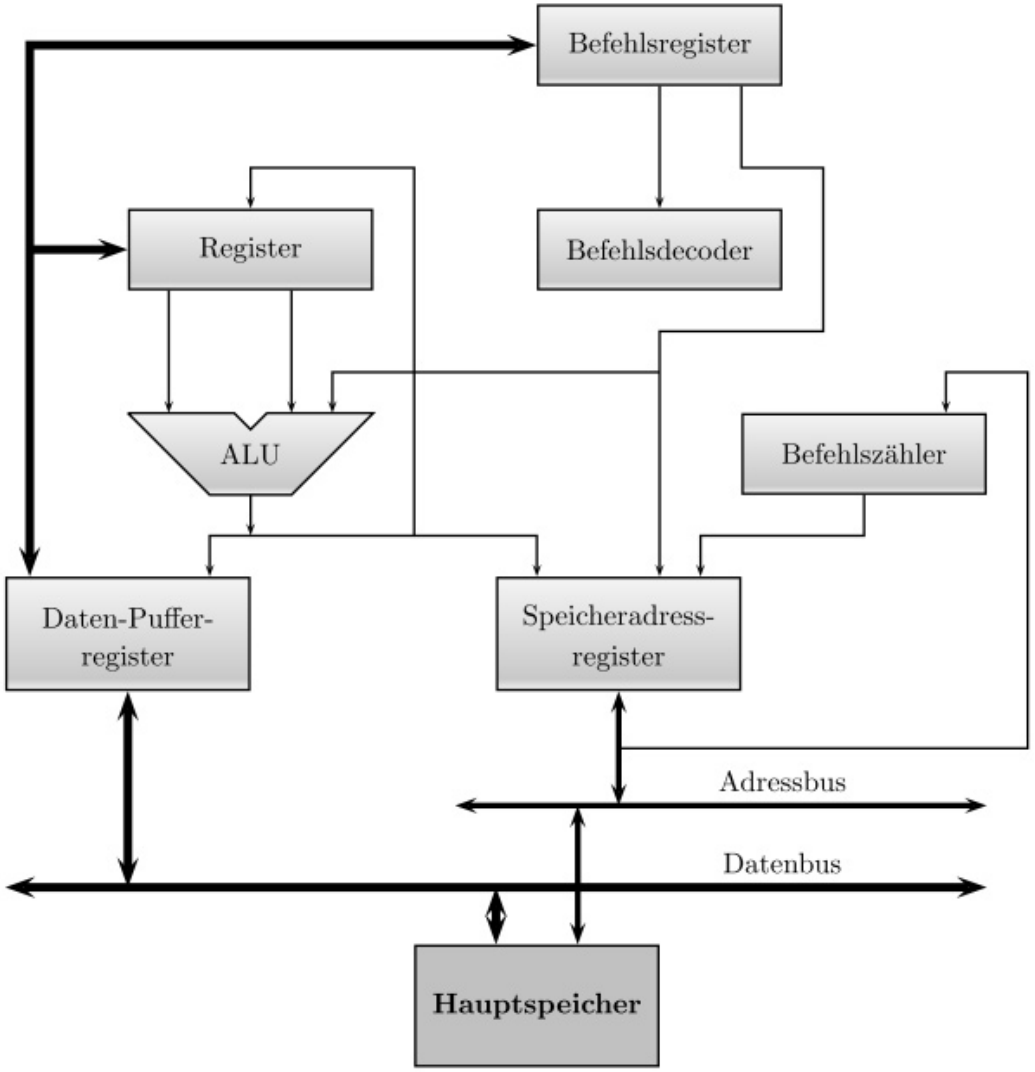


Flexibler & Leistungsfähiger
Längere Instruktionen
Mehr Speicherverbrauch

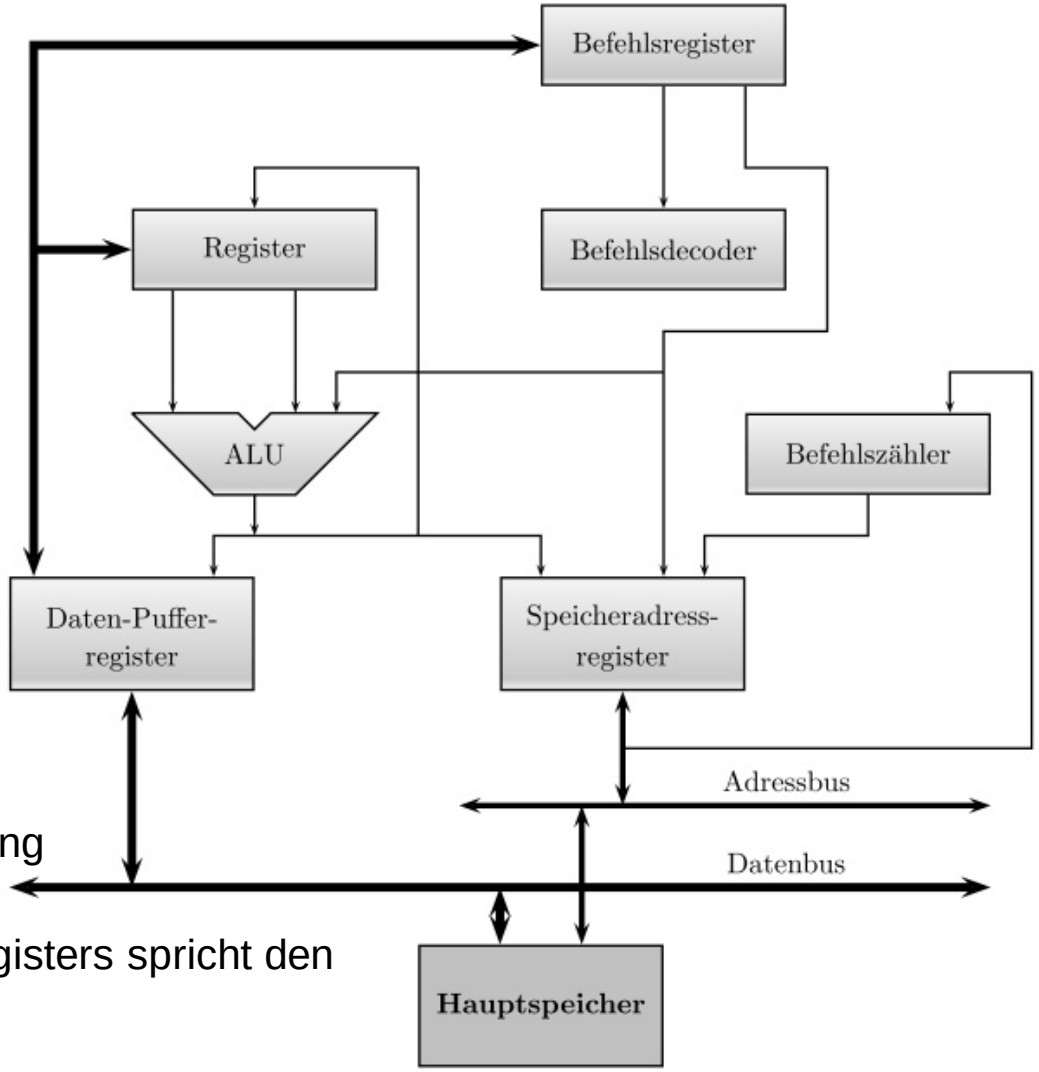


Kürzere Instruktionen
Weniger Speicherverbrauch
Weniger Flexibilität & Leistung

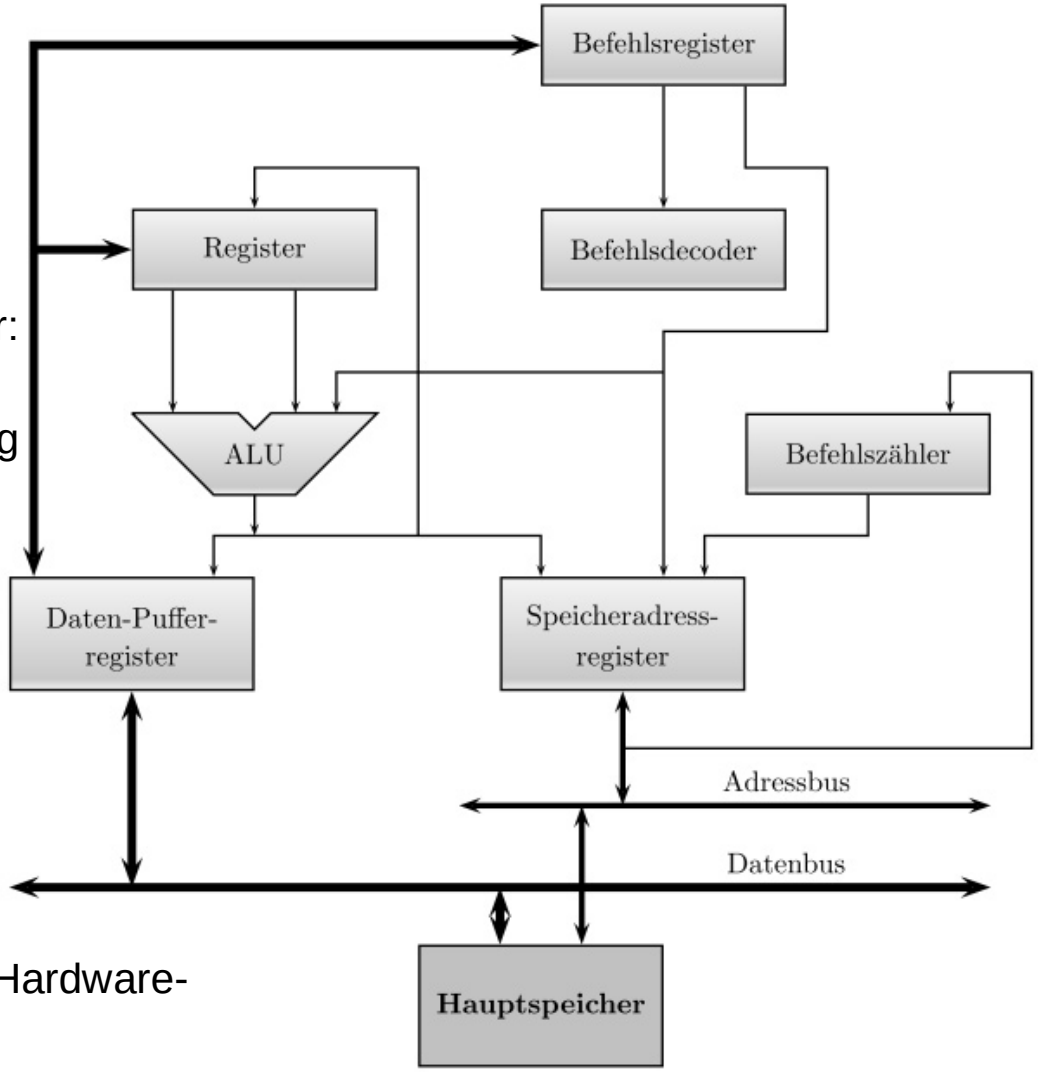
- Das Steuerwerk:
  - Dient der Koordination der temporären Abläufe im Rechner (FDE-Zyklus)
    - Holen der auszuführenden Instruktionen aus dem Arbeitsspeicher (Fetch)
    - Dekodieren der Instruktionen (Decode)
    - Sowie die Steuerung der Ausführung (Execute)
  - Abbildung: Struktur und Datenfluss in einer Zentraleinheit



