Rechnerorganisation
Sommersemester 2023 – 2. Vorlesung

Prof. Stefan Roth, Ph.D.

Technische Universität Darmstadt

24. April 2023

Inhalt

Team & Organisation

Team – Dozent

Prof. Stefan Roth, Ph.D.

- seit 2007 am FB Informatik, 2007–2013 Juniorprofessor, seit 2013 Professor
- Leiter Fachgebiet Visuelle Inferenz
- Forschungsgebiete: Computer Vision, Maschinelles Lernen
- Weitere Arbeitsgebiete: Sprecher Forschungsfeld Information and Intelligence (I+I), Direktor Konrad Zuse School of Excellence in Learning and Intelligent Systems (ELIZA), Direktor ELLIS Unit Darmstadt, ...
- Weitere Interessen: Bergsteigen & Wandern, Foto- & Videografie, ...
- Sprechstunde: siehe Webseite
- Raum: S2 | 02, A304

Team – Tutor:innen & Wissenschaftlicher Mitarbeiter

Das Team

- Tutor:innen: Adrian Schmidt, Berenike Peter, Daniel Simon, Florian Piana, Jan Kai Braun, Julian Rupprecht, Konstantin Aurel Meudt, Leena Jamil, Mehmet Bulut, Ömer Yilmaz, Senad-Leandro Lemes Galera, Tim Carlo Päpke, Yu Xiao
- Praktikum in der Lehre: Simon Cramm, Alexander Marc Anastasius Gallus, Eray Dogan, Minh Huy Tran, Josef Samir
- WiMi: Robin Hesse

Organisation I

Zentrales Moodle

- Material (Vorlesungsfolien, Übungsblätter, Aufzeichnungen, ergänzende Informationen) gibt es im Moodle `https://moodle.tu-darmstadt.de/course/view.php?id=32097`
- Kein Einschreibeschlüssel erforderlich, Zugang via Anmeldung zur **Lehrveranstaltung** in TUCaN

Anmeldung

- Bitte für Lehrveranstaltung in TUCaN anmelden (falls noch nicht geschehen).
- Anmeldung zur Studienleistung in TUCaN nicht vergessen!
- Anmeldung zu den Übungsgruppen in Moodle verlängert bis **Dienstag, 25.04.2023**.

Organisation II

- Übungen ab heute
- Zum 1. Übungsblatt
  - keine Rechnungen mit Papier und Bleistift in VZB, 2K, IEEE 754
  - aber Binärcodes/Dualcodes, Hexadezimaldarstellung, ... muss man kennen
- Aufbau der Programmierumgebung (Handwerkszeug)
- Zum 2. Übungsblatt
  - erste Assemblerprogramme

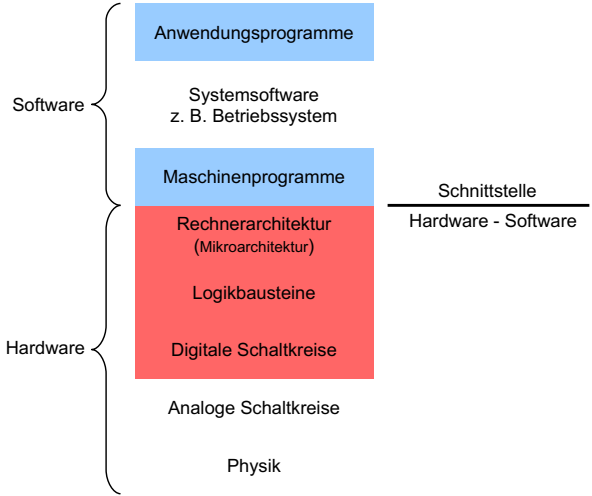
Organisation III

Credits

- Materialien sind über Jahre gewachsen und haben sich bewährt.
- Erstellt insbesondere von **Dr. Wolfgang Heenes** sowie einer Reihe engagierter **Tutor:innen**.

