

Rechnertechnik

Prof. Dr. rer. nat. Peter Gerwinski

11. 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

3 Architekturmerkmale von Prozessoren

4 Der CPU-Stack

...

2.6 Computer-Sprachen

2.6.1 Maschinensprache

Computer

- Rechenwerk (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Befehle abarbeiten
- Peripherie: Kommunikation mit der Außenwelt

→ in Maschinensprache programmierbar

In jedem Takt:

- dort aus dem Hauptspeicher lesen, wohin das Register **IP** zeigt
→ *Befehl* (*Instruction* – **IP** = *Instruction Pointer*)
- den *Befehl* an den *Funktion*-Eingang der *ALU* legen
- auf ähnliche Weise weitere Daten an den *Akkumulator*- und den *Daten*-Eingang der *ALU* legen
- auf ähnliche Weise den *Ergebnis*-Ausgang der *ALU* in den Hauptspeicher schreiben → Befehl ausgeführt
- Register **IP** hochzählen → nächster Befehl

→ Maschinensprache

2.6 Computer-Sprachen

2.6.1 Maschinensprache

- Daten aus dem Speicher in Register einlesen
- Daten aus Register in den Speicher schreiben

→ Lade- und Speicher-Befehle

- Daten aus Registern (oder Speicher) an ALU legen, Ergebnis in Register (oder Speicher) schreiben

→ arithmetische Befehle

- Daten aus Register oder Speicher in das **IP**-Register laden

→ unbedingter Sprungbefehl

- Sprungbefehl nur dann, wenn im Status-Ausgang der ALU ein bestimmtes Bit gesetzt ist

→ bedingter Sprungbefehl

→ Der Computer kann „alles“ – *Turing-Vollständigkeit*

2.6 Computer-Sprachen

2.6.1 Maschinensprache

- Lade- und Speicher-Befehle
arithmetische Befehle
unbedingte und bedingte Sprungbefehle

→ Der Computer kann „alles“ – *Turing-Vollständigkeit*

- Maschinensprache = Zahlen → für Menschen schwer handhabbar

→ Namen für die Befehle: *Mnemonics*

→ *Assembler*-Sprache

2.6.2 Maschinensprache → Assembler

Beispiel: Intel-x86-16-Bit-Assembler

- Lade- und Speicher-Befehle
arithmetische Befehle
unbedingte und bedingte Sprungbefehle
- Register

mov, ...
add, sub, inc, dec, xor, cmp, ...
jmp, jz, jae, ...
ax, bx, ...

Beispiel: Atmel-AVR-8-Bit-Assembler

- Lade- und Speicher-Befehle
arithmetische Befehle
unbedingte und bedingte Sprungbefehle
- Register

ldi, lds, sti, ...
add, sub, subi, eor, cp, ...
rjmp, brsh, brlo, ...
r0, r1, ...

→ für jeden Prozessor anders

2.6.3 Maschinensprache → Assembler → Hochsprachen

Beispiel: Intel-x86-16-Bit-Assembler

- Lade- und Speicher-Befehle
- arithmetische Befehle
- unbedingte und bedingte Sprungbefehle
- Register

mov, ...
add, sub, inc, dec, xor, cmp, ...
jmp, jz, jae, ...
ax, bx, ...

Beispiel: Atmel-AVR-8-Bit-Assembler

- Lade- und Speicher-Befehle
- arithmetische Befehle
- unbedingte und bedingte Sprungbefehle
- Register

ldi, lds, sti, ...
add, sub, subi, eor, cp, ...
rjmp, brsh, brlo, ...
r0, r1, ...

→ für jeden Prozessor anders

Hochsprache → für jeden Prozessor gleich

2.6.3 Maschinensprache \longrightarrow Assembler \longrightarrow Hochsprachen

Compiler-Sprachen

- *Compiler* übersetzt Hochsprachen-*Quelltext* in die Assembler-Sprache
- *Assembler* übersetzt Assembler-Quelltext in die Maschinensprache
- Compiler und Assembler sind Programme, geschrieben in Maschinensprache, Assembler oder einer Hochsprache
- Beispiele: Fortran, Algol, Pascal, Ada, C, C++, ...

Interpreter- oder Skript-Sprachen

- *Interpreter* liest Hochsprachen-*Quelltext* und führt ihn sofort aus
- Der Interpreter ist ein Programm, geschrieben in Maschinensprache, Assembler oder einer Hochsprache
- Beispiele: Unix-Shell, BASIC, Perl, Python, ...

Kombinationen

- *Compiler* erzeugt *Zwischencode* für eine *virtuelle Maschine*
- *Interpreter* liest Hochsprachen-*Zwischencode* und führt ihn sofort aus
- Die virtuelle Maschine ist ein Programm, geschrieben in Maschinensprache, Assembler, einer Hoch- oder Skript-Sprache
- Beispiele: UCSD-Pascal, Java, ...

2.7 Programmieren in Assembler

Beispiel: PC, 1980er bis 1990er Jahre

- Prozessor: Intel 8086
- Takt: 4,77–100 MHz

Anwendung von Assembler: zeitkritische Programmteile,
z. B. Text- und Grafikausgabe

```
28F5:0117 cmp al,0
28F5:0119 jnz 10f
28F5:011B ret
28F5:011C
-u 100
28F5:0100 B800B8      MOV     AX,B800
28F5:0103 8EC0        MOV     ES,AX
28F5:0105 BE0002      MOV     SI,0200
28F5:0108 BF000A      MOV     DI,0A00
28F5:010B B40D        MOV     AH,0D
28F5:010D 8A04        MOV     AL,[SI]
28F5:010F 268905      MOV     ES:[DI],AX
28F5:0112 47          INC     DI
28F5:0113 47          INC     DI
Hello, world!        INC     SI
28F5:0115 8A04        MOV     AL,[SI]
28F5:0117 3C00        CMP     AL,00
28F5:0119 75F4        JNZ     010F
28F5:011B C3          RET
28F5:011C 0D5DCA      OR      AX,CA5D
28F5:011F 0800        OR      [BX+SI],AL
-g=100
```

Program terminated normally (0000)

2.7 Programmieren in Assembler

Beispiel: PC, 1980er bis 1990er Jahre

- Prozessor: Intel 8086
- Takt: 4,77–100 MHz

Anwendung von Assembler: zeitkritische Programmteile,
z. B. Text- und Grafikausgabe

Beispiel: Arduino Uno

- Prozessor: ATmega 328p
- Takt: 16 MHz

Anwendung von Assembler: zeitkritische Programmteile,
z. B. Mikrosekunden-Timing

2.7 Programmieren in Assembler

Beispiel: Arduino Uno

- Prozessor: ATmega 328p
- Takt: 16 MHz

Anwendung von Assembler: zeitkritische Programmteile,
z. B. Mikrosekunden-Timing

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
```

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`

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

3 Architekturmerkmale von Prozessoren

4 Der CPU-Stack

...