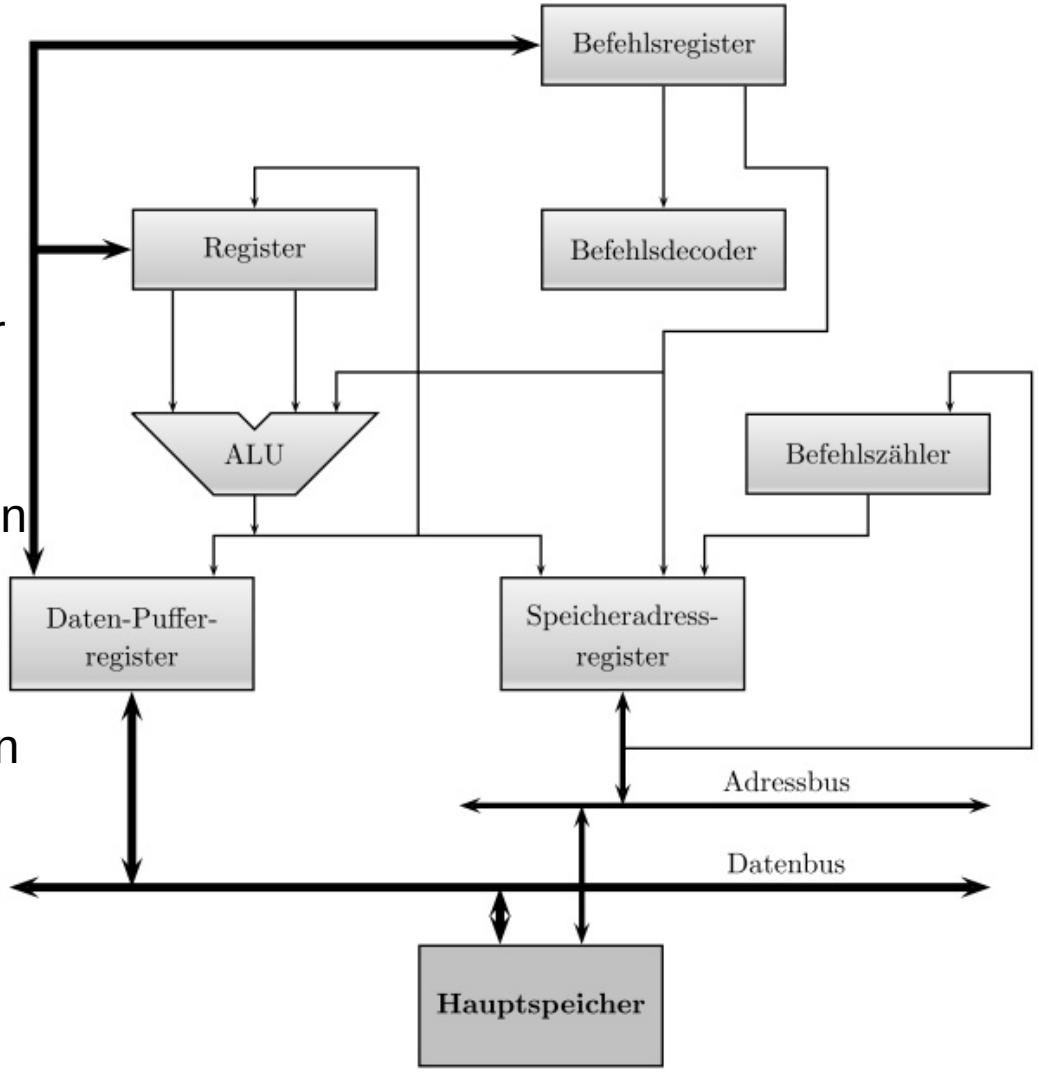
- Aufbau des Steuerwerks:
  - **Befehlsregister** oder Instruction Register (IR):
    - Beinhaltet den jeweils aktuellen Befehl der gerade abgearbeitet wird.
    - Spezielles Register der CPU, das anwenderseitig nicht ansprechbar ist.
  - **Befehlszähler** oder Instruction Counter (IC):
    - Wie das IR ebenfalls ein reserviertes CPU-Register
    - Enthält die Arbeitsspeicheradresse des gerade in Abarbeitung befindlichen Befehls.
    - Erhöhung des Inhalts dieses Registers spricht den nächsten Befehl an.