Struktur eines Rechnersystems

Schichtenmodell eines Computers I



Komponenten und Struktur eines Rechnersystems I

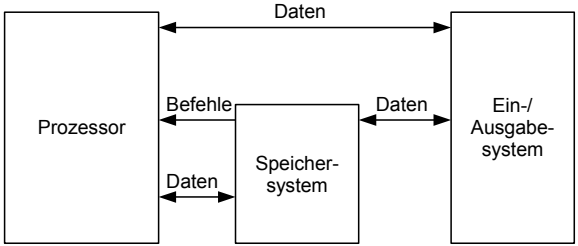


Abbildung: Komponenten eines Rechnersystems (abstrakte Darstellung)

Anmerkung: Die Komponenten eines Rechnersystems müssen in der Regel eine gemeinsame Zeitbasis haben, damit das Zusammenspiel funktioniert. Diese Zeitbasis nennt man auch Takt/Taktsignal/Systemtakt.

Komponenten und Struktur eines Rechnersystems II

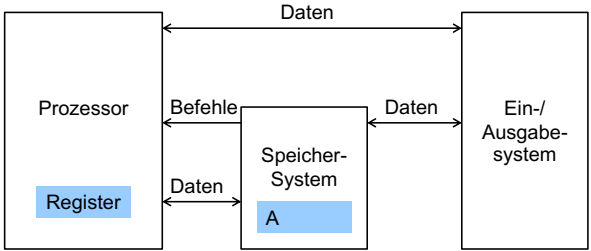


Abbildung: Komponenten eines Rechnersystems (verfeinerte Darstellung)

Speicherhierarchie I – Klassifikation von Speichern

- Explizite und transparente[1] Nutzung
  - Speicher können zunächst danach klassifiziert werden, ob sie für der/die Programmierer:in (bzw. das später auszuführende Maschinenprogramm) explizit zugreifbar sind oder nur implizit, d. h. für das Maschinenprogramm transparent, verwendet werden.
- explizite Nutzung
  - interner Prozessorspeicher
  - Hauptspeicher
  - Sekundärspeicher
- transparente Nutzung
  - bestimmte Register auf dem Prozessor
  - Cache-Speicher

[1] Unter Transparenz versteht man im Zusammenhang mit der Computer- und Netzwerktechnik, dass ein bestimmter Teil eines Systems zwar vorhanden und in Betrieb, aber ansonsten „unsichtbar“ ist und daher vom/von der Benutzer:in nicht als vorhanden wahrgenommen wird.

Speicherhierarchie II – Explizit nutzbare Speicher

- interner Prozessorspeicher/Register
  - schnelle Register zur temporären Speicherung von Maschinenbefehlen und Daten
  - direkter Zugriff durch Maschinenbefehle
  - Technologie: Halbleiter ICs[2]
- Hauptspeicher
  - relativ großer und schneller Speicher für Programme und Daten während der Ausführung
  - direkter Zugriff durch Maschinenbefehle
  - Technologie: Halbleiter ICs
- Sekundärspeicher
  - sehr großer, aber langsamer(er) Speicher für die permanente Speicherung von Programmen und Daten
  - indirekter Zugriff über Ein-/Ausgabe-Programme, die Daten in den Hauptspeicher transferieren
  - Technologie: Halbleiter ICs, Magnetplatten, optische Laufwerke, Magnetbänder

[2] Integrated Circuit, integrierter Schaltkreis

Speicherhierarchie III – Übersicht

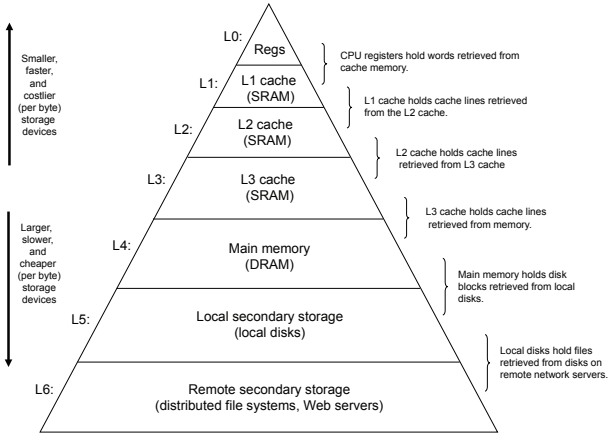


Abbildung: Quelle: [BO10, S. 48]

Einführung in die maschinennahe Programmierung

Einführung in die maschinennahe Programmierung

- Einleitung
- Assemblersprache
- Maschinensprache
- Programmierung
- Adressierungsmodi
- Compilieren, Assemblieren und Linken
- ...

