

Rechnertechnik

Prof. Dr. rer. nat. Peter Gerwinski

18. 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

...

2.7 Programmieren in Assembler

Cross-Entwicklung

- Programmieren auf PC

- Compilieren auf PC:

```
avr-gcc -Wall -Os -mmcu=atmega328p blink.c -o blink.elf
```

- Speicherabbild auf PC erstellen:

```
avr-objcopy -O ihex blink.elf blink.hex
```

- Speicherabbild auf den Mikrocontroller herunterladen:

```
avrdude -P /dev/ttyACM0 -c arduino -p m328p \  
-U flash:w:blink.hex
```

- Präprozessor auf PC:

```
avr-gcc -Wall -Os -mmcu=atmega328p blink.c \  
-E -o blink.E
```

- Compilieren auf PC, Assembler-Quelltext erzeugen:

```
avr-gcc -Wall -Os -mmcu=atmega328p blink.c -S
```

2.7 Programmieren in Assembler

C nach Assembler übersetzen:

```
$ gcc -S pruzzel.c
```

erzeugt `pruzzel.s`,

Assembler für den Standard-Prozessor
(hier: 64-Bit-AMD-Architektur – amd64).

```
$ avr-gcc -S pruzzel.c
```

erzeugt `pruzzel.s`,

Assembler für 8-Bit-Atmel-AVR-Prozessoren.

2.7 Programmieren in Assembler

C-Programme auf Assembler-Ebene debuggen:

```
$ gcc -g pruzzel.c -o pruzzel  
$ gdb -tui ./pruzzel  
(gdb) break main  
(gdb) run  
(gdb) layout split  
(gdb) nexti
```

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

The diagram illustrates four IA-32 assembly instructions with red arrows pointing to specific operands and their addressing modes:

- `addl $1, %eax`: An arrow points from the red text "unmittelbar" to the immediate value `$1`. Another arrow points from the red text "Register" to the register operand `%eax`.
- `movb %al, b`: An arrow points from the red text "Speicher (absolut)" to the memory operand `b`.
- `cmpb (%ebx), %dl`: An arrow points from the red text "indirekt mit Register" to the register operand `%ebx` within the parentheses.
- `jbe .L2`: An arrow points from the red text "Speicher (relativ)" to the label operand `.L2`.

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