DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

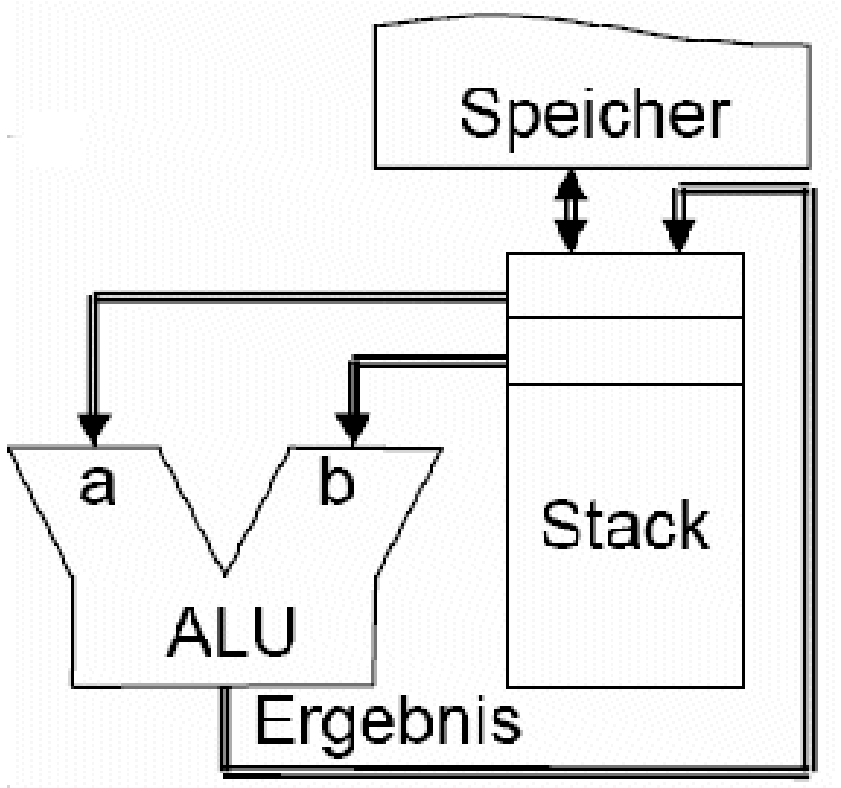
Eigenschaften von Prozessoren

Klassifizierung von Prozessoren

Kann erfolgen nach:

- Operandenstruktur der ALU: Stackzugriff, Akkumulator, Register-Register, Register-Speicher
- Busaufbau: v. Neumann, Harvard
- Befehlssatzumfang: CISC, RISC (complex or reduced instruction set computer)
- Speicherorganisation: Little endian, Big endian
- Befehlssatzdesign: 4, 3, 2, 1, 0 Operanden

1. Klassifizierung gemäß *Operandenstruktur der ALU*: 1a) Stack-Architektur



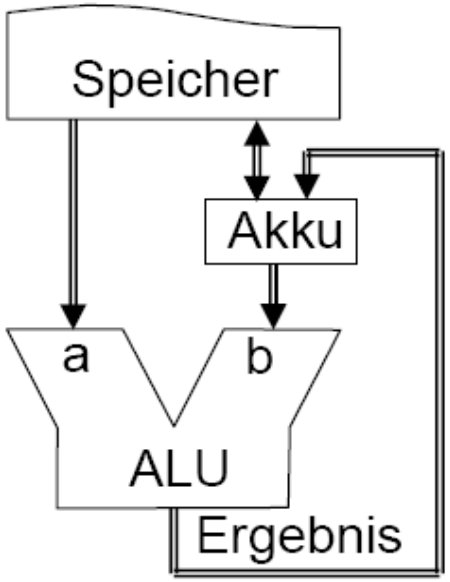
Stacks werden unabhängig von der jeweiligen Architektur bei Unterprogrammaufrufen und Parameterübergaben verwendet.

Stackarchitekturen besitzen keine Register für explizite Berechnungen.