Einleitung

- Architektur/Programmiermodell (vgl. [HH16, S. 295 ff.])
  - Programmierersicht auf Computer
  - Definiert durch Maschinenbefehle[3] und Operanden
- Mikroarchitektur (vgl. [HH16, S. 385 ff.])
  - Hardware-Implementierung der Architektur

[3] Instruktionen, Operationen

Programmierparadigmen

- Der Begriff des **Paradigma**
- Synonyme Begriffe sind: Denkmuster, Musterbeispiel
- Verwendung des Paradigma-Begriffes in der Informatik
  - ein Paradigma bezeichnet ein übergeordnetes Prinzip
  - dieses Prinzip ist für eine ganze Teildisziplin typisch
  - es ist jedoch nicht klar ausformulierbar, sondern manifestiert sich in Beispielen
- Maschinensprache (bzw. Assembler) ist ein primitives Paradigma

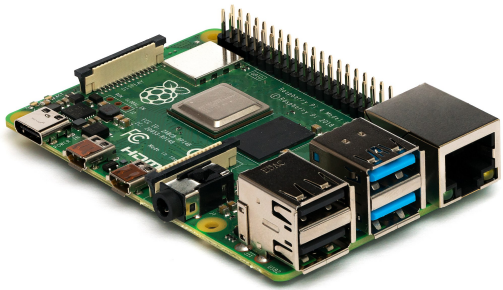
Programmiermodell

- Programmiermodell ist ein Begriff, der häufig in unterschiedlichen Definitionen verwendet wird.
- Bei höheren Programmiersprachen
  - Grundlegende Eigenschaften einer Programmiersprache (Programmiermodell dieser Hochsprache)
  - Unterschied: Programmiermodell und Programmierparadigma?
  - Und es gibt noch viel andere Begriffe: Verarbeitungsmodell, Modellierungsmodell ...
- In der maschinennahen Programmierung bezeichnet das Programmiermodell $\Rightarrow$ Registersatz eines Prozessors sowie die verfügbaren Befehle (Befehlssatz)
- Register die prozessorintern verwendet werden (z. B. das Statusregister) zählen nicht zum nutzbaren Registersatz des Prozessors

Assemblersprache

- Programmieren in der Sprache des Computers
  - Maschinenbefehle: Einzelne Worte
  - Befehlssatz: Gesamtes Vokabular
- Befehle geben Art der **Operation** und ihre **Operanden** an
- Zwei Darstellungen
  - Assemblersprache: für Menschen lesbare Schreibweise für Instruktionen
  - Maschinensprache: maschinenlesbares Format (1en und 0en)

Verwendetes Rechnersystem I – ARM®-Architektur



Raspberry Pi 4 Model B, mit ARM® Cortex-A72
Bild: Michael H. („Laserlicht“) / Wikimedia Commons

Verwendetes Rechnersystem II – ARM®-Architektur

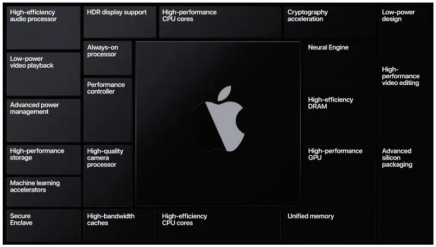
- Die ARM®-Architektur wurde 1983 von Acorn entwickelt. ARM® steht für Acorn RISC Machines/ Advanced RISC Machines.
- Der Befehlssatz wurde von Sophie Wilson entwickelt.
- Die Mikroarchitektur wurde Steve Furber entwickelt.
- 1987 Acorn Archimedes, Home-Computer
- Heute: Große Verbreitung der ARM®-Architektur in Smartphones, Laptops, Supercomputern

Rechnersysteme aus der Praxis

Apple wechselt in seinen Macs von Intel- auf ARM-Prozessoren

Auf der Keynote zu der Entwicklerkonferenz WWDC hat Apple den Übergang von Intel- auf ARM-Prozessoren bekanntgegeben.

