I. GRUNDLAGEN

Aufgaben der Hardware:

Ein- und Ausgabe von Daten
Verarbeiten von Daten
Speichern von Daten

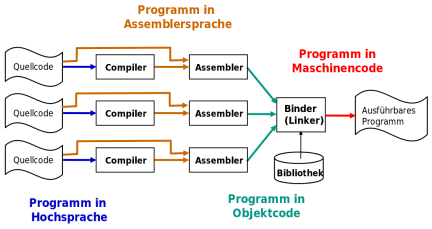
Klassische Hardwarekomponenten:

Ein- und Ausgabe
Hauptspeicher
Rechenwerk
Leitwerk

II. ANFORDERUNGEN HÖHERER PROGRAMMIERSPRACHEN

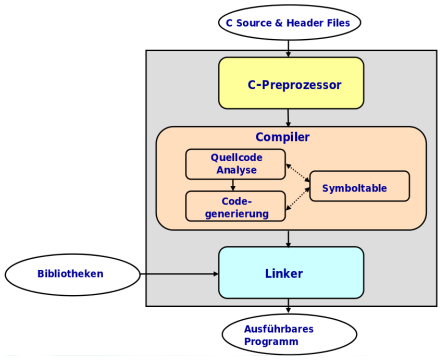
Begriffe:

Maschinensprache: Für Prozessor verständliche Anweisungsrepräsentation, z.B. 00101101001110101
Assemblersprache: Für Menschen verständliche Maaschinensprache, z.B. `add $s2, $s1, $s0`
Assembler: Übersetzt Assemblersprache eindeutig in Maschinensprache
Objektcode: Maschinenprogramm mit ungelösten externen Referenzen
Binder/Linker: Löst ungelöste Referenzen auf, verbindet alles zu einem ausführbaren Maschinenprogramm



Programmiersprache C:

Zwischenstellung zwischen Assembler und Hochsprache
hohe Portabilität trotz guter Architekturanpassung
einfache Programmierung
Datentypen: `char, int, float, double`
Kontrollstrukturen: Entscheidungen, Schleifen, Blöcke, Unterprogramme
Zeiger als Parameter möglich



C - Datentypen:

`char`: Ein Zeichen, meist 1 Byte
`int`: Integerzahl, 2 oder 4 Byte
`float`: Gleitkommazahl, meist 4 Byte
`double`: Gleitkommazahl, meist 8 Byte

C - Operatoren:

`*`: Multiplikation (`x*y`)
`/`: Division (`x/y`)
`%`: Modulo (`x%y`)
`+`: Addition (`x+y`)
`-`: Subtraktion (`x-y`)
`+` und `-` auch als Prä- und Postfix, alle auch als assign (`=` anhängen)

C - Bit-Operatoren:

`~`: Bitwise NOT (`~x`)
`<<`: links schieben (`x<<y`)
`>>`: rechts schieben (`x>>y`)
`&`: bitweise AND (`x&y`)
`^`: bitweise XOR (`x^y`)
`|`: bitweise OR (`xy|`)
alle auch als Assign (`=` anhängen)

C - Vergleichsoperatoren:

`>,<`: größer, kleiner als (`x>y, x<y`)
`>=,<=`: größergleich, kleinergleich als (`x>=y, x<=y`)
`==,!=`: gleich, ungleich (`x==y, x!=y`)

C - Spezialoperatoren:

Auswahloperator: `z = (a < b) ? a : b` (`z=a`, falls `a<b`, sonst `z=b<`)

C - Operatoren-Priorität

Operator Type	Operator	Associativity
Primary Expression Operators	`() [] . -> expr++ expr--`	left-to-right
Unary Operators	`* & + - ! ~ ++expr --expr (typecast) sizeof`	right-to-left
Binary Operators	`* / %`	left-to-right
	`+ -`	
	`>> <<`	
	`< > <= >=`	
	`== !=`	
	`&`	
	`^`	
	`|`	
	`&&`	
	`||`	
Ternary Operator	`?:`	right-to-left
Assignment Operators	`= += -= *= /= %= >>= <<= &= ^= |=`	right-to-left
Comma	`,`	left-to-right

C - Kontrollstrukturen

```
if (Bedigung) { Aktionen_if } else { Aktionen_else }
switch (var) { case a: ... break; ... default: ... break; }
while (Bedigung) { ... }
for (init; Bedingung; reinit) { ... }
do { ... } while (Bedingung)
```

C - Programmaufbau

1. Präprozessor-Anweisungen:
   (a) `#include <stdio.h>` (Bibliotheken einbinden)
   (b) `#include "modul.h"` (Module einbinden)
   (c) `#define COLOR blau` (Globale Textersetzung)
2. Globale Deklarationen/Definitionen:
   (a) `int i;` (Deklaration)
   (b) `int j = 13;` (Definition)
   (c) `int fakultaet (int n);` (Funktionsprototyp)
3. Funktionen/Programmstruktur

   `int fakultaet (int n) { ... }`
   jedes Programm enthält Funktion `void main(...) { ... }`
   Unterprogramm = Funktion
   Programmstart: `main` wird aufgerufen
   Rekursion ist zulässig

C - Parameterübergabe

1. Call by Value: Normalfall, Kopie des Parameters wird an Funktion übergeben, bei Änderung keine Auswirkung beim Aufrufer
2. Call by Reference: Mit Zeigern umsetzbar, selbe Speicheradresse wie Aufrufer

C - globale und lokale Variablen

Global: Sind gesamtem Programm bekannt (zu vermeiden)
Lokal: Nur in Block deklariert

C - Speicherklassen

`auto`: lokale Variablen
`register`: wird in CPU-Register gespeichert, nur für zeitkritische Variablen zu verwenden
`static`: statischer Speicherplatz
`extern`: globale Variable

C - Zeiger und Vektoren

Pointer: Enthält Adresse, die auf Daten verweist
`int* p` (`p` ist Zeiger auf `int`)
`a = 3; p = &a` (`p` enhält Adresse von `a`)
`int b = *p + 1` (=4)

```
int *p;
int *q;
int a = 3;
int b;

p = &a;
b = *p + 1;
q = (int*) 0x8010
```

	Adresse	Inhalt
p	...	0x8004
a	0x8004	3
b	...	4
q	...	0x8010
	0x8010	