PUSH und POP erfolgen in diesem Fall zwischen Datenspeicher und Stack(-Speicher).

```
PUSH Op_A     ; Von Speicher auf Stack
PUSH Op_B
ADD           ; in ALU
POP  Op_C     ; Von Stack in Speicher
```

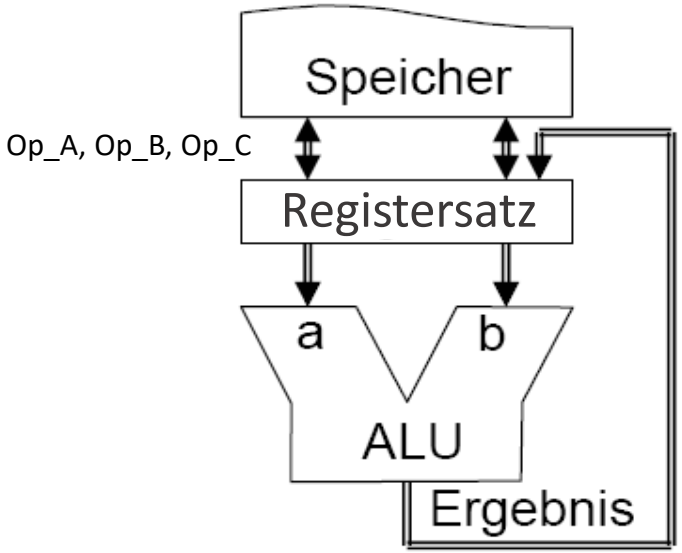
1. Klassifizierung gemäß *Operandenstruktur der ALU* : 1b) Akkumulator-Architektur



- Ausgezeichnetes Register: Akkumulator (bisher: ACC)
- LOAD und STORE wirken nur auf Akkumulator. Er ist als impliziter Operand an jeder Operation beteiligt. Jede Operation braucht nur eine Adresse
- Sehr kompaktes Befehlsformat

LDA	Op_B ;	Op_B von Speicher -> Akku
ADD	Op_A ;	Akku = Akku + Op_A
STO	Op_C ;	Akku -> Speicher (Op_C)

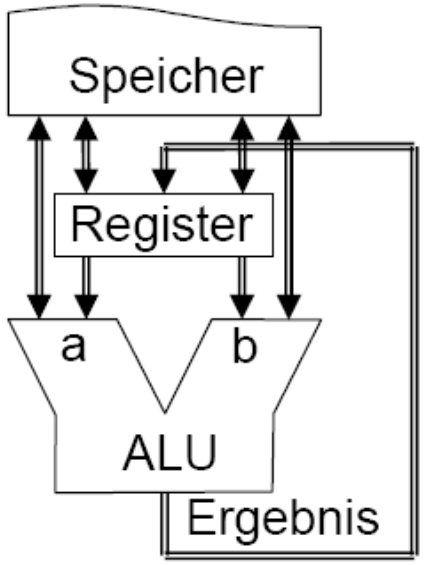
1. Klassifizierung gemäß *Operandenstruktur der ALU* : 1c) Register-Register-Architektur (Load-Store)



- RISC (Load-Store-Architektur)
- alle Rechenoperationen greifen nur auf Register zu,
- nur die Befehle LOAD und STORE greifen auf den Speicher zu
- Registersatz: 32 – 512 Register verfügbar
- einfaches Befehlsformat fester Länge
- alle Instruktionen brauchen in etwa gleich lange

LOAD	R1, Op_A ;	lade Op_A aus Speicher in R1
LOAD	R2, Op_B ;	lade Op_B aus Speicher in R2
ADD	R3, R1, R2 ;	addiere R1 und R2, Ergbn. -> R3
STORE	Op_C, R3 ;	speichere R3 -> Op_C (=Op_A + Op_B)