Lesezeit: 2 Min. In Pocket speichern 1433



22.06.2020 21:01 Uhr | Mac & i

Von Johannes Schuster

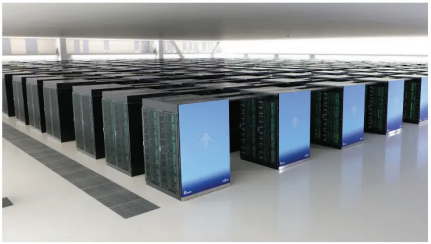
Apple hat auf seiner in diesem Jahr nur online gestreamten Entwicklerkonferenz WWDC den Wechsel von Intel- auf ARM-Prozessoren bei seinen Macs angekündigt. Laut Tim Cook sollen innerhalb von zwei Jahren alle Mac-Modelle auf die ARM-Architektur umgestellt werden. Es soll aber auch weiterhin sowohl Macs mit Intel-Prozessor als auch Software dafür geben. Der erste Serien-Mac mit ARM-

Rechnersysteme aus der Praxis

Top500-Supercomputer: Japan überrundet alten Spitzenreiter mit 415 PFlops

Der erste Supercomputer, den man je nach Gesichtspunkt bereits als Exascale-System bezeichnen kann, kommt nicht aus China oder den USA, sondern aus Japan.

Lesezeit: 3 Min. In Pocket speichern 33



(Bild: Riken Center for Computational Science)

22.06.2020 17:00 Uhr

Von Andreas Stiller

Mit 415 PFlops im Linpack-Benchmark setzt sich der japanische Supercomputer Fugaku mit ARM-Prozessoren von Fujitsu ganz klar an die Spitze der neuen 55. Top500-Liste der Supercomputer, die heute zum Auftakt der online abgehaltenen ISC 2020 Digital vorgestellt wurde.

Rechnersysteme aus der Praxis

Fugaku: Weltweit schnellster CPU-only-Supercomputer mit 7,7 Millionen Kernen

Anstelle von DDR4-RAM kommen 5 Petabyte High-Bandwidth Memory (HBM2) mit einer gemeinsamen Übertragungsrate von 163 PByte/s zum Einsatz.

Lesezeit: 1 Min. In Pocket speichern 16

(Bild: Riken Center for Computational Science)

15.05.2020 16:41 Uhr

Von Mark Mantel

Sechs Jahre nach dem Startschuss zur Entwicklung eines neuen japanischen Supercomputers ist Fugaku fertiggestellt. Früher unter dem Namen „Post-K" designt, nutzt Fugaku 158.976 A64FX-Prozessoren mit jeweils 48 ARM-CPU-Kernen, die Fujitsu für das Projekt entwickelt hat. Die knapp 7,7 Millionen Rechenkerne enthalten jeweils zwei Scalable Vector Extensions (SVE), die ähnlich Intels AVX-512-Instruktionen 512-Bit-Werte verarbeiten können.

Verfeinerung des Rechnersystem

- Ein modernes Rechnersystem hat folgende Struktur

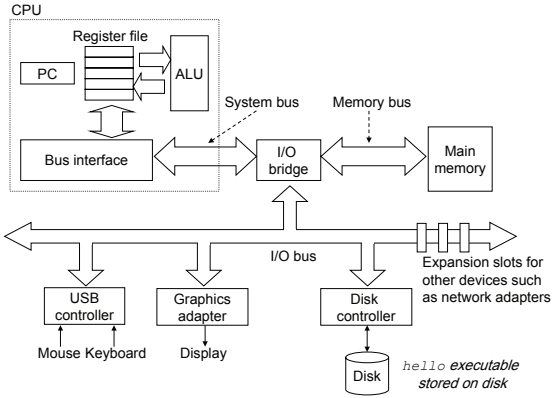


Abbildung: Quelle: [BO10, S. 42]

Rechnersystem Komponenten

- CPU/Prozessor: führt die im Hauptspeicher abgelegten Befehle aus.
- ALU[4]:
  - Ausführung der Operationen
  - Typische Befehle sind: `add`, `sub`, `mul` und `div`
- PC[5]: Programmzähler, der immer auf den nächsten Maschinenbefehl im Hauptspeicher (Main memory) zeigt.
- Register (Register file): schneller Speicher für Operanden.
- Hauptspeicher (Main memory): Speichert Befehle und Daten
- Bus Interface: Verbinden der einzelnen Komponenten (z. B. System bus, Memory bus)

[4] Arithmetic Logic Unit
[5] Program Counter

Beispielprogramm in C

- Wie wird auf dem Rechnersystem ein Programm ausgeführt?
- Beispiel: hello world (Datei `hello.c`)

```
1 #include <stdio.h>
2 int main()
3 {
4   printf("Hello World\n");
5   return 0;
6 }
```