- Aufbau des Steuerwerks:
  - **Speicheradressregister** oder Memory Adress Register (MAR):
    - Beinhaltet entweder die Hauptspeicheradresse der nächsten auszuführenden Instruktion, oder:
    - Die Haupspeicheradresse eines Datenwortes, falls zur Ausführung einer Instruktion ein Datenwort aus dem Hauptspeicher in ein CPU-Register geladen werden oder von der CPU in den Hauptspeicher transferiert werden muss.
  - **Befehlsdecoder**:
    - Decodiert die Maschineninstruktionen und generiert die Steuersignale zu den entsprechenden Hardwarekomponenten.



- Aufbau des Steuerwerks:
  - Im Speicheradressregister wird jede Speicheradresse solange zwischengespeichert, bis der Systembus frei ist. Erst dann kann das Speicheradressregister die Hauptspeicheradresse auf den Adressbus legen.
  - Die analoge Aufgabe dazu für den Datentransfer erfüllt das Daten-Puffer-Register. Daten, die die zentrale Recheneinheit über den Systembus zum Ziel transferieren muss, müssen solange gepuffert werden, bis der Datenbus frei ist.



- Aufbau des Steuerwerks:
  - Das Steuerwerk erhält die durchzuführenden Aufgaben durch die Instruktionen des in der Abarbeitung befindlichen Programms. Diese Maschinenbefehle geben an, welche Operation mit welchen Daten (Operanden) auszuführen sind und an welche Hauptspeicherstelle das Resultat abzuspeichern ist.
  - Maschinenbefehle bestehen daher aus 2 Teilen:
    - **Opcode** oder Operation Code: gibt die Art des auszuführenden Befehls an.
    - **Operandenteil** oder Operand Specifier: enthält, je nach Rechenart, die Hauptspeicheradressen und/oder Registeradressen der Operanden, auf die die Operation angewendet werden soll.
  - Beim Operandenteil differenziert man zwischen Null-, Ein-, Zwei und Dreiadressbefehlen.
  - Nulladressinstruktionen werden eingesetzt bei der sogenannte Kellerarchitektur (push- & pop-Operationen).

- Der Opcode:
  - Einadressbefehl:
    - Der Operandenteil der Instruktion gibt die Adresse eines Operanden an, der andere Operand wird automatisch in den Akkumulator geladen. Das Resultat der Operation wird ebenfalls automatisch in den Akkumulator geschrieben.

  - Zweiadressbefehl:
    - Im Operandenteil der Instruktion stehen die Adressen beider Operanden. Das Ergebnis wird je nach Rechnertyp unter die Registeradresse des 1. oder 2. Operanden gespeichert. Ein Akkumulator wird bei diesem Design nicht benötigt.

    - Beispiel einer Assembleranweisung für Intel 80x86: *add AX, BX*

      Durch diese Anweisung werden die Inhalte der beiden Register AX und BX addiert, das Ergebnis wird im Register AX abgelegt.

- Der Opcode:
  - Dreiadressbefehl:
    - Zusätzlich zu den beiden Operanden wird noch die Adresse (z.Bsp. ein Register) für das Ergebnis in der Instruktion angegeben.

- Anmerkung: Eine in einer Hochsprache wie zum Beispiel C formulierte Anweisung wie sum = a + b; verwendet drei Variablen. Prinzipiell sind daher zumindest 3 Adressen not-wendig, diese Hochsprachenanweisung in Maschinensprache umzusetzen. Der Compiler der Hochsprache muss nun diese Anweisung in das jeweilige Adressformat der zugrundeliegenden Struktur der Maschinenbefehle übersetzen. Die folgende Tabelle zeigt die unterschiedlichen Implementierungen von Maschineninstruktionen.

