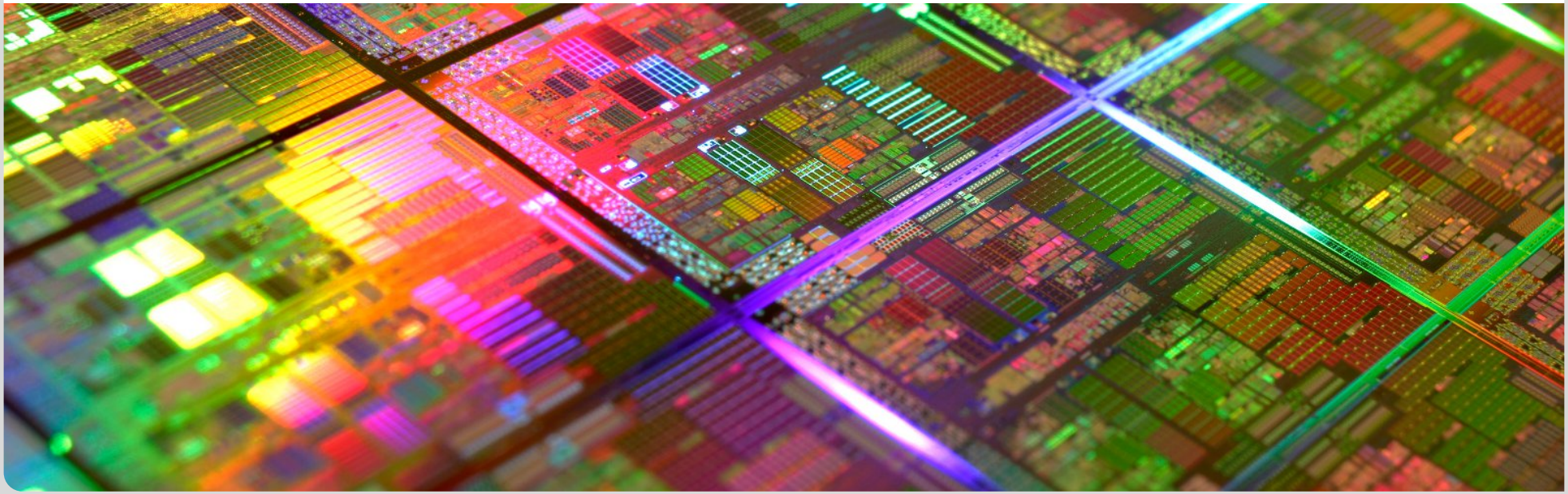
Karlsruher Institut für Technologie

Tutorium Rechnerorganisation

Woche 2

Tutorien 3 und 4 zur Vorlesung Rechnerorganisation



Heute

- C-Buildumgebung
- Datentypen in C
- Kontrollstrukturen in C
- Operatoren in C
- C-Zeiger

Einleitung

- Es wird in Rechnerorganisation wahrscheinlich keine C-Programmieraufgaben geben
- (Grundsätzliche) C-Kenntnisse sind aber dennoch nötig
- Um Code ausprobieren zu können, empfiehlt es sich, eine C-Buildumgebung auf seinem Rechner einzurichten

Notwendige Komponenten

- Benötigt werden:
  - Compiler (z.B. gcc)
  - Linker (z.B. gcc)
  - C-Standardbibliothek inkl. Header (z.B. glibc)
- Am besten ein “Gesamtpaket” installieren:
  - GNU Toolchain unter Linux
  - Cygwin bzw. mingw mit Dev-C++ unter Windows
  - Komplettpaket mit IDE von Microsoft (Visual Studio)

Ganzzahlige C-Datentypen

- signed/unsigned char (immer 1 Byte)
- signed/unsigned short [int] (oft 2 Byte)
- signed/unsigned int (oft 4 Byte)
- signed/unsigned long [int] (oft 4 oder 8 Byte)
- signed/unsigned long long [int] (oft 8 Byte)
- Die Größe der einzelnen Datentypen ist bis auf Mindestgrößen und die Größe von char nicht vorgeschrieben!
- Es gilt aber immer:
  $sizeof(short\ int) \leq sizeof(int) \leq sizeof(long\ int)$

C-Datentypen für Gleitkommazahlen

- short (“single precision”, 4 Byte)
- double (“double precision”, 8 Byte)
- Der Aufbau von Gleitkommazahlen ist standardisiert (IEEE 754) und wird später noch behandelt

Sonstige C-Datentypen

- Zeiger
- Arrays (Strings als Spezialfall)
- Selbstdefinierte Strukturen (struct)
- Enumerationen (enum)
- Unions (union)

Kontrollstrukturen

- Sollten bereits aus Java (1. Semester) bekannt sein
- Bedingte Ausführung und Fallunterscheidung:
  - if (<Bedingung>) {...} [else ...]
  - switch (<Ausdruck>) {case <Wert>: ...; case <Wert>: ...; default: ...}
- Schleifen:
  - for (<Initialisierung>; <Bedingung>; <Iteration>) {...}
  - while (<Bedingung>) {...}
  - do {...} while (<Bedingung>);

Weitere Schlüsselwörter zur Kontrollflusssteuerung

- break: Innerste Schleife sofort verlassen
- continue: Nächsten Schleifendurchlauf sofort starten
- return: Funktion (ggf. mit Rückgabewert) sofort verlassen
- goto: Spärlich verwenden!

Operatoren

- Arithmetische Operatoren: +, -, *, /, %, a++, ++a, a--, --a
- Logische Operatoren: !, ==, !=, <, >, >=, <=, &&, ||
- Bitweise Operatoren: &, |, ^, <<, >>, ~
- Zuweisungsoperatoren: =, += (etc.), <<= (etc.), &= (etc.)
- Sonstiges: &, *, a?b:c

Deklaration von Zeigern

- Zeiger sind wichtiges Konzept in C!
- Ein Zeiger ist eine Variable, deren Wert eine Speicheradresse ist
- Werden bei der Deklaration durch einen Stern gekennzeichnet
- Bei Deklaration angeben, auf welchen Datentyp er zeigen wird (Achtung: Keine Prüfung zur Laufzeit!) oder einen void-Zeiger deklarieren
- Beispiele:
  - int*, char*, char**, void**
- Häufig: “int *a” statt “int* a”
  - “int* a, b” erzeugt einen Zeiger a und eine int-Variable b!
  - “int *a, *b” bzw. “int *a, b” sind besser lesbar

Verwendung von Zeigern

- Mit dem Adressoperator (&) ermittelt man die Adresse einer Variable
- Mit dem Dereferenzierungsoperator (*) greift man auf die Speicherzelle zu, deren Adresse ein Zeiger enthält
- Nicht verwechseln: Stern bei der Deklaration vs. Stern als Dereferenzierungsoperator!
- Pfeil-Operator (“a->b”): Kurzschreibweise für “(*a).b”
- Aber: Wozu braucht man Zeiger überhaupt?

Weiteres zu Zeigern

- Zeiger können wie andere C-Datentypen gecastet werden (schlechter Stil!)
- Im Unterschied zu Java-Referenzen keinerlei Schutzmechanismen:
  - Keine Garantie, dass Zeiger auf deklarierten Typ zeigt
  - Keine Garantie, dass Zeiger-Ziel aligned (ausgerichtet) ist
  - Keine Garantie, dass Zeiger auf gültigen Speicher zeigt (Nullzeiger sind explizit erlaubt und werden gezielt eingesetzt)
  - Was passiert, wenn man (im x86-Protected Mode) auf ungültigen Speicher zugreift?

Fertig!