Phasen der Übersetzung

- Das C Programm ist für den Menschen verständlich.
- Zur Ausführung auf dem Rechnersystem muss es in Maschinenbefehle übersetzt werden.
- Beispiel: Unix System: `gcc -o hello hello.c`

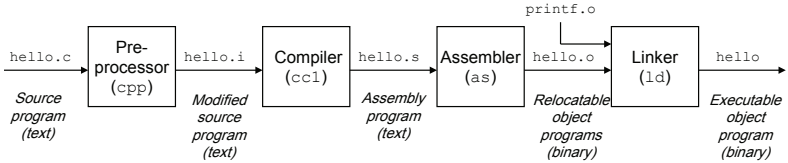


Abbildung: Quelle: [BO10, S. 39]

Phasen der Übersetzung

- Übersetzungsvorgang ist in verschiedene Phasen unterteilt
- 1. Phase (Preprocessor)
  - Aufbereitung durch Ausführung von Direktiven (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 das C-Programm `hello.i`
  - in ein Assemblerprogramm `hello.s`
- 3. Phase (Assembler)
  - Übersetzt `hello.s` in Maschinensprache
  - Ergebnis ist das Objekt-Programm `hello.o`

Phasen der Übersetzung

- 4. Phase (Linker)
  - Zusammenfügen verschiedener Module
    - Beispielprogramm nutzt die `printf` Funktion
    - Der Code von `printf` existiert bereits übersetzt in einer Bibliothek (der Standard C Library) als `printf.o` Datei
  - Der Linker kombiniert `hello.o` und `printf.o` zu einem ausführbaren Programm (u. a. Auflösen von Referenzen)
  - Ausgabe des Bindevorgangs: `hello` Datei
  - `hello` ist eine ausführbare Objekt-Datei, die in den Speicher geladen und ausgeführt werden kann

Wie wird das Programm `hello` ausgeführt?

- Wie wird das Programm `hello` ausgeführt?
- Ausgangspunkt:
  - ausführbares Objektprogramm `hello` auf der Festplatte
  - Starten der Ausführung des Programms unter Nutzung eines speziellen Programms, der Shell
    `./hello`
  - die Shell ist ein Kommandozeileninterpeter
    - sie druckt Eingabeaufforderung (Prompt)
    - wartet auf Eingabe einer Kommandozeile
    - führt Kommando aus (bzw. initiiert die Ausführung)

Schritte zur Ausführung von `hello`

- Shell liest zunächst die Zeichen des Kommandos in die Register und speichert den Inhalt dann im Hauptspeicher ab.

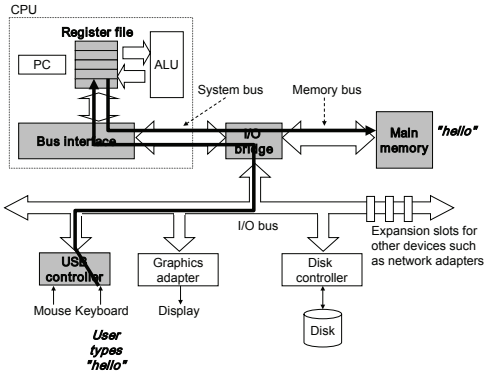


Abbildung: Quelle: [BO10, S. 45]

Schritte zur Ausführung von `hello`

- Schrittweises Kopieren der Befehle und Daten von Festplatte in den Hauptspeicher (hier wird Direct Memory Access (DMA) benutzt).

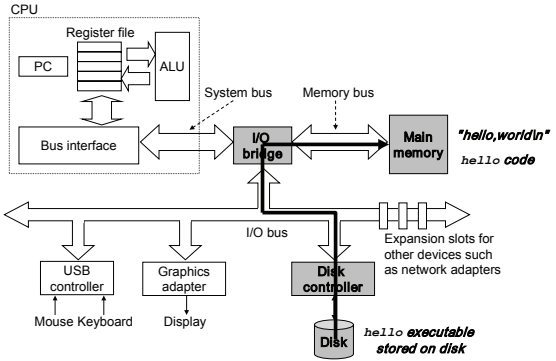


Abbildung: Quelle: [BO10, S. 45]

