Rechnerarchitektur

M. Jakob

Gymnasium Pegnitz

24. Februar 2019

Inhaltsverzeichnis

In diesem Abschnitt

Aufbau eines Computersytems

- Ü 1.1: Aufbau eines Computersystems

(a) Buch, S. 88f Kurzvortrag

(b) Ordne die realen Rechnerkomponenten den logischen Komponenten zu

- VorlageAufbau.graphml
`run:Arbeitsmaterial/Uebungen/Aufbau/VorlageAufbau.graphml`

Übungen

- Ü 1.2: Preishit

(a) Erkläre, um welche Komponenten es hier geht (Hilfe: Website MediaMarkt)

- VorlagePreishit.graphml
`run:Arbeitsmaterial/Uebungen/Preishit/VorlagePreishit.graphml`

(b) Entwirf zwei Preishit-Auto-Werbungen mit Komponenten, die überhaupt nicht zusammenpassen (keine Grafik).

- Ü 1.3: Supercomputer

Fasse den Wikipedia-Artikel zum Thema

- Supercomputer
`http://de.wikipedia.org/wiki/Supercomputer`

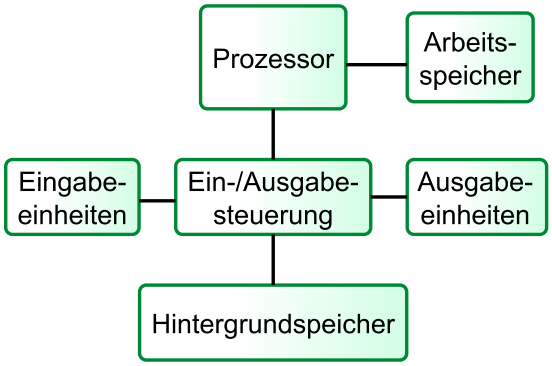
zusammen. Achte darauf, dass du nur Dinge erwähnst, die du einigermaßen erklären kannst. Ein Geschwindigkeitsvergleich mit handelsüblichen Rechnern muss enthalten sein.

Aufbau eines Computersystem

Aufbau eines Computersystem

Die wesentlichen Komponenten eines Computers sind auf der Hauptplatine (engl. Motherboard) untergebracht. Die Hauptplatine enthält

- einen Sockel für den Prozessor
- Slots für den Arbeitsspeicher
- die Steuerelektronik
- Slots und Stecker für Ein- und Ausgabeeinheiten, Hintergrundspeicher und div. Erweiterungen



In diesem Abschnitt

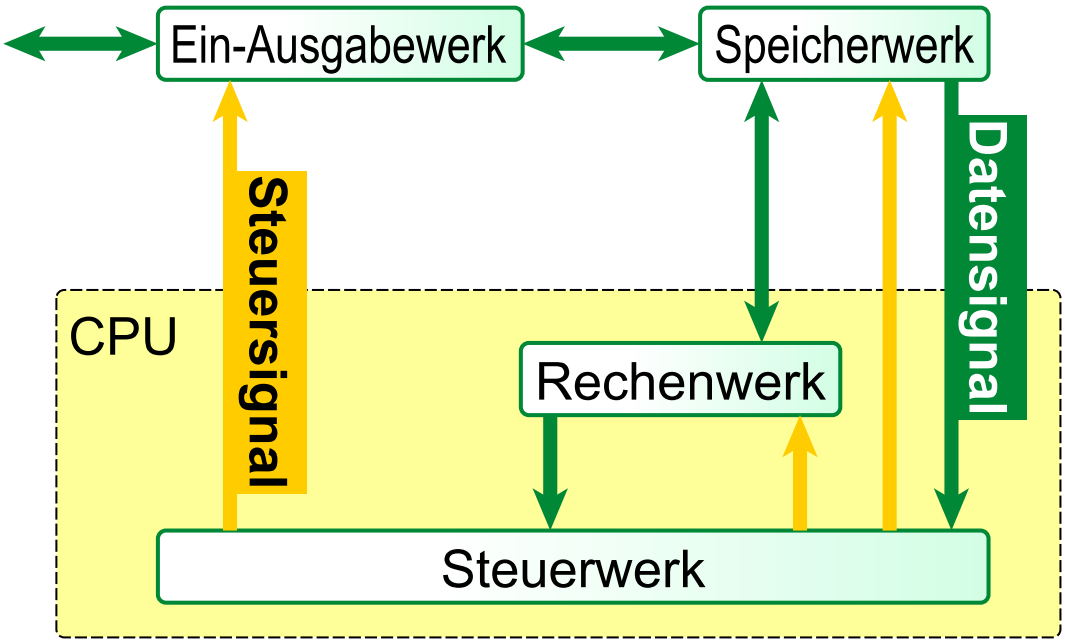
Von-Neumann-Rechner (Registermaschine)

