Rechnerorganisation

Jonas Milkovits

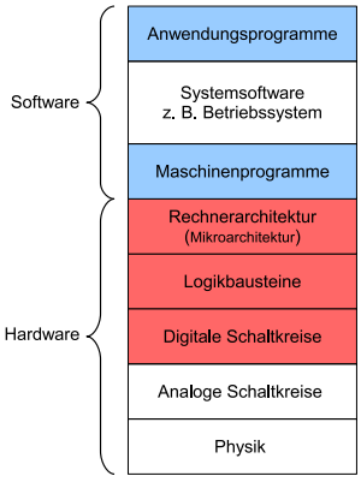
Last Edited: 8. Mai 2020

Inhaltsverzeichnis

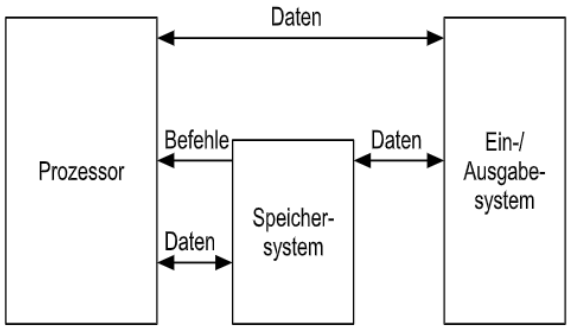
1 Einführung

1.1 Begrifflichkeiten und Grundlagen

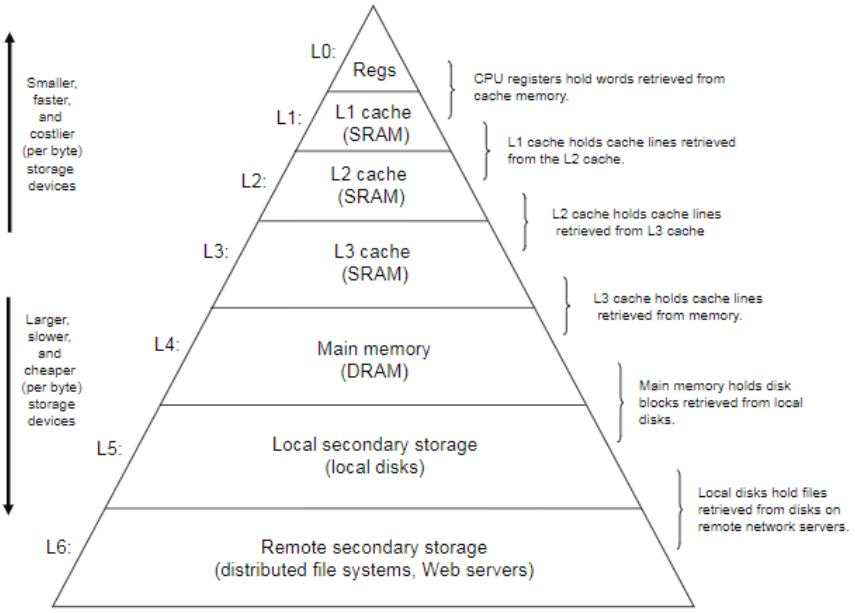
- **Abstraktion**
  - Wichtiges und zentrales Konzept der Informatik
  - Verstecken unnötiger Details (für spezielle Aufgabe unnötig)
- **Schichtenmodell**



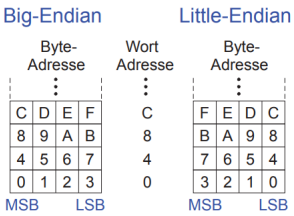
  - Untere Schicht erbringt Dienstleistungen für höhere Schicht
  - Obere Schicht nutzt Dienste der niedrigeren Schicht
  - Eindeutige Schnittstellen zwischen den Schichten
  - Vorteile:
    - Austauschbarkeit einzelner Schichten
    - Nur Kenntnis der bearbeitenden Schicht notwendig
  - Nachteile:
    - 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

