Rechnerorganisation

Jonas Milkovits

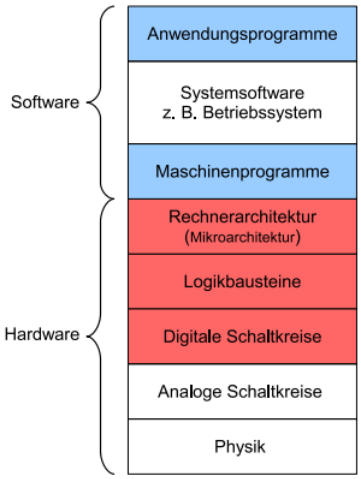
Last Edited: 1. Mai 2020

Inhaltsverzeichnis

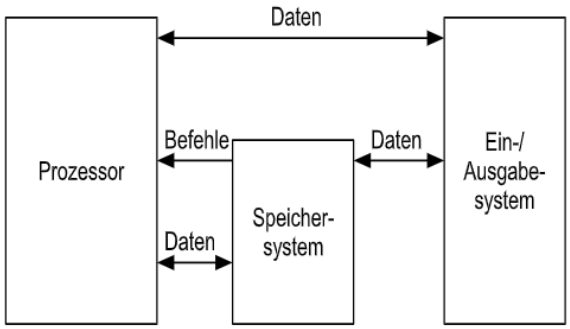
1 Einführung

1.1 Begrifflichkeiten und Grundlagen

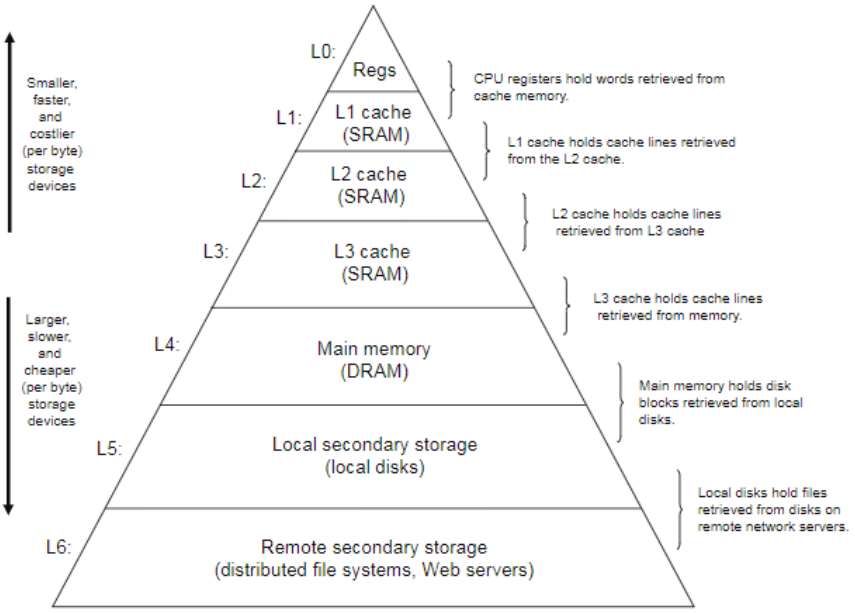
- **Abstraktion**
  - Wichtiges und zentrales Konzept der Informatik
  - Verstecken unnötiger Details (für spezielle Aufgabe unnötig)
- **Schichtenmodell**



