

## 3. Funktionsweise eines Rechners

### 3.3 Systemnahe Programmierung

Als Simulationsprogramm verwenden wir die Minimaschine  
( <https://schule.awiedemann.de/minimaschine.html> )

1	Vorbeleg:	LOADI 253
2		STORE 100
3		LOADI 244
4		STORE 101
5		
6	Start:	SUB 100
7		JMPNN DiffPos
8		LOAD 100
9		SUB 101
10		
11	DiffPos:	STORE 102
12		
13	Ende:	HOLD

- Programm öffnen oder schreiben
- Werkzeuge - Assemblieren

Das Programm und die Startwerte werden durch das Assemblieren in den Speicher geladen:

The screenshot displays a computer simulation interface with two main windows: 'Speicheranzeige' (Memory Display) and 'CPU-Kontrolle' (CPU Control).

**Speicheranzeige (Memory Display):** A table showing memory addresses (0 to 280) and their corresponding values. The values are loaded from a program and initial data.

Address	0	1	2	3	4	5	6	7	8	9
0	532	253	277	100	532	244	277	101	267	100
10	287	16	276	100	267	101	277	102	99	0
20	0	0	0	0	0	0	0	0	0	0
30										
40										
50										
60										
70										
80										
90										
100										
110										
120										
130										
140										
150										
160										
170										
180										
190										
200										
210										
220										
230										
240										
250										
260										
270										
280	0	0	0	0	0	0	0	0	0	0

**CPU-Kontrolle (CPU Control):** A window showing the internal state of the CPU. It includes a 'Datenbus' (Data Bus) and an 'Adressbus' (Address Bus). The CPU state is divided into two main sections: 'Akkumulator' (Accumulator) and 'Befehlsregister' (Instruction Register).

**Akkumulator:** Contains a 'Status' register with flags Z, N, and V. The value 0 is displayed in the accumulator.

**Befehlsregister:** Contains a 'Programmzähler' (Program Counter) with the value 0.

**Initial Data:** A table showing the initial data loaded into memory.

Address	0	1
0	532	
1	253	
2	277	
3	100	
...		
0	532	
1	253	

**Program Code:** A list of assembly instructions with their addresses.

Address	Instruction
1	Vorbeleg: LOADI 2
2	STORE 1
3	LOADI 2
4	STORE 1
5	
6	Start: SUB 100
7	JMPNN 1
8	LOAD 1
9	SUB 101
10	
11	DiffPos: STORE 1
12	
13	Ende: HOLD

**Buttons:** 'Ausführen' (Execute), 'Einzelschritt' (Single Step), and 'Mikroschritt' (Micro Step).

## Ausführen des Programms:

1 Vorbeleg: LOA

2 STO

3 LOA

4 STO

5

6 Start: SUB

7 JMP

8 LOA

9 SUB

10

11 DiffPos: STO

12

13 Ende: HOL

Speicheranzeige										
	0	1	2	3	4	5	6	7	8	9
0	532	253	277	100	532	244	277	101	267	100
10	287	16	276	100	267	101	277	102	99	0
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0
100	253	244	9	0	0	0	0	0	0	0
110										
120										
130										
140										
150										
160										
170										
180										
190										
200										
210										
220										
230										
240										
250										
260										
270										
280										
290										
300										
310										
320										
330										
340										
350										
360										

Datenbus

Akkumulator

9

Status

Z: N: V:

Befehlsregister

HOLD 0

Programmzähler

20

18 99

19 0

20 0

21 0

...

102 9

103 0

Adressbus

Ausführen

Einzelschritt

Mikroschritt

Befehlssatz:

Speicherbefehle	
<b>LOAD adresse</b>	Lädt den Wert von der angegebenen Adresse in den Akkumulator.
<b>LOADI zahl</b>	Lädt die angegebenen Zahl in den Akkumulator.
<b>STORE adresse</b>	Speichert den Wert im Akkumulator an der angegebenen Adresse.