Von-Neumann-Rechner (Registermaschine)

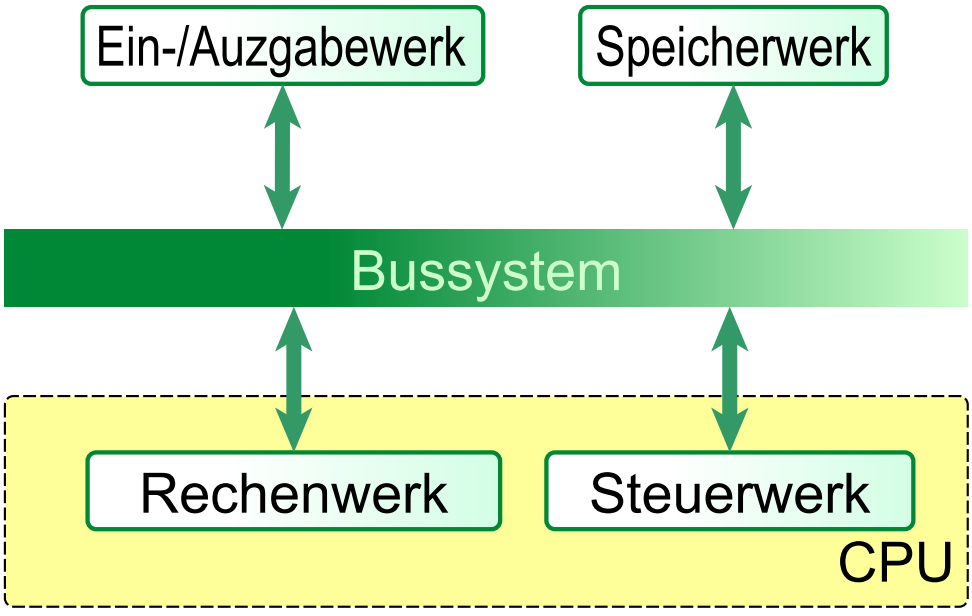
Fast alle gängigen Rechner sind Von-Neumann-Rechner, die folgende Komponenten haben:

Speicherwerk	speichert Programme und Daten
Rechenwerk	führt elementare Rechenoperationen aus
Steuerwerk	steuert Ablauf und Informationsaustausch
Ein-/Ausgabewerk	regelt die Kommunikation mit der Außenwelt
Bussytem	zentral gesteuertes und getaktetes Übertragungssystem bestehend aus Daten-, Adress- und Steuerbus.

Von-Neumann-Rechner schematisch



Von-Neumann-Rechner noch schematischer



Von-Neumann-Rechner: Details

- Der Speicher ist in gleichgroße Zellen unterteilt, die fortlaufend nummeriert sind.
- Programme und Daten sind binär codiert. Sie werden in demselben Speicher abgelegt.
- Aufeinander folgende Befehle eines Programms werden in aufeinander folgenden Speicherzellen abgelegt und nacheinander abgearbeitet. Zusätzlich existieren Sprungbefehle.
- Elementare Operationen sind u.a.
  - arithmetische Befehle, z.B. Addieren oder Multiplizieren;
  - logische Befehle, z.B. Vergleiche;
  - Transportbefehle, z.B. zum Transport von Befehlen und Daten vom Speicher zum Rechenwerk und umgekehrt;
  - Sprungbefehle (mit oder ohne Bedingung).

Übungen

• Ü 1.4: „Menschlicher“ Computer

Spiele die Multiplikation zweier kleiner natürlicher Zahlen nach der Von-Neumann-Architektur in einem Rollenspiel nach (Klett, S. 99/2)

• Ü 1.5: ROM

S. 97/2

Binär- und Hexadezimaldarstellung von Zahlen

- Rechnen mit Stift und Papier . . .

Der Ganzzahlkreis

Der Ganzzahlkreis

In einer Speicherzelle mit n Bits können höchstens 2^n verschiedene Zahlen abgespeichert werden. Üblicherweise wird dabei das vorderste Bit als Vorzeichenbit verwendet und somit die Zahlen $-2^{n-1} \ldots 2^{n-1} - 1$ codiert.