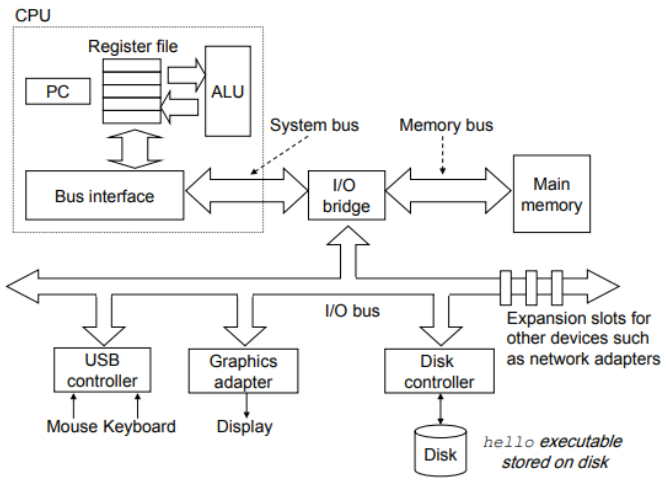
1.3 Ethik in der Informatik

- 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

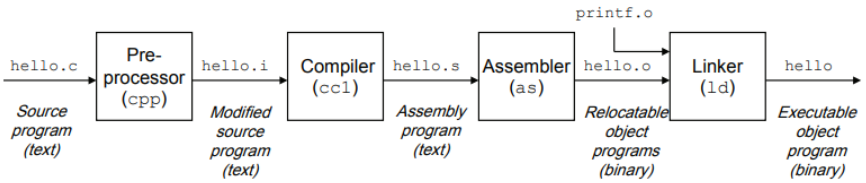
- **Assembler**
  - Programmieren in der Sprache des Computers
    - Maschinenbefehle: Einzelnes Wort
    - Befehlssatz: Gesamtes Vokabular
  - Befehle geben Art der Operation und ihre Operanden an
  - Zwei Darstellungen:
    - Assemblersprache: Für Menschen lesbare Schreibweise für Instruktionen
    - 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

2.2 Phasen der Übersetzung

- Beispielhaftes C-Programm:

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

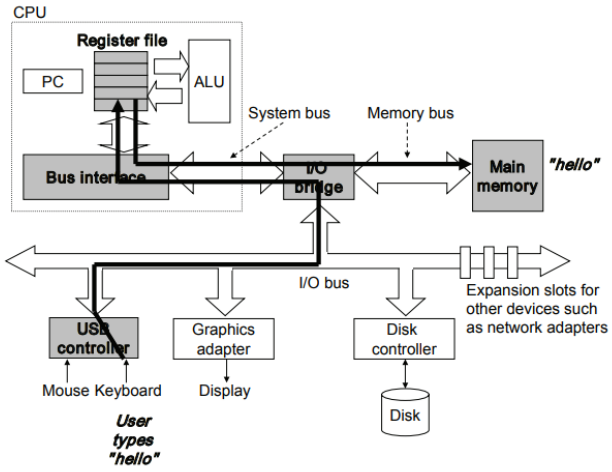
- C-Programm an sich für den Menschen verständlich
- Übersetzung in Maschinenbefehle für Ausführung auf dem Rechnersystem:



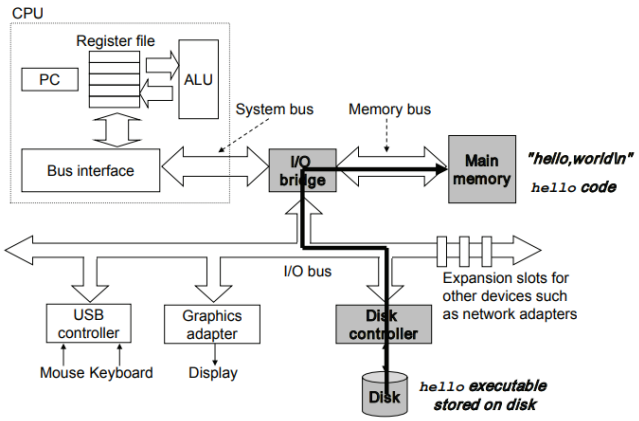
- 1. Phase **(Preprocessor)**
  - Aufbereitung durch Ausführung von Direktiven (Code mit #)
  - z.B.: Bearbeiten von `#include <stdio.h>`
    - Lesen des Inhalts der Datei `stdio.h`
    - Kopieren des Inhalts in die Programmdatei
  - Ausgabe: C-Programm mit der Endung `.i`
- 2. Phase **(Compiler)**
  - 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 von `printf` exisitert bereits als `print.o`-Datei
  - Linker kombiniert `hello.o` und `printf.o` zu ausführbarem Programm
  - Ausgabe des Bindevorgangs: ausführbare `hello`-Objektdatei

