Assembly

Programming Concepts & Paradigms

Melvin Werthmüller, Christopher Christensen

Was ist Assembly?

- Low-level Programmiersprache
- Abstraktion von Maschinencode für bessere Verständlichkeit
- Spezifisch für bestimmte Computer-Architektur (sprich Prozessoren)
- Instruktionen in symbolischen Code repräsentiert
- Wird in ausführbaren Maschinencode konvertiert durch Dienstprogramm (Assembler), wie NASM oder MASM

https://www.tutorialspoint.com/assembly_programming/

Vorteile

- weniger Memory und Ausführzeiten
- Erlaubt hardwarespezifisch komplexere Aufgaben einfacher umzusetzen
- geeignet für zeitkritische Aufgaben
- geeignet für Interrupt Service Routines
- geeignet für Memory Resident Programme

https://www.tutorialspoint.com/assembly_programming/

Architekturen

(Typen von Prozessoren)

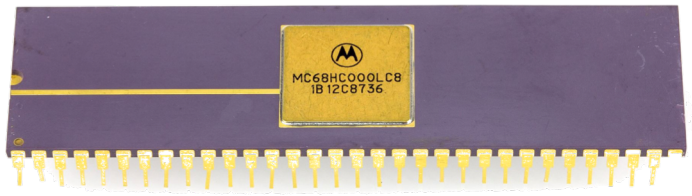
- CISC
  - **C**omplex **I**nstruction **S**et **C**omputer
  - Ziel: Ein Task in so wenig Zeilen wie möglich abarbeiten
  - Vorfahre von RISC
- RISC
  - **R**educed **I**nstruction **S**et **C**omputer
  - Ziel: Einfache Instruktionen in einem Clock-Cycle abarbeiten
  - Wurde wegen des Fortschreitens von 8- und 16-Bit- zu 32-Bit-Architekturen benötigt

Viele Prozessoren heute sind ein Mix aus CISC und RISC

https://www.tutorialspoint.com/assembly_programming/

RISC vs. CISC

CISC	RISC
Schwerpunkt Hardware	Schwerpunkt Software
Multi-clock komplexe Anweisungen	Single-clock (reduced instruction only)
Weniger Code, hohe Zyklen pro Sekunde	Mehr Code, niedrige Zyklen pro Sekunde
Transistoren zum Speichern verwendet komplexe Anweisungen	Verwendet mehr Transistoren in den Speicherregistern
Memory-to-memory: "LOAD" und "STORE" in Anweisungen eingebettet	Register-to-register: "LOAD" und "STORE" unabhängige Anweisungen



Beispiele CISC: Motorola 68000 (68K), DEC VAX, PDP-11, mehrere Generationen von Intel x86, und 8051

Beispiele RISC: MIPS, PowerPC, Atmel's AVR, Microchip PIC processors, Arm processors, RISC-V, und alle modernen Mikroprozessoren haben mindestens Elemente von RISC

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

Assembly

Basic Instructions: Syntax

Anweisung	Beschreibung
mov	Bewegt einen Wert in ein Register (move)
add	Addition
sub	Subtraktion
push	Wert auf Stack pushen
pop	Wert von Stack entfernen
int	Interrupt

```
operation [operand, …]
```

```
mov ebx, 123          ; ebx  = 123
add ebx, ecx          ; ebx += ecx
```

https://www.tutorialspoint.com/assembly_programming/

Assembly

Basic Instructions: System Calls

- Methoden, um vom Betriebssystem bestimmte Funktionalitäten aufzurufen (z.B. Lesen oder Schreiben von Daten)
- Kontrolle wird während der Ausführung dem Kernel übergeben

https://www.tutorialspoint.com/assembly_programming/

Assembly

Basic Instructions: System Calls

eax	System Call
1	sys_exit
2	sys_fork
3	sys_read
4	sys_write
5	sys_open
6	sys_close

Vorgehen:

1. **System Call** in Register eax speichern
2. **Argumente** zum System Call in die Register ebx, ecx, edx speichern
3. **Interrupt** mit ox8o ausführen

https://www.tutorialspoint.com/assembly_programming/

Assembly

Basic Instructions: System Calls

1. System Call in Register eax speichern

```
mov eax, 1          ; sys_exit
mov ebx, 0          ; exit status = 0
int 0x80            ; System Call ausführen
```

2. Argument zum System Call in das Register ebx speichern

3. Interrupt mit 0x80 ausführen (0x80 = Interrupt Handler für System Calls)

https://www.tutorialspoint.com/assembly_programming/

Assembly

Basic Instructions: Jumps

- Problem: benötigen einen Weg, um die Reihenfolge der Instruktionen zu ändern
- Lösung: Jumps
- **Unconditional Jumps** (JMP)
- **Conditional Jumps**
- Bedingungen mit CMP überprüft

```
CMP DX, 00   ; Wert von DX wird mit 0 verglichen
JE  L7       ; Falls zutrifft, springe zu L7
.
.
L7: ...
```

https://www.tutorialspoint.com/assembly_programming/assembly_conditions.htm

Assembly

Zusammenspiel mit HW: Registers

- The registers store data elements for processing without having to access the memory. A limited number of registers are built into the processor chip
- General registers
  - Data registers
  - Pointer registers
  - Index registers
- Control registers
- Segment registers

https://www.tutorialspoint.com/assembly_programming/

Assembly

Zusammenspiel mit HW: Memory Addressing

- Addressen sind Referenzen auf Daten, die der Prozessor während dem Ausführen eines Programs verwendet
- 3 Formen
  - **Register Addressing** (Register der Operanden)
  - **Immediate Addressing** (Konstanten)
  - **Direct Memory Addressing** (Direkter Zugriff auf Memory)

https://www.tutorialspoint.com/assembly_programming/

Assembly

Zusammenspiel mit HW: Memory Addressing

- **Register Addressing**

```
mov eax, ebx
; Wert von ebx ins Register eax bewegen
```

- **Immediate Addressing**

```
mov AX, LEN
; die Konstante LEN wird ins Register AX bewegt
```

- **Direct Memory Addressing**

```
mov BX, WORD_VALUE
```

https://www.tutorialspoint.com/assembly_programming/

Assembly

Code Beispiel

- Stack Counter (Basic Instructions)
- Fibonacci (RISC)

https://www.tutorialspoint.com/assembly_programming/