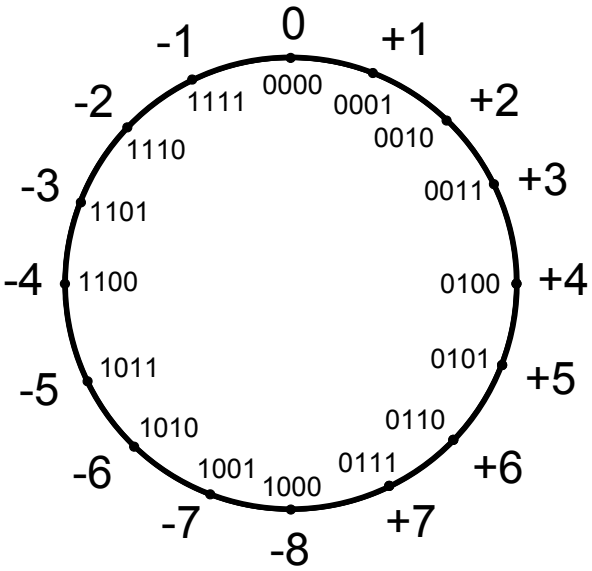
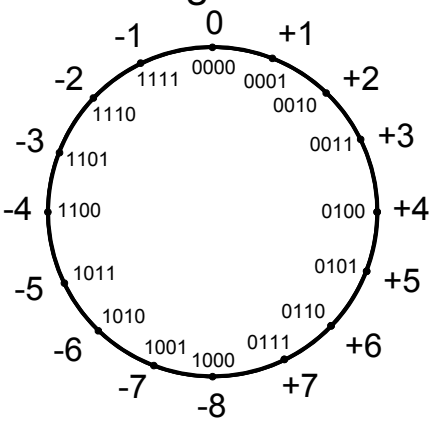


Abbildung: *4-Bit Ganzzahlkreis*

Binäraddition

Das Ergebnis Addition $a + b$ erhält man . . .

auf dem Zahlenkreis, wenn man von a insgesamt b Schritte im Uhrzeigersinn wandert



durch den üblichen Additionsalgorithmus, wenn man Überlaufbits wegfallen lässt.

$$\begin{array}{rcccc} & 0 & 1 & 0 & 1 \quad 5_{10} \\ + & 0_1 & 1 & 0_1 & 1 \quad 5_{10} \\ \hline (0) & 1 & 0 & 1 & 0 \quad -6_{10} \end{array}$$

$$\begin{array}{rcccc} & 1 & 0 & 1 & 1 \quad -5_{10} \\ + \ {\scriptstyle 1} & 1 & 0_1 & 1_1 & 1 \quad -5_{10} \\ \hline (1) & 0 & 1 & 1 & 0 \quad 6_{10} \end{array}$$

- Ganzzahlüberlauf mit BlueJ-Direkteingabe demonstrieren

Übungen

- Ü 2.1: Binär- und Hex-Darstellung

(a) S. 97/3

(b) S. 98/4
Zusatz: Gib eine Regel an, mit der man zu einer Binärzahl die negative Binärzahl findet. (Die Regel heißt Zweierkomplement bilden)

(c) Verwandle die angegebenen Ganzzahlen eines 16-Bit-Rechner in die anderen beiden Darstellungen und übe das Addieren in allen drei Zahlensystemen.

- $(1000\ 0000\ 1010\ 0110)_2$; $(0110\ 1100\ 1110\ 0101)_2$
- $(E70B)_{16}$; $(2A86)_{16}$; $(15932)_{10}$; $(7206)_{10}$

- Rechner zur Kontrolle
`http://binaer-dezimal-hexadezimal-umrechner.miniwebapps.de/`