Schritte zur Ausführung von `hello`

- Ausführen der Maschinenbefehle des hello Programms

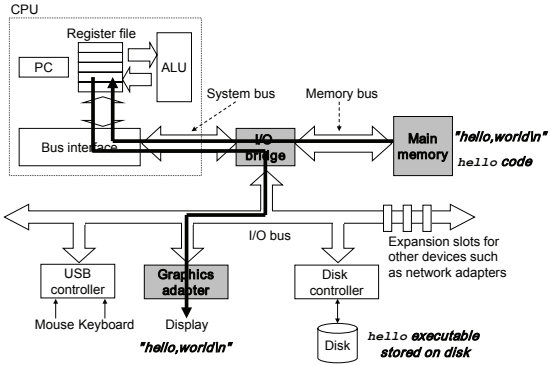


Abbildung: Quelle: [BO10, S. 46]

- Learning Nugget 01 - Video zur Ausführung von `hello`

Ein erstes Assemblerprogramm – noch in C

```
1  /* Addition */
2  #include <stdio.h>
3
4  int main()
5  {
6    int p = 5;
7    int q = 12;
8    int result = p + q;
9    printf("result ist %d \n",result);
10   return 0;
11 }
```

- `gcc addition.c` übersetzt das C-Programm
- `gcc -S addition.c` generiert das Assemblerprogramm

Ein erstes Assemblerprogramm II – addition.s (ARM® Architektur)

```
1           .file   "addition.c"
2           .section        .rodata
3           .align  2
4   .LC0:
5           .ascii  "result_ist_%d_\012\000"
6           .text
7           .align  2
8           .global main
9           .syntax unified
10          .arm
11          .fpu vfp
12          .type   main, %function
13  main:
14          @ args = 0, pretend = 0, frame = 16
15          @ frame_needed = 1, uses_anonymous_args = 0
16          push    {fp, lr}
17          add     fp, sp, #4
18          sub     sp, sp, #16
19          mov     r3, #5
20          str     r3, [fp, #-8]
21          mov     r3, #12
22          str     r3, [fp, #-12]
23          ldr     r2, [fp, #-8]
24          ldr     r3, [fp, #-12]
25          add     r3, r2, r3
26          str     r3, [fp, #-16]
27          ldr     r1, [fp, #-16]
28          ldr     r0, .L3
29          bl      printf
30          [...]
```

Ein zweites Assemblerprogramm III – addition.s (Intel Architektur)

```
1           .file   "addition.c"
2           .section        .rodata
3   .LC0:
4           .string "result_ist_%d_\n"
5           .text
6           .globl  main
7           .type   main, @function
8   main:
9   .LFB0:
10          .cfi_startproc
11          pushq   %rbp
12          .cfi_def_cfa_offset 16
13          .cfi_offset 6, -16
14          movq    %rsp, %rbp
15          .cfi_def_cfa_register 6
16          subq    $16, %rsp
17          movl    $5, -12(%rbp)
18          movl    $12, -8(%rbp)
19          movl    -8(%rbp), %eax
20          movl    -12(%rbp), %edx
21          addl    %edx, %eax
22          movl    %eax, -4(%rbp)
23          movl    -4(%rbp), %eax
24          movl    %eax, %esi
25          movl    $.LC0, %edi
26          movl    $0, %eax
27          call    printf
28          [...]
```

Befehle eines Rechnersystems I – CISC, RISC

- Was für Maschinenbefehle soll ein Prozessor eigentlich haben?
  - ergibt sich aus den Aufgaben, die ein Prozessor hat
  - z. B. `add` (für Addition), `sub`, `mul`, ...
- Wieviele Befehle soll ein Prozessor haben?
  - soviel wie notwendig
  - Eine Multiplikation kann auf eine Folge von Additionen zurückgeführt werden. Ist ein Befehl für Multiplikation dann notwendig?
- Es gibt zwei Ansätze bei der „Implementierung“ eines Maschinenbefehls in einem Prozessor
  - Einführung von (komplexen) Befehlen, die eine Funktion (z. B. die Multiplikation mit den Quelloperanden Register und Speicher) vollständig ausführen.
  - Einführung von (reduzierten) Befehlen, die eine Funktion als Folge von mehreren Befehlen nacheinander ausführen.