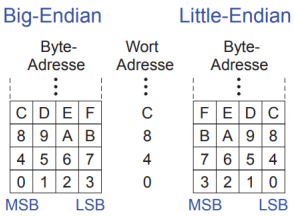
  - Untere Schicht erbringt Dienstleistungen für höhere Schicht
  - Obere Schicht nutzt Dienste der niedrigeren Schicht
  - Eindeutige Schnittstellen zwischen den Schichten
  - Vorteile:
    - Austauschbarkeit einzelner Schichten
    - Nur Kenntnis der bearbeitenden Schicht notwendig
  - Nachteile:
    - ggf. geringere Leistungsfähigkeit des Systems
- **Grundbegriffe**
  - Computer:
    - Datenverarbeitungssystem
    - Funktionseinheit zur Verarbeitung und Aufbewahrung von Daten
    - Auch Rechner, Informationsverarbeitungssystem, Rechnersystem,..
    - Steuerung eines Rechnersystems folgt über ladbares Programm (Maschinenbefehle)
  - Grundfunktionen, die ein Rechner ausführt
    - Verarbeitung von Daten (Rechnen, logische Verknüpfungen,..)
    - Speichern von Daten (Ablegen, Wiederauffinden, Löschen)
    - Umformen von Daten (Sortieren, Packen, Entpacken)
    - Kommunizieren (Mit Benutzer, mit anderen Rechnersystemen)
- **Komponenten eines Rechnersystems**
  - Prozessor
    - Zentraleinheit, Central Processing Unit (CPU)
    - Ausführung von Programmen
  - Speicher
    - Enthält Programme und Daten (Speichersystem)
  - Kommunikation
    - Transfer von Informationen zwischen Speicher und Prozessor
    - Kommunikation mit der Außenwelt (Ein-/Ausgabesystem)



- **Nähere Informationen zum Speicher**
  - Explizite Nutzung des Speichersystem
    - Internet Prozessorspeicher/Register
      - schnelle Register zur temporären Speicherung von Daten/Befehlen
      - direkter Zugriff durch Maschinenbefehle
      - Technologie: Halbleiter ICs
    - Hauptspeicher
      - relativ großer und schneller Speicher für Programme/Daten
      - direkter Zugriff durch Maschinenbefehle
      - Technologie: Halbleiter ICs
    - Sekundärspeicher
      - großer, aber langsamer Speicher für permanente Speicherung
      - indirekter Zugriff über E/A-Programme (Daten → Hauptspeicher)
      - Technologie: Halbleiter ICs, Magnetplatten, optische Laufwerke
      - z.B.: Festplatte
  - Implizite (transparente) Nutzung
    - Für das Maschinenprogramm transparent
    - bestimmte Register auf dem Prozessor
    - Cache-Speicher



  - Speicherorganisation: Big-Endian und Little-Endian



    - Schemata für Nummerierung von Bytes in einem Wort
    - Big-Endian: Bytes werden vom höchstwertigen Ende gezählt
    - Little-Endian: Bytes werden vom niederstwertigen Ende gezählt

1.2 Streifzug durch die Geschichte

- **Übersicht über die geschichtliche Entwicklung mit wichtigsten Meilensteinen**

Bezeichnung	Technik und Anwendung	Zeit
Abakus, Zahlenstäbchen	mechanische Hilfsmittel zum Rechnen	bis ca. 18. Jahrhundert
mechanische Rechenmaschinen	mechanische Apparate zum Rechnen	1623 - ca. 1960
elektronische Rechenanlagen	elektronische Rechenanlagen zum Lösen von numerischen Problemen	seit 1944
Datenverarbeitungs-anlage	Rechner kann Texte und Bilder bearbeiten	seit ca. 1955
Informations-verarbeitungssystem	Rechner lernt, Bilder und Sprache zu erkennen (KI)	seit 1968

- **Fünf Rechnergenerationen im Überblick:**

Generation	Zeitdauer (ca.)	Technologie	Operationen/sec
1	1946 - 1954	Vakuumröhren	40000
2	1955 - 1964	Transistor	200000
3	1965 - 1971	Small und medium scale integration (SSI, MSI)	1000000
4	1972 - 1977	Large scale integration (LSI)	10000000
5	1978 - ????	Very large scale integration (VLSI)	100000000

- **Rechner im elektronischen Zeitalter**
  - 1954: Entwicklung der Programmiersprache Fortran
  - 1955: Erster Transistorrechner
  - 1957: Entwicklung Magnetplattenspeicher, Erste Betriebssysteme für Großrechner
  - 1968: Erster Taschenrechner
  - 1971: Erster Mikroprozessor
  - 1981: Erster IBM PC, Beginn des PC-Zeitalters