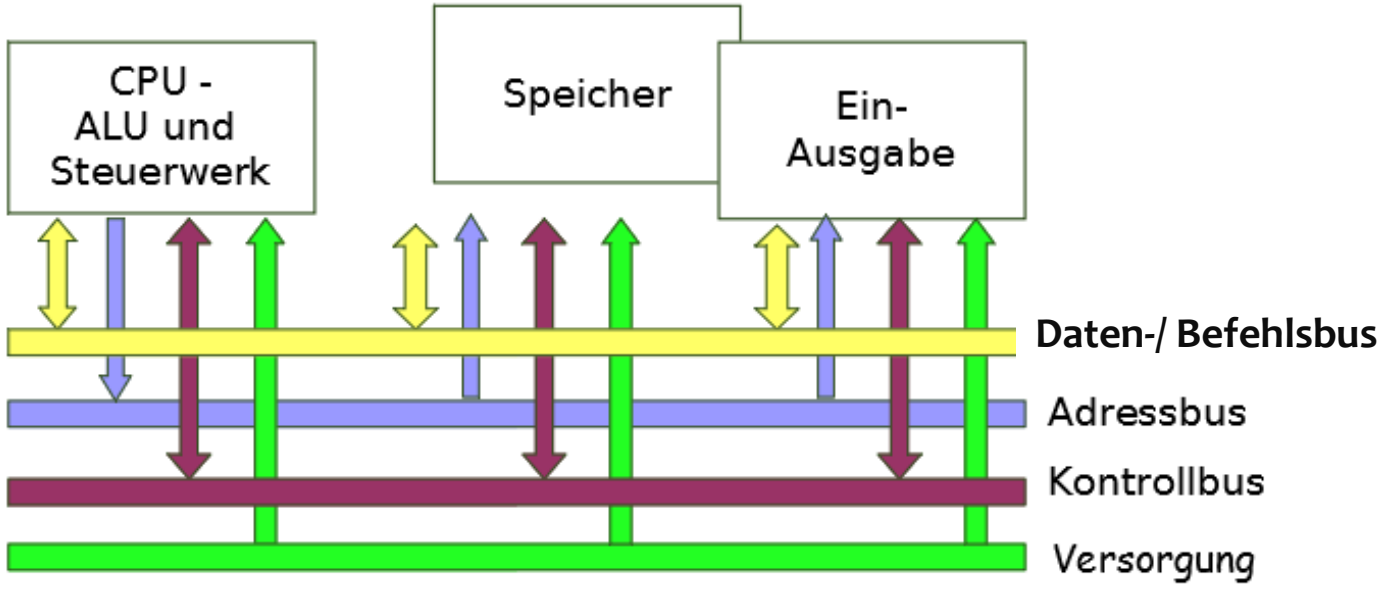
1. Klassifizierung gemäß *Operandenstruktur der ALU* : 1d) Register-Speicher-Architektur



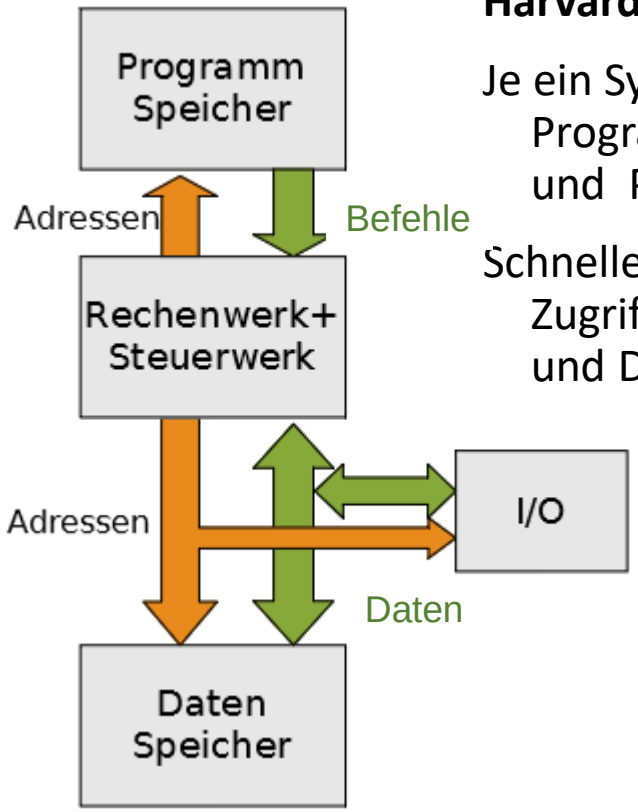
- CISC (Mischung von Akkumulator- und Load-Store-Architektur)
- Operationen greifen auf Register und/oder Speicher zu
- Befehlsformat variabler Länge
- mächtige Befehle
- stark unterschiedliche Zeiten für Instruktionsausführung