Definition Register

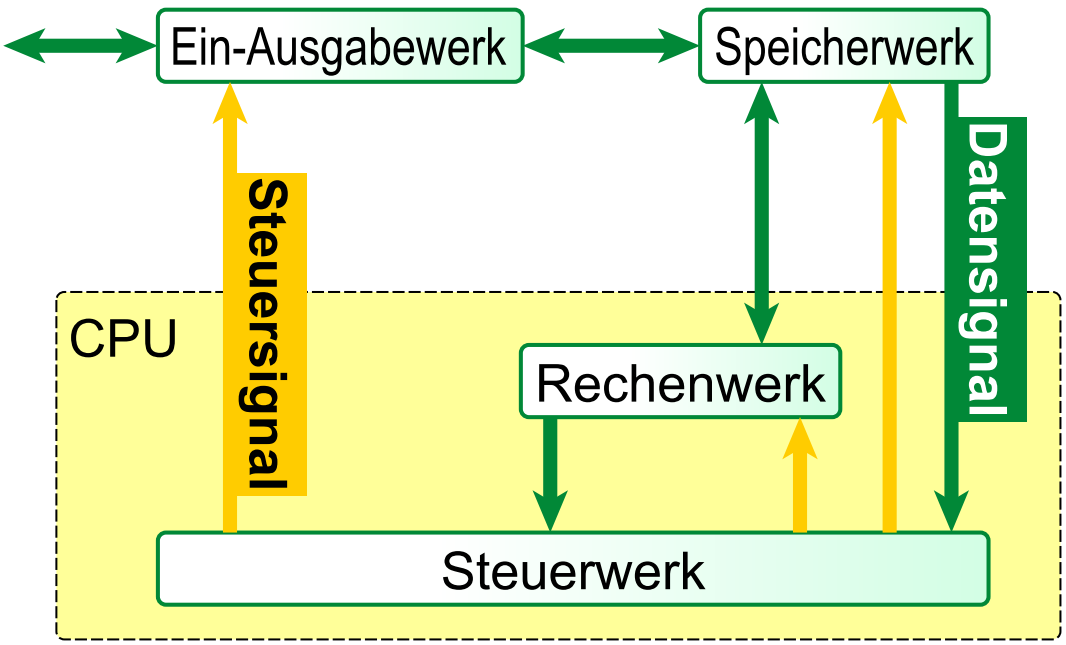
Definition Register

Register sich Speicherzellen, die in die CPU integriert sind und auf die ein schneller Zugriff möglich ist. Registermaschinen sind Computer, die Register nutzen.

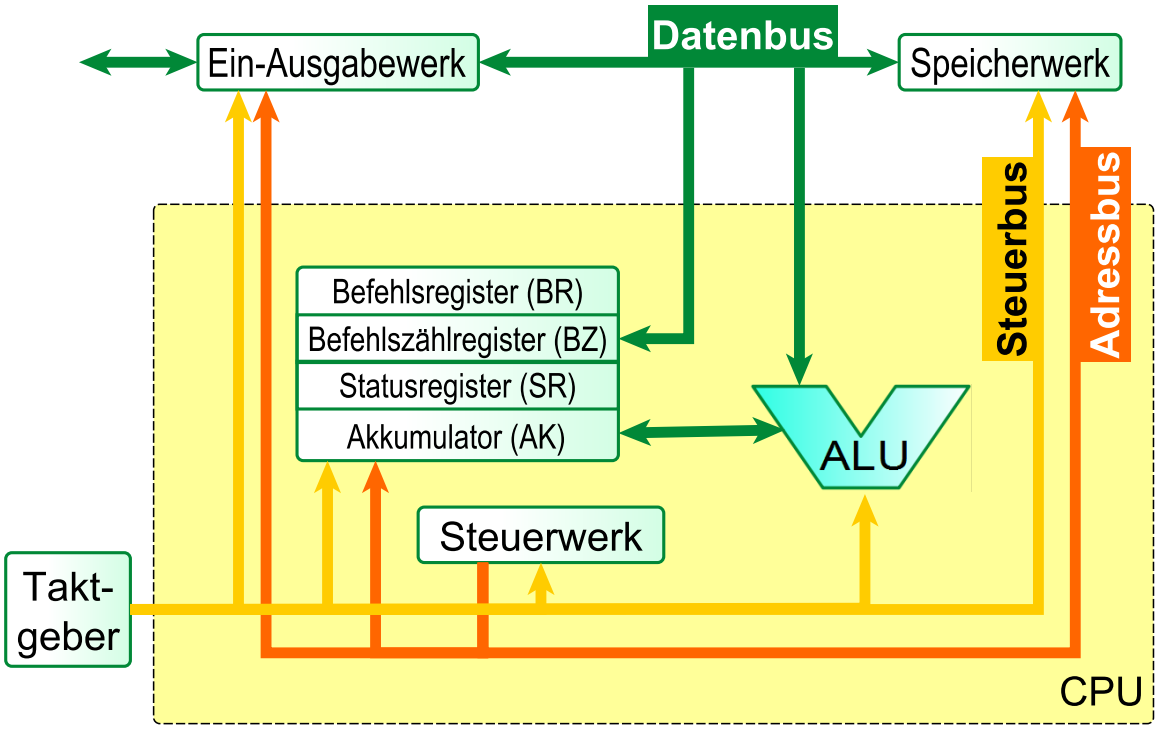
Beachte

- Wenn wir hier von Speichern sprechen, ist niemals ein externer Speicher wie die Festplatte oder DVD gemeint. Die sind für diese Zwecke viel zu langsam.
- Bei uns sind Registermaschinen immer Erweiterungen der Von-Neumann-Rechner.