Prozessor	Anzahl Instr.	Länge
IBM 370	≈ 140	2,4,6, Byte
DEC VAX	≈ 330	2 - 16 Byte
Motorola 68000	≈ 83	2 - 5 Byte
8086	90	1 - 6 Byte

- Der Fetch-Decode-Execute Zyklus
  - Die vom Steuerwerk koordinierte Ausführung der Maschineninstruktionen wird bei einer Top-Level Betrachtung in die folgenden drei Phasen aufgeteilt:
    - **FETCH**: Holen von Befehlen
      Der Inhalt der von dem Befehlszähler (IC) bestimmten Speicherzelle wird aus dem Arbeitsspeicher in das Befehlsregister (IR) geladen.
    - **DECODE**: Entschlüsseln von Befehlen
      Der Befehl wird in seine Bestandteile zerlegt, nämlich in
      - Operationsteil
      - Adressteil
      - Operandenteil

      Die Steuereinheit muss erkennen, um welche Befehlsart es sich handelt.
    - **EXECUTE**: Initiierung der Befehlsausführung
      Versorgung aller an der Befehlsausführung beteiligten Funktionseinheiten mit den notwendigen Steuersignalen (durch eine Folge von Mikrooperationen).
      Z.Bsp. Adressierung & Laden von Operanden, Speichern von Ergebnissen oder Veränderung des Befehlszählers. Wenn alle notwendigen Signale gesetzt sind, beginnt automatisch die Ausführung des Befehls.

- Der Fetch-Decode-Execute Zyklus – Beispiel
  - Als Beispiel untersuchen wir hier die Ablaufsteuerung bei der Abarbeitung der Instruktionsfolge

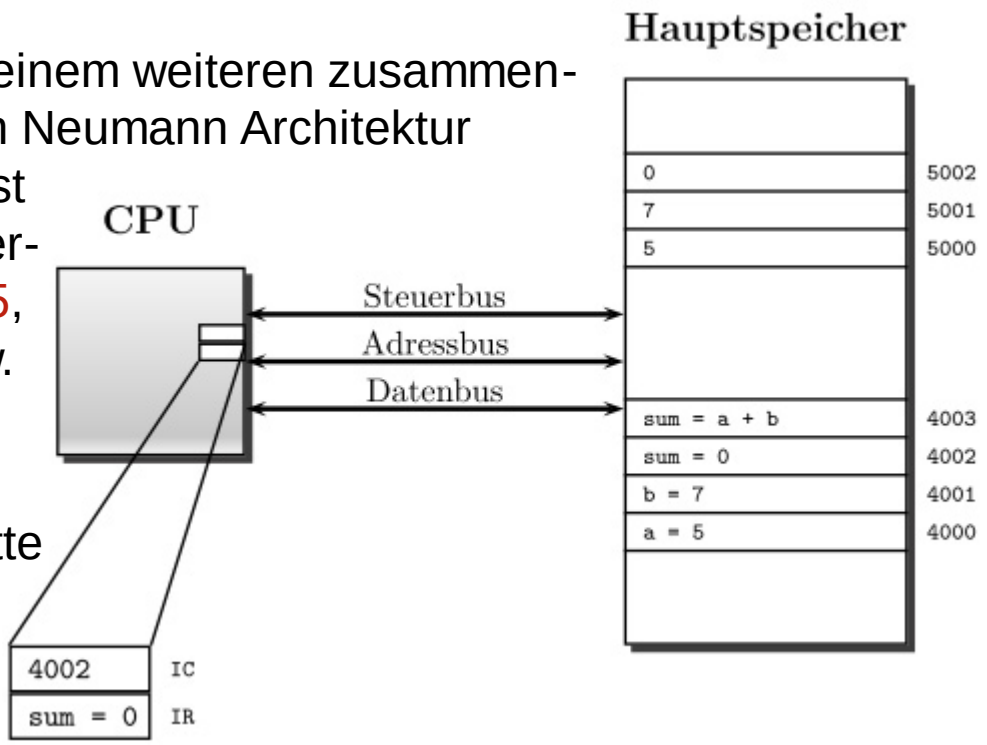
```
int a,b,sum;

a = 5;
b = 7;
sum = 0;

sum = a + b;
```

  - Vereinfachend nehmen wir dabei an, dass direkt auf den Hauptspeicher zugegriffen wird, d.h. wir ignorieren den Zwischenschritt, dass der Compiler die Additions-anweisung der Variablen beispielsweise in die Einzelschritte

