

# Rechnertechnik

Prof. Dr. rer. nat. Peter Gerwinski

25. Mai 2021

# Rechnertechnik

## 1 Einführung

## 2 Vom Schaltkreis zum Computer

2.1 Logik-Schaltkreise

2.2 Binärdarstellung von Zahlen

2.3 Vom Logik-Schaltkreis zum Addierer

2.4 Negative Zahlen

2.5 Vom Addierer zum Computer

2.6 Computer-Sprachen

2.7 Programmieren in Assembler

2.8 Struktur von Assembler-Programmen

## 3 Architekturmerkmale von Prozessoren

## 4 Der CPU-Stack

...

## 2.8 Struktur von Assembler-Programmen

### Beispiel 1: IA-32-Assembler

```
addl $1, %eax  
movb %al, b  
cmpb (%ebx), %dl  
jbe .L2
```

## 2.8 Struktur von Assembler-Programmen

### Beispiel 1: IA-32-Assembler

Befehl    Größen-Suffix



```
addl $1, %eax  
movb %al, b  
cmpb (%ebx), %dl  
jbe .L2
```

## 2.8 Struktur von Assembler-Programmen

### Beispiel 1: IA-32-Assembler – Adressierungsarten

Befehl    Operanden



addl \$1, %eax

movb %al, b

cmpb (%ebx), %dl

jbe .L2

## 2.8 Struktur von Assembler-Programmen

### Beispiel 1: IA-32-Assembler

unmittelbar

addl \$1, %eax ← Register

movb %al, b ← Speicher (absolut)

cmpb (%ebx), %dl

jbe .L2

indirekt mit Register

Speicher (relativ)

The diagram illustrates four assembly instructions with red arrows pointing to specific operands and their types:

- `addl $1, %eax`: An arrow points from the label `$1` to the word `unmittelbar` (immediate). Another arrow points from the label `%eax` to the word `Register`.
- `movb %al, b`: An arrow points from the label `b` to the word `Speicher (absolut)` (absolute memory).
- `cmpb (%ebx), %dl`: An arrow points from the label `(%ebx)` to the word `indirekt mit Register` (indirect with register).
- `jbe .L2`: An arrow points from the label `.L2` to the word `Speicher (relativ)` (relative memory).

## 2.8 Struktur von Assembler-Programmen

Beispiel 2: Redcode (ICWS '88) – Core War[s] (Krieg der Kerne)

Virtuelle Maschine: Memory Array Redcode Simulator (MARS)

Instruktionen:

**dat** B – Daten – „Du hast verloren!“

**mov** A, B – kopiere A nach B

**add** A, B – addiere A zu B

**sub** A, B – subtrahiere A von B

**jmp** A – unbedingter Sprung nach A

**jmz** A, B – Sprung nach A, wenn  $B = 0$

**jmn** A, B – Sprung nach A, wenn  $B \neq 0$

**djn** A, B – „decrement and jump if not zero“

**cmp** A, B – „compare“: überspringe, falls gleich

**spl** A – „split“: Programm verzweigen

Adressierungsarten:

grundsätzlich: Speicher relativ

**#** – unmittelbar

**\$** – direkt

**@** – indirekt

**<** – indirekt mit Prä-Dekrement

Programm „Nothing“:

**jmp** 0

Programm „Knirps“:

**mov** 0, 1

## 2.8 Struktur von Assembler-Programmen

Unbedingte Verzweigung

Beispiel: Endlosschleife von „Dwarf“

```
bomb  dat #0
start add #4, bomb
      mov bomb, @bomb
      jmp start
      end start
```

Instruktionen:

```
dat B
mov A, B
add A, B
sub A, B
jmp A
jnz A, B – „jump if zero“
jmn A, B – „if not zero“
djn A, B – „dec. & jmn“
cmp A, B – vgl. & überspr.
spl A – verzweigen
```