2. Klassifizierung gemäß *Busaufbau*: Bussysteme

- Ein Systembus ist aus einem Daten-/Befehls-Bus, Adressbus und Kontrollbus sowie einem Bus zur elektrischen Versorgung der Komponenten aufgebaut.
- Einige Architekturen beinhalten zusätzlich noch einen I/O-Bus.



2. Klassifizierung gemäß *Busaufbau*: Vergleich Harvard / von Neumann Architektur



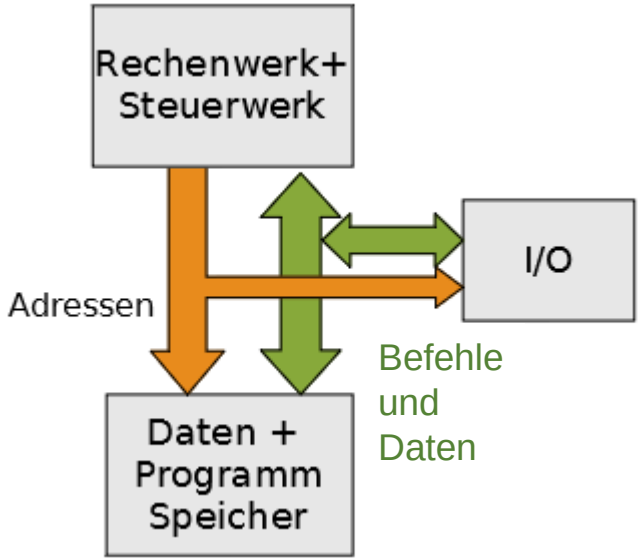
Harvard - Architektur

Je ein Systembus für Programmbefehle- und Programmdaten

Schneller gleichzeitiger Zugriff auf Programm und Daten

Von Neumann - Architektur

Nur ein Bus für Befehle und Daten



3. Klassifizierung gemäß *Befehlssatzumfang*: Orthogonale Befehlssätze

Wenn jeder Opcode beliebig mit Adressierungsart und Datentyp kombiniert werden kann, spricht man von einem **Orthogonalen Befehlssatz**

Vorteile:

- Vereinfacht die Nutzung der verfügbaren Instruktionen

Nachteile:

- Sehr umfangreiche Befehlssätze durch Kombinatorik

3. Klassifizierung gemäß *Befehlssatzumfang* : 3a) CISC Kriterien (Complex Instruction Set Computing)

Befehle unterschiedlicher Länge von 1 - 17 Byte (Intel-x86-Prozessorfamilie (ab Pentium inkl. RISC-Eigenschaften), Motorola 68000, Zilog Z80).

+ Speichereffizient durch optimale Befehlslängen
– Komplexe Befehlsdekodierung

Komplexer Befehlssatz

- Anpassung an Compiler-Hochsprachenkonstrukte im Assembler zur Erleichterung der Assemblerprogrammierung
- Kurze Programme, da mächtige Befehle (SW-Mikroprogramme werden dafür ausgeführt)

Direkte Operationen im Speicher

- Register waren teuer, daher nur wenige Register
- Unterschied in Zugriffszeit zwischen Register und Speicher war noch nicht sehr hoch

Komplexe Adressierung des Speichers

- Kurze CPU-Wortlängen (Verarbeitungsbreite) führen zu komplexen Zugriffsverfahren bei der Adressierung in Bereichen, die größer als die Wortlänge der CPU sind

3. Klassifizierung gemäß *Befehlssatzumfang* : 3b) RISC Prinzipien (Reduced Instruction Set Computing)

Grundlegendes Design-Prinzip: Einfachheit (intel 8051, ARM)