Von-Neumann-Rechner (Wiederholung)



Registermaschine: Schematischer Aufbau



In diesem Abschnitt

Die Zentraleinheit die Registermaschine im Einzelnen

Die Zentraleinheit der Maschine besteht aus

- dem Rechnenwerk — es wird auch als Arithmetisch-Logische-Einheit (ALU) bezeichnet,
- dem Steuerwerk das die Befehlsverarbeitung steuert, somit Befehle lädt und interpretiert.
- Vier Register (Speicher in der CPU mit Spezialaufgaben)
  - das Befehlsregister
  - das Befehlszählregister
  - einen Akkumulator
  - das Statusregister oder Flagregister

Eigenschaften der Register

Befehlsregister speichert den momentan ausgeführten Befehl,

Befehlszählregister enthält die Speicheradresse den Befehls, der als nächstes ausgeführt wird,

Akkumulator kann eine ganze Zahl speichern,

Flag-Register Hier werden Informationen über das Ergebnis der letzten ausgeführten Rechenoperation hinterlegt. Typische Flags sind

- N(egativ)-Flag
- Z(ero)-Flag
- O(verflow)-Flag

Die Flags werden gesetzt, wenn das entsprechende Ereignis eingetreten ist.

In diesem Abschnitt

Das Bussytem die Registermaschine

Das Bussystem besteht aus Datenleitungen, auf die jede der daran angeschlossenen Komponenten lesend oder schreibend zugreifen kann. Dabei werden unterschieden:

Datenbus Damit werden die Daten zwischen den Komponenten übertragen. Pro Arbeitstakt kann der Inhalt einer Speicherzelle übertragen werden.

Adressbus Dieser Bus ist für die Übertragung von Speicheradressen zuständig.

Steuerbus Über diesen Bus wird das gesamte Bussystem gesteuert.

- Festlegung der Lese- und Schreibsteuerung beim Datenbus, also die Richtung der Datenübertragung.
- Übertragung des Arbeitstaktes.

Übung

- Ü 3.1: Kommunikation zwischen Prozessor und Arbeitsspeicher

Lies im Buch S.95ff den Abschnitt „Kommunikation zwischen Prozessor und Arbeitsspeicher“ durch und fertige eine schriftliche Kurzzusammenfassung an.

In diesem Abschnitt

Assembler-Programme

Assembler-Programme

enthalten nur sehr elementare, rechnerabhängige Befehle. Dazu gehören

Transportbefehle zum Laden der Datenregister mit Werten (`LOAD, DLOAD, STORE, ...`)

Sprungbefehle mit und ohne Bedingung (`JGE, JUMP, ...`)

Logische Verknüpfungen (`AND, OR, NOT, ...`)

Programmende (`END`)

- Beispiel eines Befehlssatzes: Buch S. 102ff

Der Maschinenbefehl-Grundzyklus

Der Maschinenbefehl-Grundzyklus

Alle Befehle einer Maschinensprache werden in einem Zyklus ausgeführt

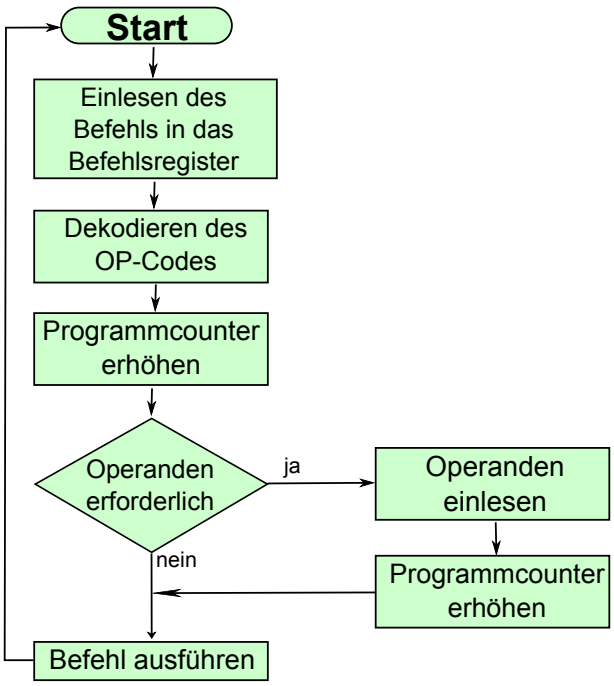
FETCH (1. Teil) Befehl holen und Befehlszähler inkrementieren.

DECODE Befehl decodieren.

FETCH (2. Teil) Befehl vervollständigen (z.B. wenn bei der Addition mehrere Operanden zu laden sind) und Befehlszähler inkrementieren.

EXECUTE Operation im Rechenwerk ausgeführen und Befehlszähler richtig setzen (z.B. bei Sprüngen).