Arithmetikbefehle	
<b>ADD adresse</b>	Addiert den Wert von der angegebenen Adresse zum Akkumulator.
<b>SUB adresse</b>	Subtrahiert den Wert der angegebenen Adresse vom Akkumulator.
<b>MUL adresse</b>	Multipliziert den Wert von der angegebenen Adresse zum Akkumulator.
<b>DIV adresse</b>	Dividiert den Wert im Akkumulator durch den Wert der angegebenen Adresse.
<b>MOD adresse</b>	Dividiert den Wert im Akkumulator durch den Wert der angegebenen Adresse und speichert den Rest im Akkumulator.
<b>CMP adresse</b>	Vergleicht den Wert der angegebenen Adresse mit dem Akkumulator und setzt Null- und Negativflag entsprechend. Dazu wird berechnet Akku - Wert von adresse
<b>ADDI zahl</b>	Addiert den angegebenen Wert zum Akkumulator.
<b>SUBI zahl</b>	Subtrahiert den angegebenen Wert vom Akkumulator.
<b>MULI zahl</b>	Multipliziert den angegebenen Wert zum Akkumulator.
<b>DIVI zahl</b>	Dividiert den Wert im Akkumulator durch den angegebenen Wert.
<b>MODI zahl</b>	Dividiert den Wert im Akkumulator durch den angegebenen Wert und speichert den Rest im Akkumulator.
<b>CMPI zahl</b>	Vergleicht den angegebenen Wert mit dem Akkumulator und setzt Null- und Negativflag entsprechend. Dazu wird berechnet Akku - zahl

Sprungbefehle	
<b>JMPP adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation positiv ( $> 0$ ) war, d. h. weder N noch Z-Flag sind gesetzt.
<b>JMPNN adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation nicht negativ ( $\geq 0$ ) war, d. h. das N-Flag ist nicht gesetzt.
<b>JMPN adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation negativ ( $< 0$ ) war, d. h. das N-Flag ist gesetzt.
<b>JMPNP adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation nicht positiv ( $\leq 0$ ) war, d. h. das N-Flag oder das Z-Flag ist gesetzt.
<b>JMPZ adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation zero ( $= 0$ ) war, d. h. das Z-Flag ist gesetzt.
<b>JMPNZ adresse</b>	Springt zur angegebenen Adresse, wenn das Ergebnis der letzten Operation nicht zero ( $\neq 0$ ) war, d. h. das Z-Flag ist nicht gesetzt.
<b>JMPV adresse</b>	Springt zur angegebenen Adresse, wenn die letzte Operation einen Überlauf verursacht hat, d. h. das V-Flag ist gesetzt.
<b>JMP adresse</b>	Springt zur angegebenen Adresse.

#### Sonstige Befehle

<b>HOLD</b>	Hält den Prozessor an.
<b>RESET</b>	Setzt den Prozessor auf den Startzustand zurück.
<b>NOOP</b>	Tut einfach nichts (NO OPeration).

#### Speicherorganisation

<b>WORD zahl</b>	Besetzt eine Speicherzelle mit der angegebenen Zahl.
------------------	------------------------------------------------------

## Umsetzung von einfachen Algorithmen:

### Sequenz:

Lade drei Werte in die Zellen [100], [101] und [102], berechne den Termwert  $([100]+[101]) \cdot [102]$  und speichere das Ergebnis in [103].

<b>Vorbelegung:</b>	<b>LOADI 502</b>
	<b>STORE 100</b>
	<b>LOADI 18</b>
	<b>STORE 101</b>
	<b>LOADI 25</b>
	<b>STORE 102</b>
<b>Start:</b>	<b>LOAD 100</b>
	<b>ADD 101</b>
	<b>MUL 102</b>
	<b>STORE 103</b>
<b>Ende:</b>	<b>HOLD</b>



#### Einseitige bedingte Anweisung:

Berechne von zwei Zahlen den Betrag der Differenz.

Algorithmus:

erg =[101]-[100]

wenn erg <0

erg=[100]-[101]

ende wenn

[102]=erg

<b>Vorbelegung:</b>	<b>LOADI 253</b>	
	<b>STORE 100</b>	
	<b>LOADI 244</b>	
	<b>STORE 101</b>	
<b>Start:</b>	<b>SUB 100</b>	berechnet [101]-[100]
	<b>JMPNN DiffPos</b>	ist das Ergebnis nicht negativ, kann gleich zu DiffPos gesprungen werden
	<b>LOAD 100</b>	
	<b>SUB 101</b>	berechnet [100]-[101]
<b>DiffPos:</b>	<b>STORE 102</b>	
<b>Ende:</b>	<b>HOLD</b>	

### 3.3. Systemnahe Programmierung

**Zweiseitig bedingte  
Anweisung:**

Ist eine Zahl gerade, so  
halbiere sie. Ist sie  
ungerade verdopple sie.