- Befehle gleicher Länge (meist 32 Bit Befehlslänge)
- Abarbeiten mit gleicher Taktzahl: erlaubt Befehlspipelines
- Eingeschränkter Befehlssatz (32 - 128 Befehle)
- Explizite Lade/Speicher-Befehle (Load/Store-Architektur)
- 3-Operanden-Befehle
- Hart verdrahtete Befehlsausführer – keine Mikroprogramme, dadurch weniger Befehle, schneller

4. Klassifizierung gemäß *Speicherorganisation:* Little / Big Endian

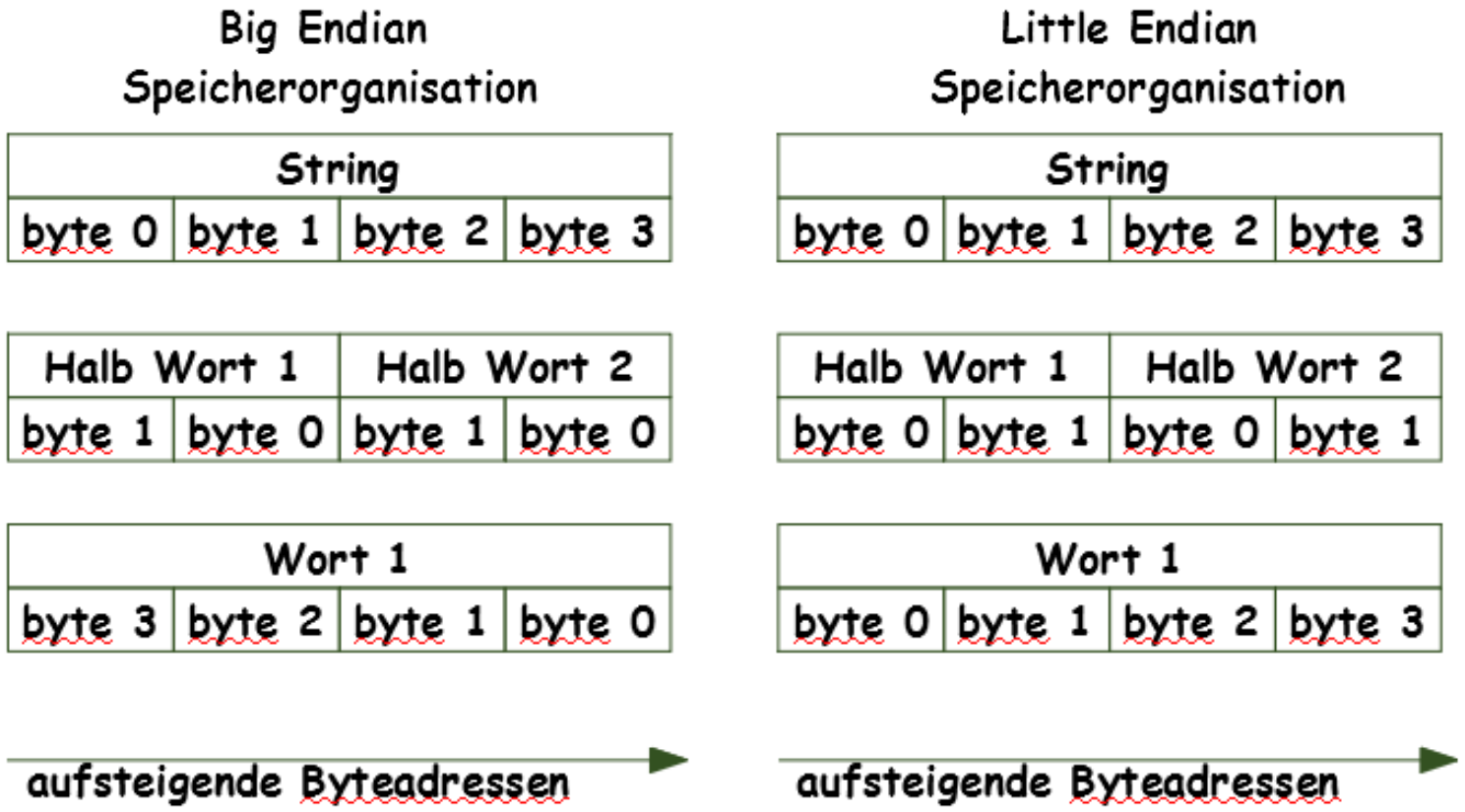
Die Begriffe Big-Endian und Little-Endian stammen aus dem Buch „Gullivers Reisen“ von Jonathan Swift.