Der Maschinenbefehl-Grundzyklus



Übung

- Buch S. 100ff durchgehen
- Operationsprinzip einer Registermaschine
- Ü 3.2: S. 104/1
- Ü 3.3: S. 104/2

In diesem Abschnitt

Mein erstes Assemblerprogramm

- RiSa: Anleitung und Befehlssätze

`run:Simulationen/RiSa/Anleitung.pdf`

```
1 load 10
2 add 11
3 store 12
4 hlt
```

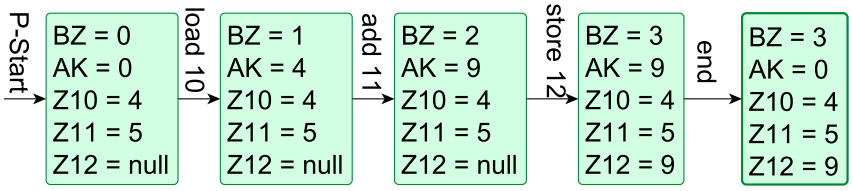
```
1 lade AK mit Inhalt von Z10
2 addiere zum Wert des AK den Inhalt von Z11,
      Ergebnis im AK
3 Speichere den Inhalt des AK in Z12
4 Beende das Programm
```

Assemblerprogramme als Zustandsübergänge

Assemblerprogramme als Zustandsübergänge

- Der Zustand der Registermaschine ist festgelegt durch den momentanen Inhalt der Speicherzellen und der Register.
- Eine Änderung des Zustands, also ein Zustandsübergang, wird dabei durch die Ausführung eines Maschinenbefehls bewirkt.
- Das Befehlsregister enthält den Befehl, der für den Zustandsübergang verantwortlich ist. Dieser Registerinhalt ist für den eigentlichen Zustand damit eigentlich nicht relevant.

Assemblerprogramm — Zustandsübergangsdiagramm und -Tabelle




```
1 load 10
2 add 11
3 store 12
4 hlt
```

	AK	Z10	Z11	Z12	BZ
P-Start	0	4	5		0
load 10	4	4	5		1
add 11	9	4	5		2
store 12	9	4	5	9	3
hlt	9	4	5	9	3

Assemblerprogramm — Zustandsübergangstabelle

Assemblerprogramm — Zustandsübergangstabelle

Den Ablauf eines Assemblerprogramms kann man übersichtlich in einer Zustandsübergangstabelle auflisten. In den Zeilen der Tabelle stehen die Programmmzeilen in der Reihenfolge der Ausführung, in den Spalten die Inhalte der relevanten Register und Speicherzellen.

	AK	Z10	Z11	Z12	BZ
P-Start	0	4	5		0
load 10	4	4	5		1
add 11	9	4	5		2
store 12	9	4	5	9	3
hlt	9	4	5	9	3

Und was steht wirklich im Speicher?

```
1 load 10
2 add 11
3 store 12
4 hlt
```

```
1 10111010
2 01011011
3 10011100
4 10000000
```

Bei einer 8-Bit-Maschine

- codieren die ersten vier Bit den Befehl,
- die nächsten vier den Operanden.
- heute sind 32- oder 64-Bit-Rechner üblich.

Übungen

- Ü 4.1: Einfache Terme

Gegeben sind folgende Terme

(i) $x - y$
(ii) $x + y + z$
(iii) $x - y + z$
(iv) $3 + x$
(v) $x + 3$
(vi) $x/7$

(a) Gib jeweils ein Assemblerprogramm zur Berechnung des Terms bei gegebenen Variablenwerten an. Die Variablenwerte sollen am Programmanfang in geeigneten Speicherzellen abgelegt werden, entsprechendes gilt für den berechneten Wert.
(b) Teste dein Programm für einige Variablenbelegungen und gib jeweils eine Zustandsübergangstabelle an.

Übungen

- Ü 4.2: Nicht so einfache Terme

Implementiere für folgende Probleme ein Assemblerprogramm.

(a) $v * x + y : z$ Beachte dabei die „Punkt-vor-Strich–Regel
(b) Bestimme den Mittelwert zweier Zahlen
(c) Vertauschen zweier Variablen.
Es sollen anfangs der Werte der Variablen *a* in der Speicherzelle 12, der von *b* in Zelle 13 liegen und nach Ende des Programms soll der Wert von *a* in Zelle 13 lieben und *b* in 12.

In diesem Abschnitt

Beispiel: zwei Werte sortieren

```
1 wenn
2   Wert Zelle 20 >Wert Zelle
      21
3 dann
4   tausche die Wert der
      Zellen
5 endewenn
```

```
0 load 20
1 sub 21
2 jmpn 9
3 load 20
4 store 22
5 load 21
6 store 20
7 load 22
8 store 21
9 hlt
```

Übungen

- Ü 4.3: Zwei Werte sortieren — ZÜT erstellen

Erstelle zu dem weiter oben abgebildeten Assemblerprogramm, das zwei Werte ggf. vertauscht zwei typische Zustandsübergangstabellen.