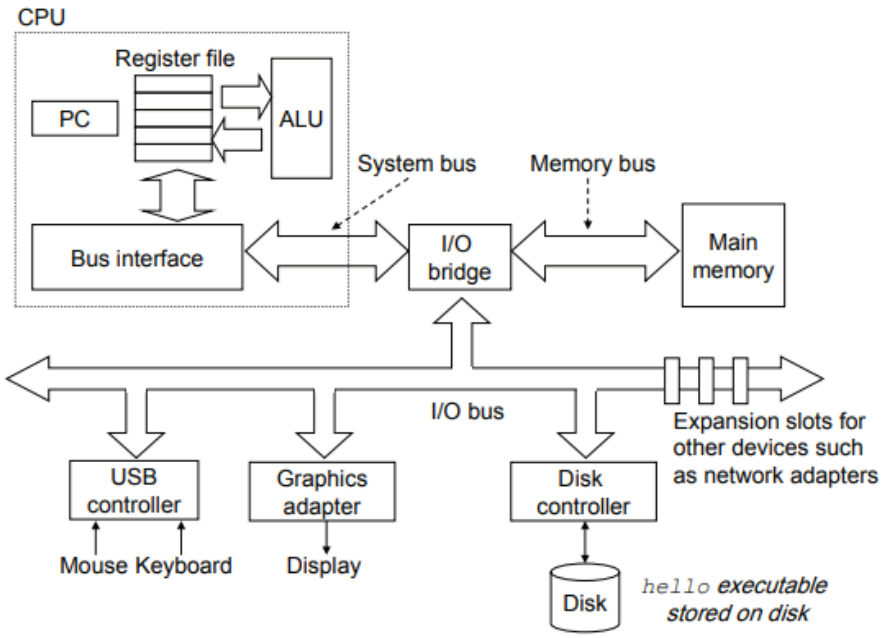
1.3 Ethik in der Informatik

- Ethik in der Informatik
  - Ethik: Bewertung menschlichen Handelns
  - Verbindung zur Informatik: Anwendung von Rechnern für kriegisches Handelns
  - **Dual-Use-Problematik**: Verwendbarkeit von Rechnern für zivile als auch militärische Zwecke
- Digitale Souveränität
  - Souveränität: Fähigkeit zur Selbstbestimmung (Eigenständigkeit, Unabhängigkeit)
  - Digitale Souveränität: Souveränität im digitalen Raum

2 Einführung in die maschinennahe Programmierung

2.1 Begrifflichkeiten und Grundlagen

- **Allgemein**
  - Architektur / Programmiermodell
    - Programmierersicht auf Rechnersystem
    - Definiert durch Maschinenbefehle und Operanden
  - Mikroarchitektur
    - Hardware-Implementierung der Architektur
- **Programmierparadigmen**
  - Synonyme: Denkmuster, Musterbeispiel
  - Bezeichnet in der Informatik ein übergeordnetes Prinzip
  - Dieses Prinzip ist für eine ganze Teildisziplin typisch
  - Manifestiert sich an Beispielen, keine konkrete Formulierung
  - Maschinensprache (Assembler) ist ein primitives Paradigma
- **Programmiermodell**
  - Bei höheren Programmiersprachen:
    - Grundlegende Eigenschaften einer Programmiersprache
  - Bei maschinennaher Programmierung:
    - Bezeichnet dort den **Registersatz** eines Prozessors
    - Registersatz besteht aus:
      - Register, die durch Programme angesprochen werden können
      - Liste aller verfügbaren Befehle (**Befehlssatz**)
    - Register, die prozessorintern verwendet werden (IP/PC) zählen nicht zum Registersatz
      - IC: Instruction Pointer
      - PC: Program Counter
- **Verfeinerung des Rechensystems**



- CPU/Prozessor: führt die im Hauptspeicher abgelegten Befehle aus
- ALU/Arithmethic Logical Unit: Ausführung der Operationen