2.3 Ausführung eines Programms im Rechnersystem

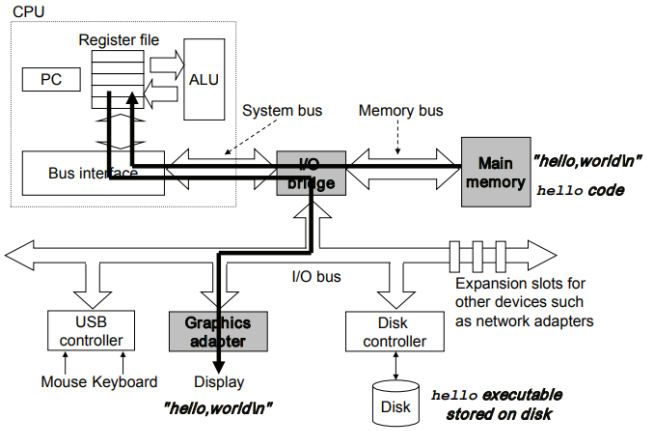
- Ausgangspunkt
  - Ausführbares Objektprogramm `hello` auf der Festplatte
  - Starten der Ausführung des Programms unter Nutzung der Shell
- Ablauf:
  - Shell liegt Zeichen des Kommandos ins Register
  - Speichert den Inhalt dann im Hauptspeicher aber



  - Schrittweises Kopieren der Befehle/Daten von Festplatte in Hauptspeicher



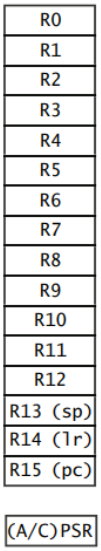
  - Ausführen der Maschinenbefehle des `hello`-Programms



2.4 Befehle eines Rechnersystems

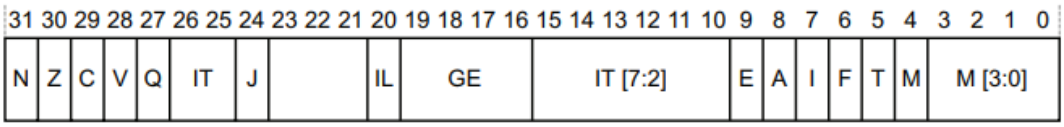
- Wieviele Befehle und was für Befehle soll ein Rechnersystem haben?
- Viele komplexe Befehle:
  - **CISC-Maschinen** (Complex Instruction Set Computer)
  - Befehlsausführung direkt im Speicher möglich
  - Verwendet von Intel-Architektur
- Weitgehend identische Ausführungszeit der Befehle
  - **RISC-Maschinen** (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,..)
- Unterschiedliche Befehlsformate:
  - Erlauben Flexibilität
  - z.B. `add` und `sub` mit drei Registern als Operanden
  - z.B. `ldr` und `str` verwenden zwei Register und Konstante
  - Anzahl an Formaten sollte jedoch klein sein
    - Hardware weniger aufwendig
    - Erlaubt evtl. höhere Verarbeitungsgeschwindigkeit
- 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)
- **Konstanten in Befehlen (intermediates)**
  - Direkt im Befehl untergebebracht $\rightarrow$ Direktwerte
  - Benötigen kein eigenes Register oder Speicherzugriff
  - Direktwert ist Zweierkomplementzahl, die 12 Bit breit ist
  - Bitbreite der Direktwertzahl vom Befehl abhängig
    - Befehle haben immer 32 Bit
    - Registeradressen werden mit 4 Bit kodiert
    - Übrigbleibende Bits für Direktwert

2.5 Registersatz



- R0: Verwendet für Rückgabe von Werten an die `Shell`
- R1-R12: General Purpose Register
- R13: Stack Pointer (sp)
- R14: Link Register (lr)
- R15: Program Counter (pc)
- Current Processor Status Register (CPSR)