Bedingte Anweisung in Assembler umsetzen

Bedingte Anweisung in Assembler umsetzen

Eine Bedingung, speziell ein Vergleich zweier Werte x und y, kann durch ein Assemblerprogramm simuliert werden, indem zunächst die Differenz $x - y$ berechnet wird (Zeile 1). Die durch diese Operation im Statusregister gesetzten Flags werden dann von einem passenden Sprungbefehl ausgewertet (Zeile 2).

```
0 load 20
1 sub 21
2 jmpn 9
3 load 20
4 store 22
5 load 21
6 store 20
7 load 22
8 store 21
9 hlt
```

Übungen

- Ü 4.4: Addition mit Minimalbefehlssatz — ZÜT erstellen

(a) Erstelle zu dem abgebildeten Assemblerprogramm eine Zustandsübergangstabelle. In den Zellen 7 und 8 sollen die Werte 5 und 3 stehen.

(b) Beschreibe den Algorithmus, nach dem das Programm zwei Zahlen addiert.

(c) Begründe, dass bei Vertauschen der Zellinhalte 7 und 8 die Laufzeit des Programms wesentlich höher ist.

```
1 | mov akk 7
2 | mov ecx 8
3 | dec ecx
4 | inc akk
5 | jnz 2
6 | mov 9 akk
7 | hlt
```


Übungen

- Ü 4.5: Einseitige Bedingte Anweisungen

Implementiere für folgende Probleme ein Assemblerprogramm und erstelle eine ZÜT

(a) Wenn in Zelle 23 ein kleiner Wert als in Zelle 24 steht, dann soll Zelle 25 den Wert -1 erhalten

(b) Wenn in Zelle 20 ein größerer Wert als in Zelle 21 steht dann soll in Zelle 25 das Produkt der Zellen 20 und 21 stehen

(c) Wenn zwei Variablen x und y gleich sind, soll $z = 0$ sein.

In diesem Abschnitt

Zweiseitige bedingte Anweisungen

```
1 wenn
2   Bedingung
3 dann
4   Sequenz1
5 sonst
6   Sequenz2
7 endewenn
```

Strategie

- bei falscher Bedingung muss nach **sonst** gesprungen werden,
- am Ende von Sequenz1 muss nach **endewenn** gesprungen werden.

Beispiel: Maxiumum zweier Zahlen bestimmen

```
1 wenn a kleiner b
2 dann
3   lege b in c ab
4 sonst
5   lege a in c ab
6 endewenn
```

```
0 load 20  //a in 20
1 sub 21   //a-b im AK
2 jmpnn 6  //nFlag nicht
     gesetzt
3 load 21  //b laden
4 store 22 //b nach 22
5 jmp  8   //sonst-Teil
     überspringen
6 load 20  //a laden
7 store 22 //a nach 22
8 hlt
```


Übungen

- Ü 4.6: Zweiseitige bedingte Anweisungen

Erstelle jeweils ein Assemblerprogramm und eine ZÜT.

(a) Wenn der Akkumulatorwert gleich Null ist, soll $z = 1$ sein, sonst soll $z = 0$ sein (Flagsimulation).

(b) Von einer vorgegebenen Zahl a soll der Betrag $|a|$ berechnet werden.

(c) Von zwei Zahlen a und b soll das Maximum von 0 und $a - b$ bestimmt werden.

In diesem Abschnitt

Wiederholungen mit (Anfangs-)Bedingung

```
1 | wiederhole solange
      Bedingung
2 |   Sequenz
3 | endewiederhole
```

Strategie

- bei zutreffender Bedingung wird die Sequenz abgearbeitet
- **sonst** muss nach **endewiederhole** gesprungen werden.

Wiederholungen mit (Anfangs-)Bedingung als Wenn-Dann-Konstrukt mit Sprung

```
1  wenn
2    Bedingung
3  dann
4    Sequenz
5    < Springe zu wenn >
6  endewenn
```

Strategie

- bei zutreffender Bedingung wird die Sequenz abgearbeitet und wieder zur **wenn**-Zeile gesprungen,
- andernfalls muss nach **endewenn** gesprungen werden.

- Bsp Potenzberechnung an der Tafel

Beispiel: Potzenz berechnen

Es soll für $a, n \in \mathbb{Z}_0$ der Wert a^n berechnet werden.
Idee: a insgesamt n-mal mit sich selbst multiplizieren.

```
1  n=3 //Z21
2  a=5 //Z20
3  erg = 1 //Z22
4  1 //Z10
5  wenn n > 0
6    dann
7    erg=erg mal a
8    n=n-1
9    JMP <4>
10 endewenn
```

```
0  load 21
1  jmpz 8
2  sub 10
3  store
     21
4  load 20
5  mul 22
6  store
     22
7  jmp 0
8  hlt
```

- Zeile 1 testet das n-Flag
- Zeile 2 vermindert n
- Zeile 7 unbedingter Sprung zur Bedingung

Wiederholungen mit (Anfangs-)Bedingung...