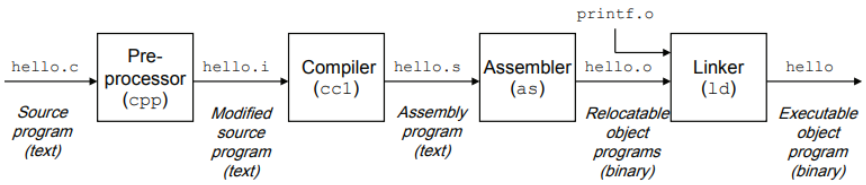
- PC/Program Counter: Verweis auf nächsten Maschinenbefehl im Hauptspeicher
- Register: Schneller Speicher für Operanden
- Hauptspeicher: Speichert Befehle und Daten
- Bus Interface: Verbinden der einzelnen Komponenten

2.2 Nötiges Vorwissen für Assembler

- **Allgemeine Informationen**
  - Programmieren in der Sprache des Computers
    - Maschinenbefehle: Einzelnes Wort
    - Befehlssatz: Gesamtes Vokabular
  - Befehle geben Art der Operation und ihre Operanden an
  - Zwei Darstellungen:
    - Assemblersprache: Für Menschen lesbare Schreibweise für Instruktionen
    - Maschinensprache: maschinenlesbares Format (1 und 0)
- **ARM-Architektur - Hier verwendetes Rechnersystem**
  - z.B. verwendet bei Raspberry Pi
  - ARM: Acorn RISC Machines / Advanced RISC Machines
  - Große Verbreitung heutzutage in Smartphones
- **Phasen der Übersetzung**
  - Beispielhaftes `C`-Programm:

```c
#include <stdio.h> /* Standard Input/Output */ /* Header-Datei*/
int main() {
printf("Hello World\n");
return 0;
}
```

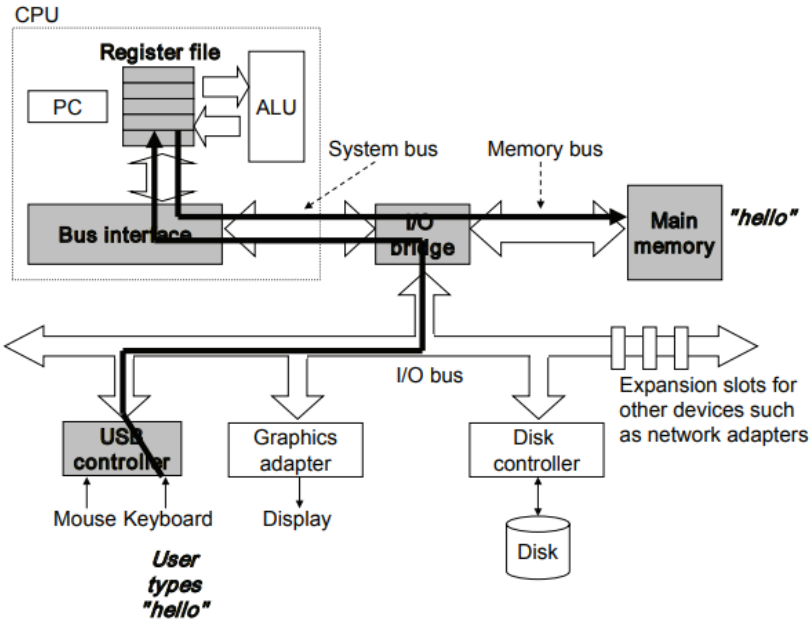
  - `C`-Programm an sich für den Menschen verständlich
  - Übersetzung in Maschinenbefehle für Ausführung auf dem Rechnersystem:



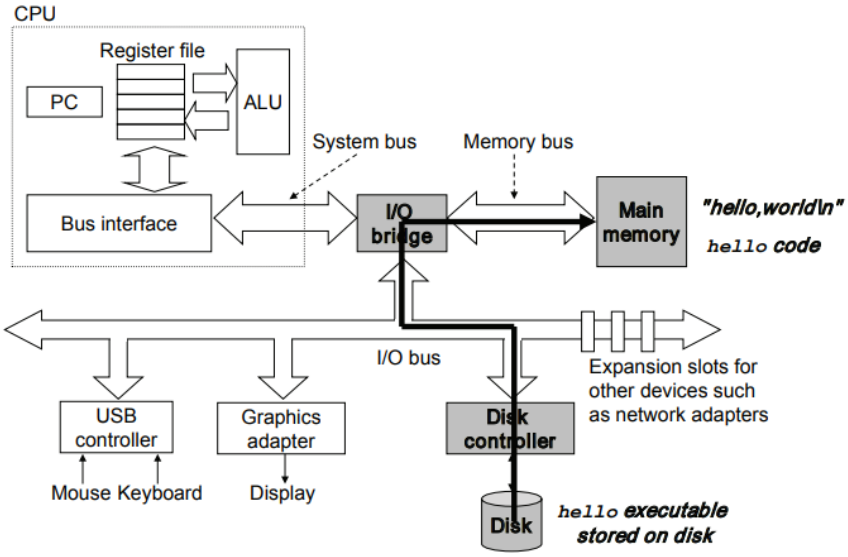
  - 1. Phase (Preprocessor)
    - Aufbereitung durch Ausführung von Direktiven (Code mit #)
    - z.B.: Bearbeiten von `#include <stdio.h>`
      - Lesen des Inhalts der Datei `stdio.h`
      - Kopieren des Inhalts in die Programmdatei
    - Ausgabe: `C`-Programm mit der Endung `.i`
  - 2. Phase (Compiler)
    - Übersetzt `C`-Programm `hello.i` in Assemblerprogramm `hello.s`
  - 3. Phase (Assembler)
    - Übersetzt `hello.s` in Maschinensprache
    - Ergebnis ist das Objekt-Programm `hello.o`
  - 4. Phase (Linker)
    - Zusammenfügen verschiedener Module

        - Code vn `printf` exisitert bereits als `print.o`-Datei
    - Linker kombiniert `hello.o` und `printf.o` zu ausführbarem Programm
    - Ausgabe des Bindevorgangs: ausführbare `hello`-Objektdatei

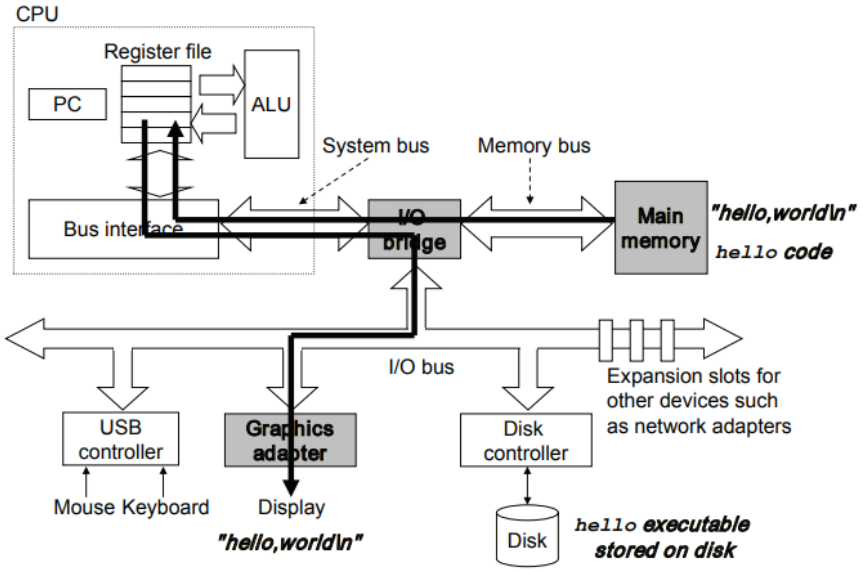
- **Ausführung des Programms**
    - Ausgangspunkt
        - Ausführbares Objektprogramm `hello` auf der Festplatte
        - Starten der Ausführung des Programms unter Nutzung der Shell
    - Ablauf:
        - Shell liegt Zeichen des Kommandos ins Register
        - Speichert den Inhalt dann im Hauptspeicher aber