Big-Endian beschreibt eine der politischen Fraktionen, die ihre Eier vom größeren Ende aus schält (der „primitive Weg“).

Sie rebellieren gegen den Lilliputanerkönig, da dieser von seinen Untergebenen (die Little Endians) erwartet, dass sie ihre Eier vom kleineren Ende aus öffnen.

4. Klassifizierung gemäß *Speicherorganisation:* Little / Big Endian

Bei Big-Endian wird das „große Ende“ (der signifikanteste Wert in der Sequenz) zuerst abgelegt.
Bsp.: RISC, TCP/IP

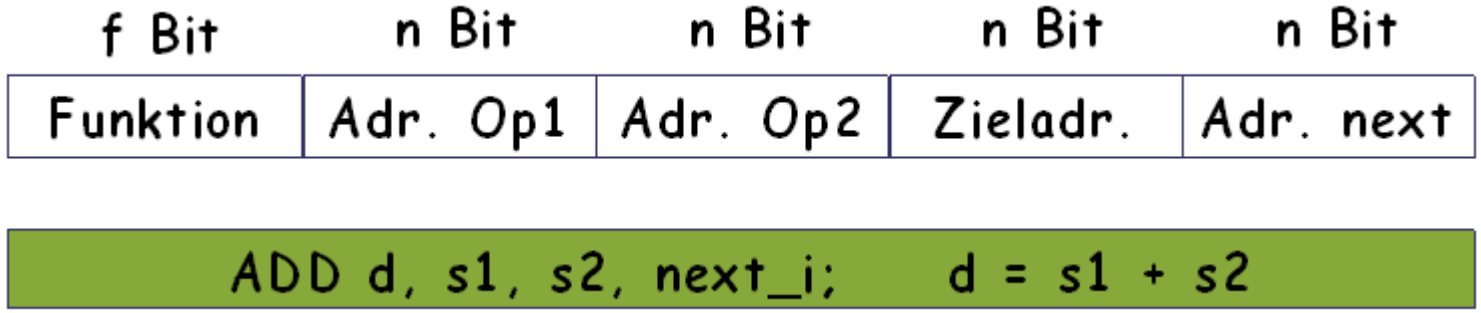


Bytefolge, keine Worte

Bsp.: Intel, DEC

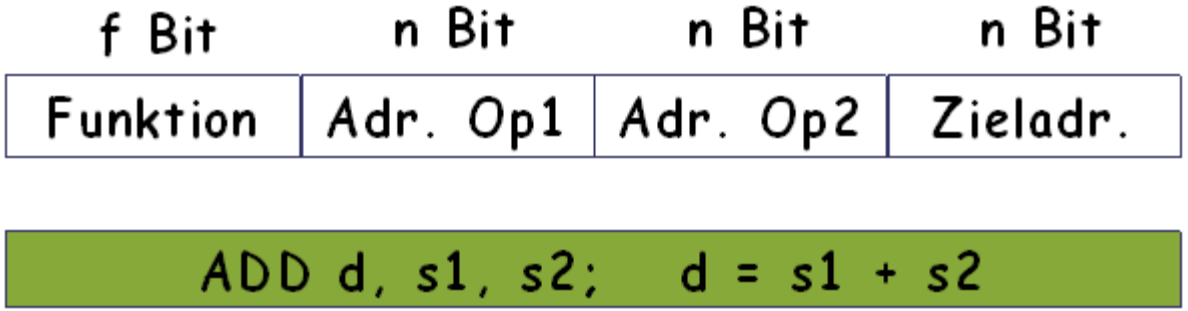
Vorteil: Eine Zahl wird rechts durch Anfügen vergrößert