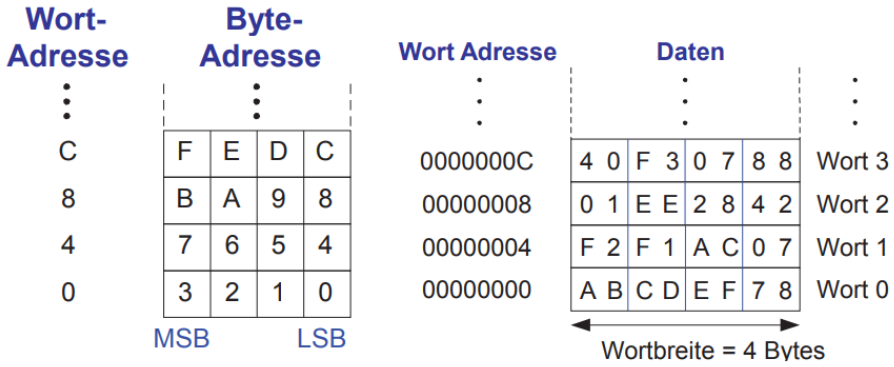
- **Current Processor Status Register**
  - Enthält unter anderem die `Statusflags`



  - Werden oft für Vergleiche (`b,beq,...`) verwendet
  - `N` (Negative): Wird verwendet um zu zeigen, dass Ergebnis negativ ist
  - `Z` (Zero): Wird verwendet um zu zeigen, dass Ergebnis 0 ist
  - `C` (Carry): Zeigt, dass Carry-Bit besteht
  - `V` (OverFlow): Zeigt, dass Overflow geschehen ist
  - Namen können je nach Prozessor stark variieren

2.6 Adressierung des Speichers, Lesen und Schreiben auf Speicher

- **Allgemeine Verwendung von Registerspeicher**
  - Meist zuviele Daten für die Register
  - Kombination des Registers und Hauptspeichers zum Halten von Daten
  - Speichern von häufig verwendeten Daten in Registern (Schleifenvariable)
- **Wort- und Byte-Adressierung von Daten im Speicher**
  - Byte-adressiert: (ARM)
    - Jedes Byte hat eine eindeutige Adresse (Zugriff auf jedes Byte möglich)
    - Ein Wort (hier 32Bit) besteht aus 4 Bytes (32 Bits)
    - Wortbreite ist von der Architektur abhängig
    - Wortadressen sind immer Vielfache von 4 (Offset von 4)



- Rechts wird ein Byte mit zwei Hexawerten dargestellt (AB : 1011 1010)

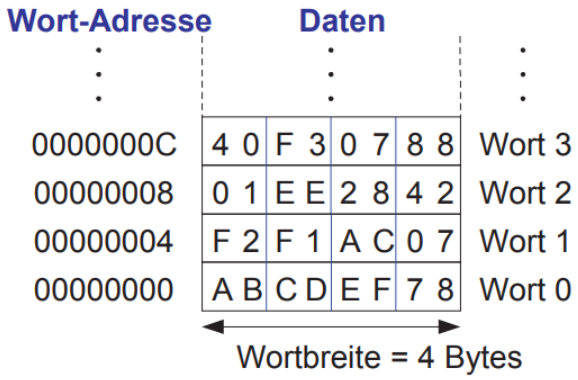
- **Lesen aus byte-adressiertem Speicher**
  - Lesen geschieht durch Ladebefehle (Transportbefehl)
  - Befehlsname: `load word (ldr)`
  - Alternative für Bytes statt Wörtern: `ldrb`
  - Adressarithmethik:
    - Adressen werden relativ zu Register angegeben
    - Basisadresse (startet bei Wort 9) plus Distanz (offset)
    - Adresse = `(r5 (Basis) + 8 (offset))`
  - Beispiel 1:
    - Lese Datenwort von Speicheradresse `(r5+8)` und schreibe es in Register `r7`

      ```
      mov r5,#0 /* Transportbefehl, schreibt Konstante 0 in r5 */
      ldr r7, [r5,#8] /* r7: Zielregister | [r5,#8] Quelle */
      ```

    - `r7` enthält das Datenwort der Speicheradresse `r5+8`
  - Beispiel 2:
    - Lesen Datenwort 3 (Speicheradresse `0xC` (12er Offset)) nach `r7`
    - (Einschub: `0x` sagt dem Compiler, dass das Folgende eine Hexzahl ist)

      ```
      mov r5,#0 /* Schreibt Konstante 0 in r5 */
      ldr r7, [r5, #0xC] /* Lädt den Wert (r5+12) in r7 */
      ```