        - Schrittweises Kopieren der Befehle/Daten von Festplatte in Hauptspeicher



        - Ausführen der Maschinenbefehle des `hello`-Programms



- **Erstes Assembler-Programm**
  - Code

```c
#include <stdio.h>

int main() {
    int p = 5; /* Definition + Variablenzuweisung */
    int q = 12;
    int result = p + q;
    printf("result ist %d \n", result); /* Platzhaltersystem für Strings */
    return 0;
}
```

  - Befehle:
    - `gcc addition.c` übersetzt das C-Programm
    - `gcc -S additionc` generiert das Assemblerprogramm
  - Assemblercode

```
        .file   "addition.c"
        .section        .rodata
        .align  2
.LC0:
        .ascii  "result ist %d \012\000"
        .text
        .align  2
        .global main
        .syntax unified
        .arm
        .fpu vfp
        .type   main, %function
main:
        @ args = 0, pretend = 0, frame = 16
        @ frame_needed = 1, uses_anonymous_args = 0
        push    {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #16
        mov     r3, #5
        str     r3, [fp, #-8]
        mov     r3, #12
        str     r3, [fp, #-12]
        ldr     r2, [fp, #-8]
        ldr     r3, [fp, #-12]
        add     r3, r2, r3
        str     r3, [fp, #-16]
        ldr     r1, [fp, #-16]
        ldr     r0, .L3
        bl      printf
        [...]
```

Abbildung 1: ARM Architektur

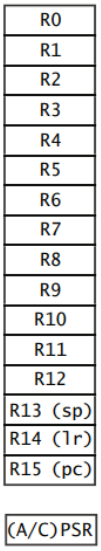
```
        .file   "addition.c"
        .section        .rodata
.LC0:
        .string "result ist %d \n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -12(%rbp)
        movl    $12, -8(%rbp)
        movl    -8(%rbp), %eax
        movl    -12(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        [...]
```

Abbildung 2: Intel Architektur

    - Unterschiedliche Syntax abhängig vom Prozessor (auch Registernamen)
    - Relevanter Code `ARM`:
        - `mov` schiebt Werte in Register (5,12)
        - `add` addiert zwei Zahlen
        - Registernamen r und Zahl (z.B. $r0$)
    - Relevanter Code `Intel`:
        - `addl`: Operation hier nur mit 2 Operanden, rechter Wert ist das Zielregister

- **Befehle eines Rechnersystems**
    - Wieviele Befehle und was für Befehle soll ein Rechnersystem haben?
    - Viele komplexe Befehle:
        - `CISC`-Maschinen (Complex Instruction Set Computer)
        - Befehlsausführung direkt im Speicher möglich
        - Verwendet von Intel-Architektur
    - Weitgehend identische Ausführungszeit der Befehle
        - `RISC`-Maschinen (Reduce Instruction Set Computer)
        - Ermöglicht effizientes Pipeling
        - Werden auch als Load/Store-Architekturen bezeichnet (Nur Ausführung im Register)
        - Verwendet von ARM-Architektur
    - Jedoch viele Befehle, die jeder Prozessor hat (AND, OR, NOT,..)
    - Interner Aufbau eines Rechners hat viele Freiheitsgrade
    - Diese Struktur hat erheblichen Einfluss auf die Leistungsfähigkeit eines Rechnersystems
    - n-Adressmaschinen
        - Einteilung nach der Anzahl der Operanden in einem Maschinenbefehl
        - 2-Adressmaschine (Intel Architektur)
        - 3-Adressmaschine (ARM Architektur)

- Programmiermodell des `ARM`-Prozessors - Registersatz



- R0-R12: Normale Register
- R13-15: Spezialregister
- R15: Program Counter
- (A/C) PSR TODO