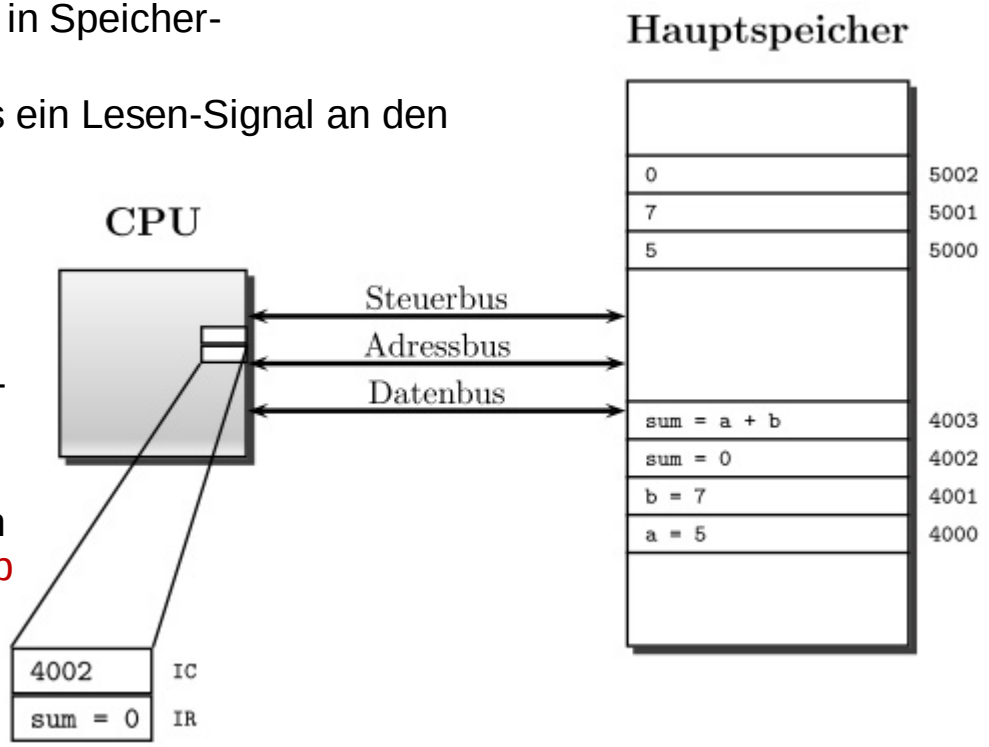
```
load R1,a
load R2,b
add R3,R2,R1
store sum,R3
```

    auflöst, wobei R1,R2,R3 drei CPU Allzweckregister bezeichnet.

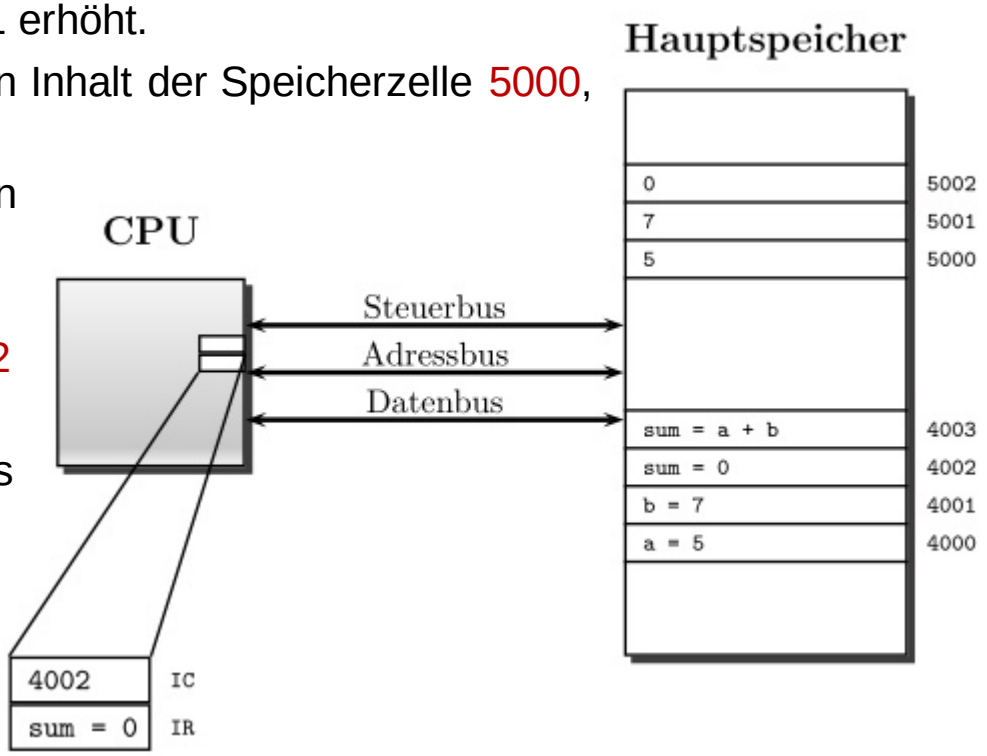
- Der Fetch-Decode-Execute Zyklus – Beispiel
  - Durch die Deklaration int a,b,sum; wird im Hauptspeicher Speicherplatz reserviert für drei Integer Variablen. Der Compiler weist diesen Variablen beispielsweise die Speicheradressen 5000 bis 5002 zu. Die Daten eines Programms werden also in einem zusammenhängenden Speicherbereich abgelegt, den Datenbereich.
  - Die Instruktionen selbst werden in einem weiteren zusammenhängenden Bereich gemäß der von Neumann Architektur sequentiell abgespeichert — dies ist der Programmteil — in der Speicherzelle 4000 liegt die Instruktion a = 5, in 4001 die Initialisierung b = 7 usw. und in der Speicherzelle 4003 die Addition sum = a + b.
  - Wir wollen nun die einzelnen Schritte betrachten, die dieses System auszuführen hat, wenn die Instruktion sum = a + b abgearbeitet wird.