    - Nach Abarbeiten des Befehls hat `r7` den Wert `0x40F30788`



- **Schreiben in byte-adressierten Speicher**
  - Schreiben geschieht durch Speicherbefehle (Transportbefehl)
  - Befehlsname: `store word (str)`
  - Alternative für Bytes statt Wörtern: `strb`
  - Beispiel:
    - Schreibe den Wert aus `r9` in Speicherwort 5

```
mov r1,#0 /* Speichert Konstante 0 in r1 */
mov r9,#42 /* Speichert Konstante 42 in r9 */
str r9, [r1,#0x14] /* Schreibt Wert des 5. Wortes von r1 in r9 */
```

- `#0x14`: $14_{16} = 0001\ 0100_2 = 20_{10}$ (5.tes Wort)

2.7 Kontrollstrukturen in Assembler

- **Statusbits**
  - Die Wichtigsten:
    - `CF (CarryFlag)`
    - `ZF (ZeroFlag)`
    - `SF (SignFlag)`
    - `OF (OverflowFlag)`
  - Verwendung:
    - Vergleiche (`cmp`)
    - Gleichheit
  - Unterschiede zwischen `Carry` und `Overflow`
    - `Overflow`: Ergebnis passt nicht in maximale darstellbare Werte (z.B. +8 bei 4 Bit im ZK)
    - `Carry`: Ergebnis passt nicht in Bitbreite (+5 - 1 = +4)
    - `Sign`: Vorzeichen negativ
- **Sprünge / Verzweigungen**
  - Änderung der Ausführungsreihenfolge von Befehlen
  - Unbedingte Sprünge
    - Werden immer ausgeführt
    - `b target /* Springt von branch zu target */`
  - Bedingte Sprünge
    - Sprünge abhängig von Bedingung
    - `beq target /* Ein Beispiel, eq für equal */`
  - Label
    - Label sind Namen für Adressen im Programm
    - Name muss unterschiedlich von Maschinenbefehlen (Mnemonics) sein
    - Label müssen mit einem Doppelpunkt abgeschlossen werden
    - Werden zur Markierung von Stellen für Sprünge verwendet (target)
- **Bedingte Sprünge**

```
mov r0,#4       /* r0 = 4 */
add r1,r0,r0    /* r1 = 8 */
cmp r0, r1      /* r0 - r1 = -4: NZCV = 1000 */
                /* StatusBits NZCV */
beq there       /* Kein Sprung: Z != 1 */
                /* Müsste bei Gleichheit (equal) 0 sein */
add r1,r1,#42   /* r1 = r1 + 42 */

there:
add r1,r1,#78   /* r1 = r1 + 78 */
```

  - Weitere Bedingungen:
    - `beq`: Equal / Gleichheit
    - `bne`: Not Equal / Ungleichheit
    - `bge`: Greater / Größer

- `ble`: Less / Kleiner

- **if-Anweisung**

```
/* r0 = 5; r1 = 10; r2 = f; r3 = i */
cmp r0,r1          /* Vergleicht r0 und r1 */
bne L1             /* Falls Werte ungleich sind, ist hier gegeben */
add r2,r3,#1       /* Wird hier übersprungen */
L1:                /* Hierhin wird gesprungen */
sub r2,r2,r3
```

- **if/else-Anweisung**

```
/* r0 = 5; r1 = 10; r2 = f; r3 = i */
cmp r0,r1
bne L1             /* Potentieller Sprung nach L1 */
add r2,r3,#1       /* else-Anweisung (wird übersprungen, falls Bedingung korrekt) */
b L2               /* Überspringen der if-Anweisung, sonst wird beides ausgeführt */
L1:
sub r2,r2,r3
L2:
...
```

- **while-Schleifen**

```
/* r0 = pow; r1 = x */
mov r0,#1
mov r1,#0
WHILE:             /* Label für Schleife*/
cmp r0,#128        /* Abbruchbedingung: Falls equal Z = 1 */
beq DONE           /* Sprung aus Schleife */
lsl r0,r0,#1       /* Linksshift um 1 Bit / Schleifencode */
add r1,r1,#1       /* x = x + 1 / Schleifencode */
b WHILE            /* Fortführen der Schleife */
DONE:
...
```

- **for-Schleifen**

```
/* r0 = i; r1 = sum */
mov r1,#0
mov r0,#0
FOR:               /* Label für Schleife */
cmp r0,#10         /* Abbruchbedingung: Falls i größer als 10 ist */
bge DONE
add r1,r1,r0       /* sum = sum + i */
add r0,r0,#1       /* i = i + 1 */
b FOR              /* Fortführen der Schleife */
DONE:
...
```