Algorithmus:

wenn  $[100] \bmod 2 = 0$   
 $[101] = [100]/2$   
ende wenn

sonst  
 $[101] = [100]*2$   
ende sonst

<b>Vorbelegung:</b>	<b>LOADI 24</b>	
	<b>STORE 100</b>	
<b>Start:</b>	<b>MODI 2</b>	berechnet den Rest bei der Division durch 2
	<b>JMPNZ Ungerade</b>	ist das Ergebnis ungleich 0, ist die Zahl ungerade. Springe zum sonst-Teil (Ungerade)
	<b>LOAD 100</b>	Wenn-Teil (halbiere die Zahl)
	<b>DIVI 2</b>	
	<b>JMP Ergebnis</b>	Überspringe den sonst-Teil!
<b>Ungerade:</b>	<b>LOAD 100</b>	Sonst-Teil (verdopple die Zahl)
	<b>MULI 2</b>	
<b>Ergebnis:</b>	<b>STORE 101</b>	
<b>Ende:</b>	<b>HOLD</b>	

### 3.3.Systemnahe Programmierung

#### Wiederholung mit Anfangsbedingung

Berechne die Summe aller ungeraden Zahlen  $15 + 13 + 11 + \dots + 3 + 1$ .

Algorithmus:

```
ergebnis = 15
summand = ergebnis - 2
wiederhole solange
summand > 0
  ergebnis = ergebnis +
  summand
  summand = summand - 2
ende wiederhole
```

<b>Vorbelegung:</b>	<b>LOADI 15</b>	
<b>Ergebnis:</b>	<b>STORE 102</b>	erster Summand und Ergebnis
	<b>SUBI 2</b>	
<b>Summand:</b>	<b>STORE 101</b>	nächster Summand
<b>Wiederhole:</b>	<b>LOAD 101</b>	lädt den aktualisierten nächsten Summanden
	<b>JMPN Ende</b>	ist der Summand negativ, wird die Wiederholung übersprungen
	<b>LOAD 102</b>	Ergebnis laden
	<b>ADD 101</b>	nächsten Summanden addieren
	<b>STORE 102</b>	Ergebnis speichern
	<b>LOAD 101</b>	wieder den Summanden laden
	<b>SUBI 2</b>	diesen um zwei verkleinern
	<b>STORE 101</b>	speichern
	<b>JMP Wiederhole</b>	zum Anfang der Schleife springen
<b>Ende:</b>	<b>LOAD 102</b>	
	<b>HOLD</b>	

### 3.3.Systemnahe Programmierung

**Wiederholung mit fester  
Anzahl :**

Verdopple n (z.B. 3) mal eine  
gegebene Zahl (z.B. 5) ;  
(5\*2\*2\*2)

Algorithmus:

anzahl = n  
zahl = 5  
ergebnis = zahl  
wiederhole anzahl mal  
ergebnis = ergebnis \* 2  
ende wiederhole

<b>Vorbelegung:</b>	<b>LOADI 5</b>	
	<b>STORE 101</b>	Zahl und Ergebnis speichern
	<b>LOADI 3</b>	
	<b>STORE 102</b>	Anzahl der Durchläufe
	<b>LOADI 1</b>	
	<b>STORE 103</b>	Zählvariable
<b>Wiederhole:</b>	<b>CMP 102</b>	vergleiche die Zählvariable mit der Anzahl der Durchläufe; berechnet [103] - [102]
	<b>JMPP Ende</b>	erreicht die Zählvariable den Wert 3, ergibt der Vergleich den Wert 0; die Schleife wird das letzte Mal durchlaufen.
	<b>LOAD 101</b>	Ergebnis laden
	<b>MULI 2</b>	verdoppeln
	<b>STORE 101</b>	Ergebnis speichern
	<b>LOAD 103</b>	Zählvariable laden
	<b>ADDI 1</b>	diese um 1 erhöhen
	<b>STORE 103</b>	speichern
	<b>JMP Wiederhole</b>	zum Anfang der Schleife springen
<b>Ende:</b>	<b>LOAD 101</b>	
	<b>HOLD</b>	