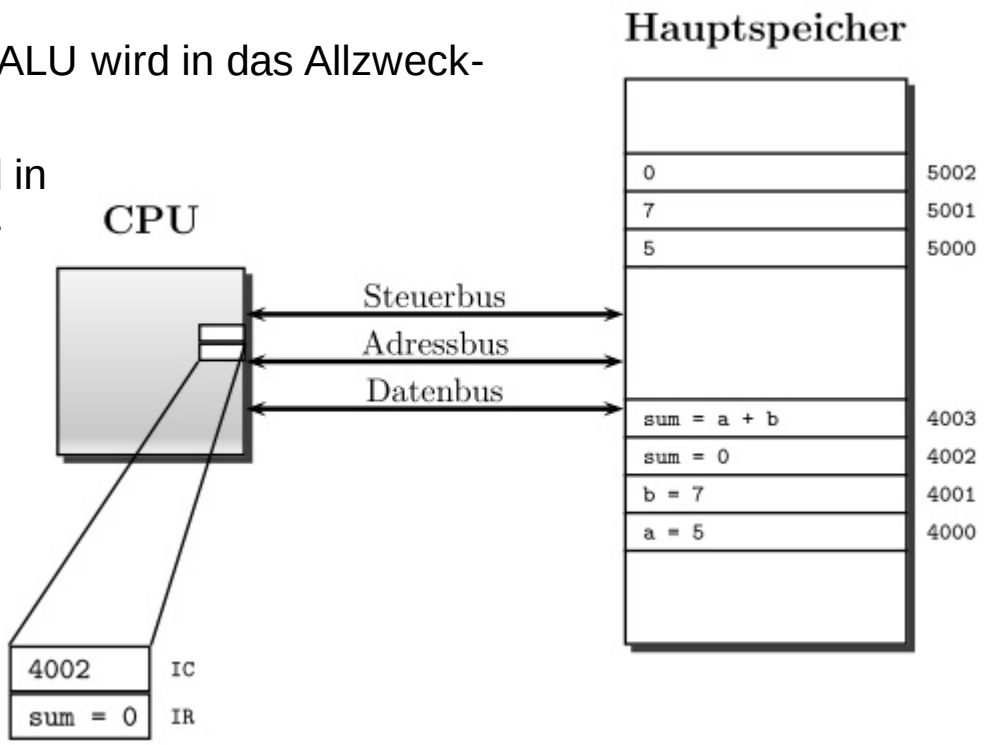
- Der Fetch-Decode-Execute Zyklus – Beispiel
  - Für die Ausgangssituation nehmen wir an, dass im Befehlszähler IC die Hauptspeicheradresse 4002 liegt, im Befehlsregister selbst die Instruktion sum = 0. Die CPU hat gerade die Bearbeitung dieser Instruktion beendet.
  - **Fetch**: Der Befehlszähler IC wird vom Steuerwerk um 1 erhöht. Im Instruction Counter steht nun der Wert 4003.
    - **1.** Inhalt des Befehlszählers (IC) wird in Speicher-adress-Register (MAR) geladen.
    - **2.** Steuerwerk sendet über Steuerbus ein Lesen-Signal an den Hauptspeichercontroller.
    - **3.** Die im Speicheradress-Register (MAR) liegende Instruktionsadresse 4003 wird über Adressbus an Haupt-speichercontroller übertragen. Mit Hilfe des Decoders wird die so adres-sierte Speicherzelle angesprochen.
    - **4.** Hauptspeichercontroller gibt den Inhalt der Speicherzelle 4003 auf den Datenbus. Die Instruktion sum = a + b wird zur CPU übertragen, in das Daten-Puffer Register geladen und anschließend in das Befehlsregister.



