

Rechnertechnik

Prof. Dr. rer. nat. Peter Gerwinski

3. 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 Struktur von Assembler-Programmen

3 Architekturmerkmale von Prozessoren

4 Der CPU-Stack

...

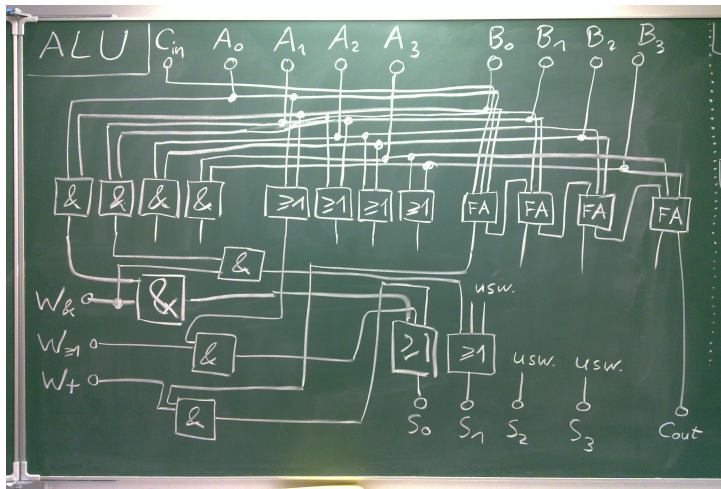
2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

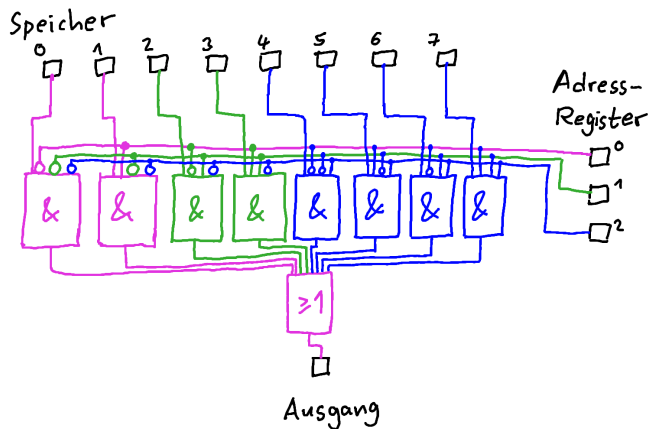
- Rechenwerk: Arithmetisch-logische Einheit (ALU)



2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

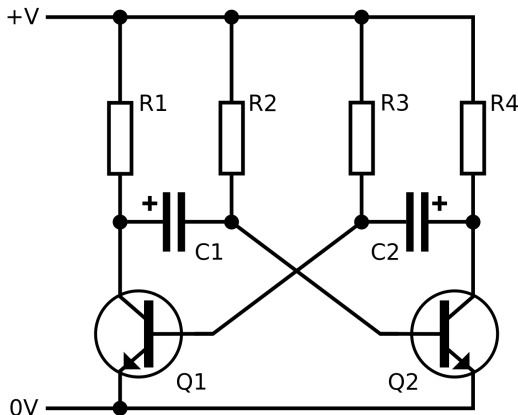
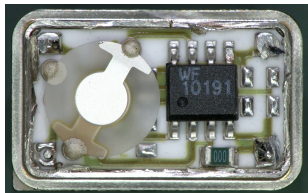
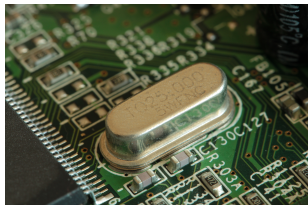
- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher



2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

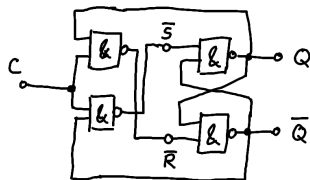
- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt:



2.5 Vom Addierer zum Computer

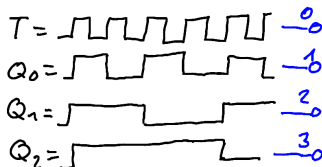
Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen



„Stromstoßschalter“

Funktions-Eingänge der ALU



Adress-Register
Speicher Bit für Bit auslesen



Zähler-Schaltung

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen und Befehle abarbeiten
 - Register-Ladebefehl:
„Lade Speicherzelle 42 an den A-Eingang der ALU.“
 - Rechenbefehl:
„Berechne die Summe der beiden Register an der ALU.“
 - Register-Speicherbefehl:
„Speichere den Status-Ausgang der ALU in Speicherzelle 137.“

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen und Befehle abarbeiten
 - Register-Ladebefehl:
„Lade Speicherzelle 42 an den A-Eingang der ALU.“
 - Rechenbefehl:
„Berechne die Summe der beiden Register an der ALU.“
 - Register-Speicherbefehl:
„Speichere den Status-Ausgang der ALU in Speicherzelle 137.“
 - Sprungbefehl:
„Lade den Inhalt der Speicherzelle 1117
in das Adressregister für den nächsten Befehl.“

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen und Befehle abarbeiten
 - Register-Ladebefehl:
„Lade Speicherzelle 42 an den A-Eingang der ALU.“
 - Rechenbefehl:
„Berechne die Summe der beiden Register an der ALU.“
 - Register-Speicherbefehl:
„Speichere den Status-Ausgang der ALU in Speicherzelle 137.“
 - Sprungbefehl:
„Lade den Inhalt der Speicherzelle 1117
in das Adressregister für den nächsten Befehl.“
 - Bedingter Sprungbefehl:
„Wenn das C_{out} -Bit der ALU den Wert 1 hat,
springe nach Speicherzelle 23.“

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen und Befehle abarbeiten
- Peripherie: Kommunikation mit der Außenwelt
 - Output-Port:
Ausgang einer Speicherzelle nach draußen führen
 - Input-Port:
Anstelle einer Speicherzelle einen eingehenden Draht abfragen

2.5 Vom Addierer zum Computer

Bau eines Turing-vollständigen Computers

- Rechenwerk: Arithmetisch-logische Einheit (ALU)
- Speicher: Register, adressierbarer Hauptspeicher
- Takt: Speicher durchgehen und Befehle abarbeiten
- Peripherie: Kommunikation mit der Außenwelt

→ in Maschinensprache programmierbar

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 Struktur von Assembler-Programmen
- 3 Architekturmerkmale von Prozessoren
- 4 Der CPU-Stack
- ...

2.1 Computer-Sprachen

2.1.1 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

2.1 Computer-Sprachen

2.1.1 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

2.1 Computer-Sprachen

2.1.1 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

—> Sprungbefehl

2.1 Computer-Sprachen

2.1.1 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

—> Sprungbefehl

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

—> bedingter Sprungbefehl

2.1 Computer-Sprachen

2.1.1 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

2.1 Computer-Sprachen

2.1.1 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*

2.1 Computer-Sprachen

2.1.1 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

2.1.2 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, ...

```
31D6:0111 mov ax, 0e22
31D6:0114 mov [4041], ax
31D6:0117 ret
31D6:0118
-u 100
31D6:0100 B800B8      MOV     AX,B800
31D6:0103 8ED8      MOV     DS,AX
31D6:0105 B8410C      MOV     AX,0C41
31D6:0108 A30004      MOV     [0400],AX
31D6:010B B8620D      MOV     AX,0D62
31D6:010E A30204      MOV     [0402],AX
31D6:0111 B8220E      MOV     AX,0E22
31D6:0114 A30404      MOV     [0404],AX
31D6:0117 C3      RET
31D6:0118 AA      STOSB
31D6:0119 00508B      ADD     [BX+SI-75],DL
31D6:011C 86F8      XCHG    BH,AL
31D6:011E FD      STD
31D6:011F 050300      ADD     AX,0003
-g =100
```

Program terminated normally (0000)

2.1.2 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

2.1.3 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

2.1.3 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++, ...

2.1.3 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, ...

2.1.3 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, ...

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 Struktur von Assembler-Programmen
- 3 Architekturmerkmale von Prozessoren**
- 4 Der CPU-Stack**
- ...