5. Klassifizierung gemäß *Befehlssatzdesign:* 5a) 4-Adress-Befehle (eigtl: 4-Operanden-Befehle)



- Allgemeinste Form für ein Befehlsformat
- next_i = Adresse des nächsten Befehls
- schwierig zu programmieren
- wird für Microcode verwendet (CISC Mikroprogramme)

5. Klassifizierung gemäß *Befehlssatzdesign:* 5b) 3-Adress-Befehle (eigtl: 3-Operanden-Befehle)

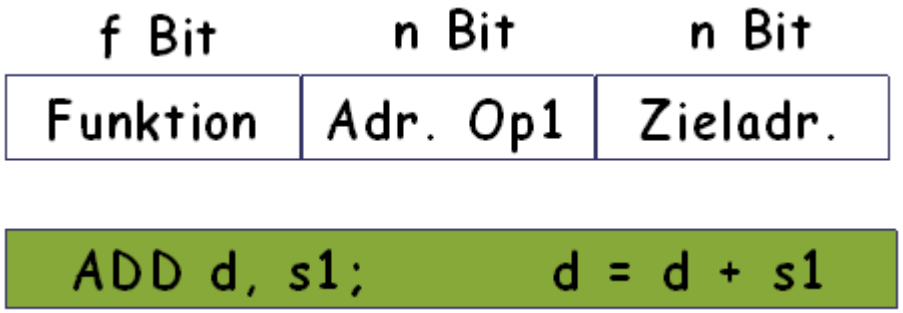


Standard bei RISC Prozessoren, z.B. ARM 32bit

Die 3 Operanden benötigen Platz, so dass dieses Format erst ab einem 32-Bit-Befehlssatz sinnvoll ist.

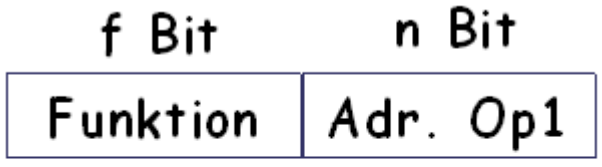
Die nächste Befehlsadresse ist implizit. Nur Sprungbefehle können das implizite Verhalten ändern.

5. Klassifizierung gemäß *Befehlssatzdesign:* 5c) 2-Adress-Befehle (eigtl: 2-Operanden-Befehle)



- Standardformat für 8 und 16 Bit Mikroprozessoren
- Format für die Intel Prozessoren
- Risc Prozessoren mit komprimiertem 16 Bit Befehlssatz nutzen ebenfalls dieses Format (ARM Thumb, MIPS).

5. Klassifizierung gemäß *Befehlssatzdesign:* 5d) 1-Adress-Befehle (eigtl: 1-Operanden-Befehle)




```
ADD s1; acc = acc + s1
```

- Das Zielregister ist implizit und wird häufig Akkumulator genannt.
- Wird im MU0 Design benutzt
- Hohe Befehlsdichte, aber geringe Flexibilität

5. Klassifizierung gemäß *Befehlssatzdesign:* 5e) 0-Adress-Befehle (eigtl: Operandenlose Befehle)

f Bit
Funktion

```
ADD     ;
top_of_stack = top_of_stack + next_on_stack
```

- Beide Operanden und das Ziel sind implizit
- Befehlssatz ist nur für eine Stackarchitektur möglich
- Wird benutzt in der Java Virtual Machine
- Es sind weitere Befehle mit Operanden zum Speichern und Laden des Stacks nötig

Übung

1. Unterschied Harvard – von Neumann?
2. Little Endian: Links steht das höchstwertige Byte?
3. Befehlssatzumfänge?
4. Was heisst „CISC“?
5. RISC-Prinzipien?
6. Welche Operandenstrukturen?

DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

ARM – Generationen und Toolchain …