Wiederholungen mit (Anfangs-)Bedingung...

...werden in einer Wenn-Dann-Konstruktion mit einem bedingten und einem unbedingten Sprung umgesetzt.

- Der bedingte Sprung (Zeile 1) legt die Abbruchbedingung der Schleife fest und steht vor der Wenn-Sequenz (Zeilen 2-7), der unbedingte Sprung steht am Ende der Wenn-Sequenz und führt zurück zur Schleifenabbruchdedingung.
- In der Wenn-Sequenz muss die Schleifenbedingung verändert werden, um endlos-Schleifen zu vermeiden (Zeile 2,3)

```
0 load 21
1 jmpz 8
2 sub 10
3 store 21
4 load 20
5 mul 22
6 store 22
7 jmp 0
8 hlt
```


Übungen

- Ü 4.7: Potenz-Variante

Im obigen Beispiel zur Berechnung der Potenz a^n wurde die Zählvariable rückwärts gezählt. Schreib das Programm so um, dass n bei 0 beginnend vorwärts zählt und erstelle eine ZÜT.

Übungen

- Ü 4.8: Teilbarkeit

Es soll der Rest bei der Division einer vorgegebenen positiven Zahl a durch 3 bestimmt werden. Dabei kann man folgende Idee ausnutzen: Von a wird solange der Wert 3 abgezogen, bis ein nicht negativer Wert übrig bleibt, der kleiner als 3 ist. Das muss dann der Rest sein.

(a) Vollziehe für $a = 11$ die Lösungsidee auf Papier nach.

(b) Formuliere die Idee als Algorithmus in Form eines Struktogramms oder in Pseudocode.

(c) Setze das Struktogramm in ein Assemblerprogramm um.

(d) Teste das Programm für $a \in \{2, 9, 11\}$.

(e) Fakultativ für Schnelle: Erweitere das Programm so, dass die Restberechnung auch bei negativen Zahlen möglich ist und dass nicht nur die Division durch 3 getestet wird.

Übungen

- Ü 4.9: GGT

Der abgebildete Algorithmus ermittelt zu zwei vorgegebenen positiven natürlichen Zahlen a und b den größten gemeinsamen Teiler:

(a) Welches bekanntes Problem wird in den Zeilen 4–6 gelöst?

(b) Vollziehe den Algorithmus mit einer Zustandstabelle für $a = 15$ und $b = 6$ nach.

(c) Implementiere ein entsprechendes Assemblerprogramm.

```
1 wiederhole solange b > 0
2   wiederhole solange a >= b
3     a = a - b
4   endewiederhole
5   ggt = b
6   b = a
7   a = ggt
8 endewiederhole
```

In diesem Abschnitt

Vergleich mit JAVA

```
1 wiederhole n-mal
2   Sequenz
3 endewiederhole
```

```
1 for(int i=0;i<n;i=i+1){
2    Sequenz}
```

Die Schleife besteht aus

- einer Zählvariable i mit einem Startwert: `for(int i = 0,*,*)`.
- die Zählvariable muss man bei jedem Durchlauf mit der Anzahl der gewünschten Durchläufe vergleichen: `for(*,i<n,*)`.
- die Zählvariable muss man bei jedem Durchlauf angepassen: `for(*,*,i=i+1)`.

Beispiel: $a * n$ ohne Multiplikation

Es soll für die Summe $a + a + \ldots + a$ von n Summanden berechnet werden.
Idee: a insgesamt n-mal aufaddieren.

```
1  a=5 //Z20
2  n=3 //Z21
3  erg = 0 //Z22
4  wenn n > 0
5  dann
6    erg=erg+a
7    n=n-1
8    JMP <4>
9  endewenn
```

```
0  LOAD 21
1  JMPZ 8
2  SUB 10
3  STORE
     21
4  LOAD 20
5  ADD 22
6  STORE
     22
7  JMP 0
8  HLT
```

- Zeile 1 testet das z-Flag
- Zeile 2 vermindert n
- Zeile 7 unbedingter Sprung zur Bedingung

Heavy-User–Übungen

- Ü 4.10: Komplexere Aufgaben

Erstelle jeweils ein Assemblerprogramm.

(a) Berechne $a * b - c$
(b) Berechne $c - a * b$
(c) Berechne $a + 2b + 3c + 4d$
(d) Berechne das Maximum dreier Zahlen
(e) Es soll festgestellt werden, ob eine Zahl gerade ist oder nicht
(f) Berechne a^n
(g) Klett, S.111f/1–8, v.a. Struktogramme