Befehle eines Rechnersystems II – CISC, RISC

- Rechnersysteme, die komplexe Befehle haben, werden als CISC[6]-Maschinen bezeichnet (z. B. Intel Architektur). Bei diesen CISC-Maschinen ist es möglich, in den Maschinenbefehlen als Operanden sowohl Register, als auch (Haupt-)Speicheradressen anzugeben.
- Rechnersysteme, die reduzierte Befehle haben, werden als RISC[7]-Maschinen bezeichnet (z. B. ARM® Architektur). Eine Eigenschaft der RISC-Maschinen ist, dass Befehle eine weitgehend identische Ausführungszeit besitzen. Dies ermöglicht effizientes Pipelining. Weil RISC-Maschinen nur auf den Registern umformende Operationen (z. B. add) ausführen können, werden sie auch als Load/Store-Architekturen bezeichnet. Das bedeutet, ein Datum muss aus dem (Haupt-)Speicher in ein Register geladen werden.

[6] Complex Instruction Set Computer
[7] Reduce Instruction Set Computer

Befehle eines Rechnersystems III – CISC, RISC

- Rechnersysteme haben ähnliche Grundstrukturen
  - Ein **Prozessor** (Zentraleinheit, CPU), der Programme ausführen kann.
  - Ein **Speicher** der Programme und Daten enthält (Speichersystem).
  - Eine Möglichkeit, zum Transferieren von Informationen zwischen dem Speicher und dem Prozessor, sowie der Außenwelt (Ein-/Ausgabesystem).
- Der interne Aufbau (Struktur) eines Rechnersystems hat viele Freiheitsgrade.
- Auch die Struktur eines Prozessores hat erheblichen Einfluss auf die Leistungsfähigkeit (und die Kosten) eines Rechnersystems
- Eine Einteilung kann z. B. nach der Anzahl der Operanden in einem Maschinenbefehl vorgenommen werden.
- Man spricht dann auch von *n*-Adressmaschinen
  - 2-Adressmaschine (Intel Architektur, teilweise aber auch drei Adressen)
  - 3-Adressmaschine (ARM® Architektur)

Programmiermodell des ARM®-Prozessors – Registersatz

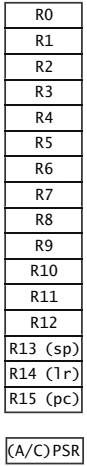
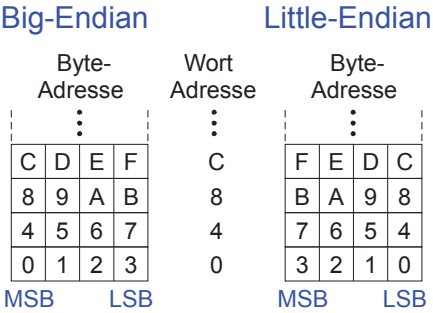


Abbildung: Quelle: DEN0024A_v8_architecture_PG. S. 4-14

Speicherorganisation: Big-Endian und Little-Endian

- Schemata für Nummerierung von Bytes in einem Wort (Wort-Adresse ist bei beiden gleich)
- Big-Endian: das höchstwertige Byte wird zuerst gespeichert, d. h. an der kleinsten Speicheradresse (Motorola-Format)
- Little-Endian: das kleinstwertige Byte wird an der Anfangsadresse gespeichert (Intel-Format)



Zusammenfassung und Ausblick

Zusammenfassung und Ausblick

Zusammenfassung

- Einführung in die maschinennahe Programmierung

Ausblick

- Konzepte der maschinennahen Programmierung

Lernkontrolle

- Ich habe die Verfeinerung des abstrakten Rechnersystems nachvollziehen können und die Idee der sog. Speicherpyramide nachvollziehen können ☑
- Die Begriffe *Programmiermodell* und *Maschinenbefehle* sind mir geläufig und ich kann die „Maschinensprache“ gegenüber Hochsprachen abgrenzen ☑
- Ich kann die Schritte der Programmübersetzung nachvollziehen und habe den Kern/die Funktion des ersten Assemblerprogramms verstanden ☑
- Die Idee zur Unterscheidung von CISC- und RISC-Architekturen habe ich nachvollziehen können ☑
- ...

Literatur

Literatur

[BO10] Bryant, Randal E. und David R. O'Hallaron: *Computer Systems – A Programmer's Perspective*.
Prentice Hall, 2010.

[HH16] Harris, David Money und Sarah L. Harris: *Digital Design and Computer Architecture, ARM® Edition*.
Morgan Kaufmann, 2016.