Adressierungsarten:

grundsätzlich:  
Speicher relativ

# – unmittelbar

\$ – direkt

@ – indirekt

< – indirekt mit  
Prä-Decrement



## 2.8 Struktur von Assembler-Programmen

Bedingte Verzweigung

Beispiel: Kopierschleife von „Mice“

```
ptr    dat #0
start  mov #12, ptr
loop   mov @ptr, <dest
       djn loop, ptr
       spl @dest
       add #653, dest
       jnz start, ptr
dest   dat #833
       end start
```

Instruktionen:

```
dat B
mov A, B
add A, B
sub A, B
jmp A
jnz A, B – „jump if zero“
jmn A, B – „if not zero“
djn A, B – „dec. & jmn“
cmp A, B – vgl. & überspr.
spl A – verzweigen
```

Adressierungsarten:

grundsätzlich:  
Speicher relativ

# – unmittelbar

\$ – direkt

@ – indirekt

< – indirekt mit  
Prä-Decrement

## 2.8 Struktur von Assembler-Programmen

Mehrere Threads

Beispiel: Multithreading von „Mice“

ptr	dat #0
start	mov #12, ptr
loop	mov @ptr, <dest
	djn loop, ptr
	spl @dest
	add #653, dest
	jmz start, ptr
dest	dat #833
	end start

Instruktionen:

dat B  
mov A, B  
add A, B  
sub A, B  
jmp A  
jmz A, B – „jump if zero“  
jmn A, B – „if not zero“  
djn A, B – „dec. & jmn“  
cmp A, B – vgl. & überspr.  
spl A – verzweigen

Adressierungsarten:

grundsätzlich:  
Speicher relativ

# – unmittelbar

\$ – direkt

@ – indirekt

< – indirekt mit  
Prä-Decrement

## 2.8 Struktur von Assembler-Programmen

Selbstmodifizierender Code

Beispiel: Selbsterkennung von „Fini“

```
num    dat #-2
start  mov num, <pos
       jmp start
pos    dat
       end start
```

Instruktionen:

```
dat B
mov A, B
add A, B
sub A, B
jmp A
jnz A, B – „jump if zero“
jmn A, B – „if not zero“
djn A, B – „dec. & jmn“
cmp A, B – vgl. & überspr.
spl A – verzweigen
```

Adressierungsarten:

grundsätzlich:  
Speicher relativ

# – unmittelbar

\$ – direkt

@ – indirekt

< – indirekt mit  
Prä-Decrement

## 2.8 Struktur von Assembler-Programmen

### Ehemaliger Praktikumsversuch (2014)

Inspiration für Projekt?

Schreiben Sie ein  
Redcode-Programm,  
das die Gegner  
**Nothing**, **Knirps**  
und **Mice** besiegt.

ICWS-88-Standard,  
max. 64 Prozesse,  
Speichergröße zufällig

Teams bis zu 3 Personen  
sind zulässig.

Instruktionen:

**dat** B  
**mov** A, B  
**add** A, B  
**sub** A, B  
**jmp** A  
**jmz** A, B – „jump if zero“  
**jmn** A, B – „if not zero“  
**djn** A, B – „dec. & jmn“  
**cmp** A, B – vgl. & überspr.  
**spl** A – verzweigen

Adressierungsarten:

grundsätzlich:  
Speicher relativ

**#** – unmittelbar

**\$** – direkt

**@** – indirekt

**<** – indirekt mit  
Prä-Decrement

# Rechnertechnik

## 1 Einführung

## 2 Vom Schaltkreis zum Computer

2.1 Logik-Schaltkreise

2.2 Binärdarstellung von Zahlen

2.3 Vom Logik-Schaltkreis zum Addierer

2.4 Negative Zahlen

2.5 Vom Addierer zum Computer

2.6 Computer-Sprachen

2.7 Programmieren in Assembler

2.8 Struktur von Assembler-Programmen

## 3 Architekturmerkmale von Prozessoren

## 4 Der CPU-Stack

...