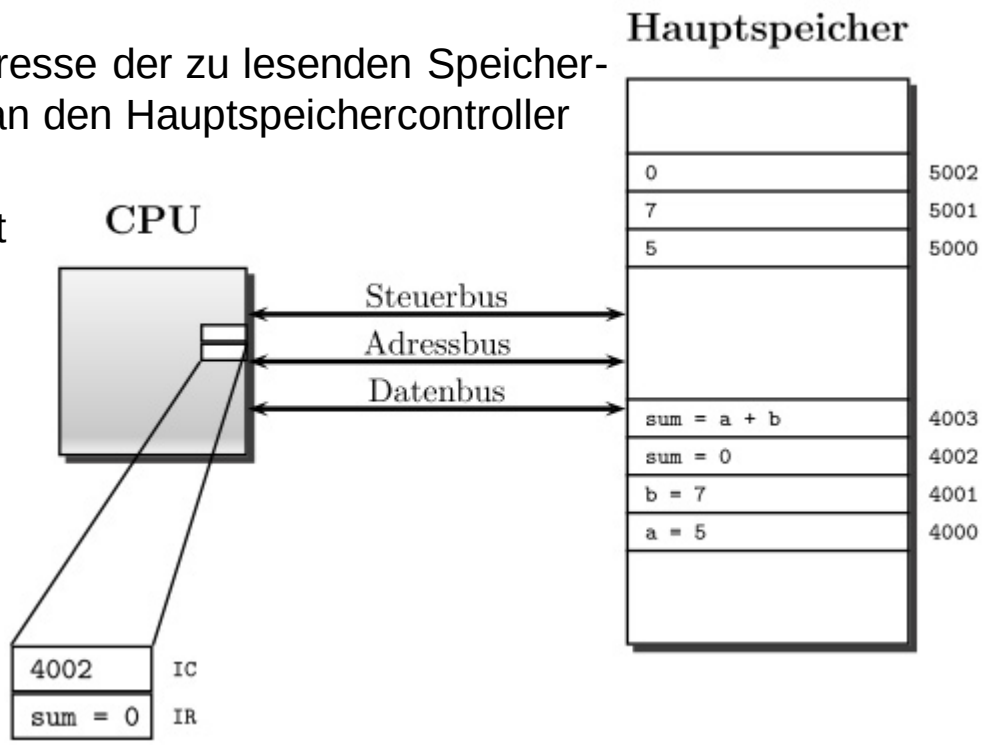
- Der Fetch-Decode-Execute Zyklus – Beispiel
  - **Decode**: Sobald die Instruktion sum = a + b in das Befehlsregister geladen ist, ist die Fetch-Phase beendet und die Decode Phase beginnt.
    Der Befehlsdekodierer muss nun die Instruktion sum = a + b in die Bestandteile zerlegen und führt dabei folgende Schritte aus:
    - **1.** Der Befehlszähler (IC) wird um 1 erhöht.
    - **2.** Lade den Operanden a, bzw. den Inhalt der Speicherzelle 5000, in ein Allzweckregister (z.B. R1)
    - **3.** Lade den Operanden b, bzw. den Inhalt der Speicherzelle 5001, in ein Allzweckregister (z.B. R2)
    - **4.** Verbinde die Register R1 und R2 mit den Eingängen der ALU
    - **5.** Steuersignal an die ALU, so dass gemäß Opcode das Addierwerk initialisiert wird
    - **6.** Der Ausgang der ALU wird auf ein freies Allzweckregister z.B. R3 geschaltet.



- Der Fetch-Decode-Execute Zyklus – Beispiel
  - **Execute**: In der Ausführphase können folgende Schritte durchgeführt werden
    - **1.** Der Inhalt der Register R1, R2 wird in die ALU geladen und die Addition – z.B. durch ein Ripple Carry Adder ausgeführt.
    - **2.** Die ALU sendet gegebenenfalls Steuersignale, um bestimmte Flags im Flag-Register zu setzen.
    - **3.** Das Resultat der Addition in der ALU wird in das Allzweck-register R3 geschrieben.
    - **4.** Der Inhalt des Registers R3 wird in die Speicherstelle 5002 des Haupt-speichers zurück geschrieben.



- Der Fetch-Decode-Execute Zyklus – Beispiel
  - **Anmerkungen**: Jeder der im obigen Ablauf auftretenden Zugriffe auf den Hauptspeicher - dies betrifft insbesondere den Transfer der Operanden vom Hauptspeicher in die CPU - muss folgende Schritte durchlaufen:
    - **1.** Das Steuerwerk sendet über den Steuerbus ein 'Lesen' Signal an den Hauptspeichercontroller.
    - **2.** Über den Adressbus wird die Adresse der zu lesenden Speicherzelle vom Speicheradressregister an den Hauptspeichercontroller transferiert.
    - **3.** Der Hauptspeichercontroller liest den Inhalt der angeforderten Speicherzelle aus.
    - **4.** Der Hauptspeichercontroller transferiert den Inhalt der angeforderten Speicherzelle über den Datenbus zur CPU in das Daten-Puffer-Register.



- Der Fetch-Decode-Execute Zyklus – Beispiel
  - **Anmerkungen**: Die obige detailliertere Betrachtung des Fetch-Decode-Execute-Zyklus legt es nahe, die Ausführung einer Instruktion in fünf Phasen einzuteilen:
    - die Fetch–Phase (Holen der Instruktion)
    - die Decode–Phase (Dekodieren der Instruktion)
    - die Operanden–Holphase (Laden der Operanden in Allzweck-register)
    - die Execute-Phase (Ausführung der Operation)
    - die Write-Back-Phase (das Zurück-schreiben des Resultats in den Hauptspeicher)