

**Hinweis:** Bei den folgenden Aufgaben bietet es sich an, den Ausgabestopp zu aktivieren, so dass der MOPS jedes Mal anhält, wenn etwas ins Ausgaberegister geschoben wird. Weiter geht es dann immer mit der Leertaste.

Später kann die Animation auch auf aus gestellt werden.

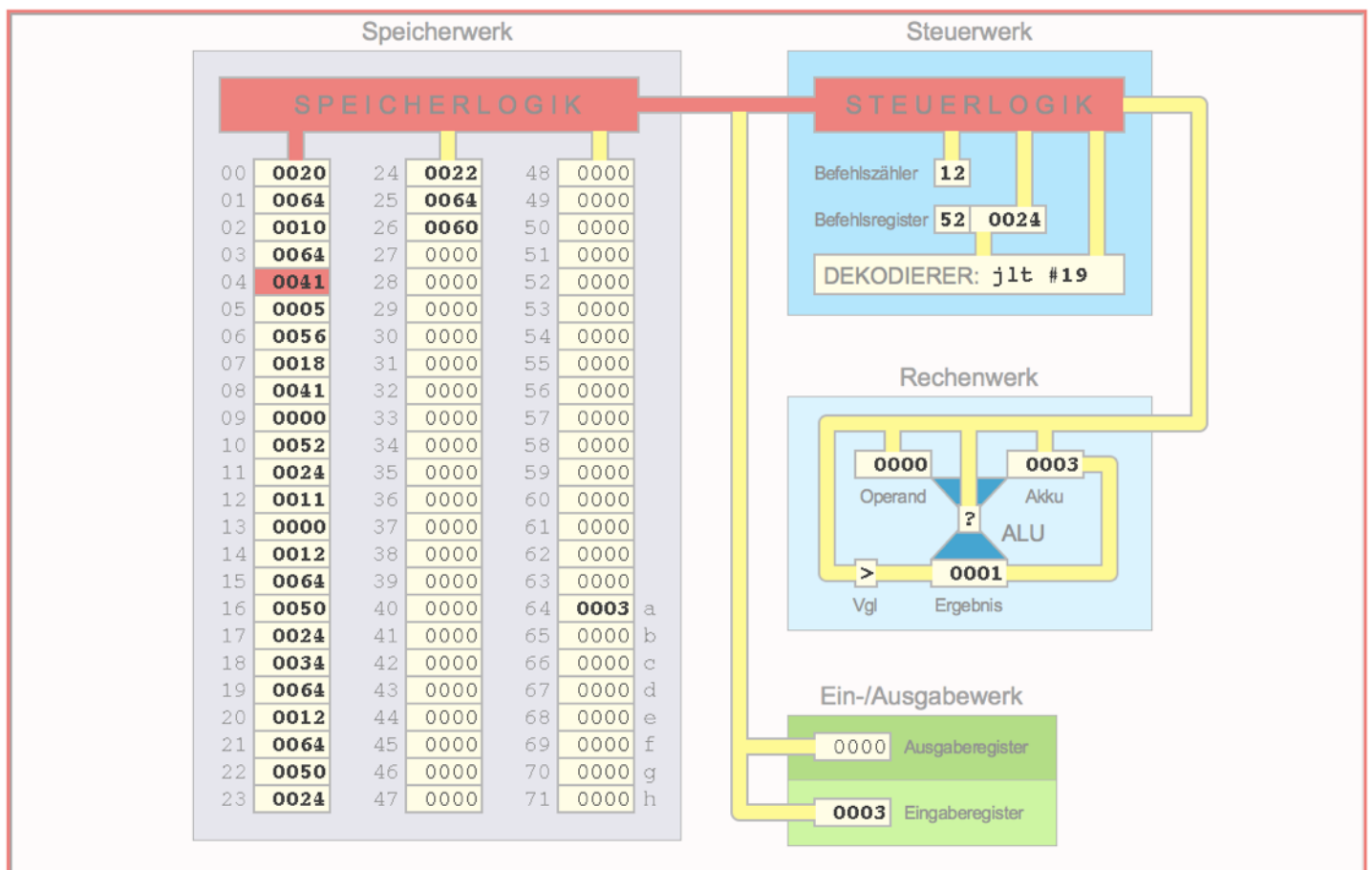
Ablauf **Ausgabestopp**

Animation **mittel**

## M O P S

MOdellrechner mit PSeudoAssembler

Aufgabensammlung **mit Lösungen**



## MOPS – Übersicht Befehle und Syntax

Befehl	Code	Funktion
ld <i>adr</i>	10	<b>load</b> : Lade den Wert an der Adresse <i>adr</i> in den Akku
ld <i>val</i>	11	<b>load</b> : Lade den Wert <i>val</i> in den Akku
st <i>adr</i>	12	<b>store</b> : Speichere den Wert des Akku an der Adresse <i>adr</i>
in <i>adr</i>	20	<b>input</b> : Schreibe den Wert des Eingaberegisters an die Adresse <i>adr</i>
out <i>adr</i>	22	<b>output</b> : Schreibe den Wert an der Adresse <i>adr</i> ins Ausgaberegister
out <i>val</i>	23	<b>output</b> : Schreibe den Wert <i>val</i> ins Ausgaberegister
add <i>adr</i>	30	<b>add</b> : Addiere den Wert an der Adresse <i>adr</i> zum Akku
add <i>val</i>	31	<b>add</b> : Addiere den Wert <i>val</i> zum Akku
sub <i>adr</i>	32	<b>subtract</b> : Subtrahiere den Wert an der Adresse <i>adr</i> vom Akku
sub <i>val</i>	33	<b>subtract</b> : Subtrahiere den Wert <i>val</i> vom Akku
mul <i>adr</i>	34	<b>multiply</b> : Multipliziere den Wert an der Adresse <i>adr</i> mit dem Akku
mul <i>val</i>	35	<b>multiply</b> : Multipliziere den Wert <i>val</i> mit dem Akku
div <i>adr</i>	36	<b>divide</b> : Dividiere den Akku durch den Wert an der Adresse <i>adr</i>
div <i>val</i>	37	<b>divide</b> : Dividiere den Akku durch den Wert <i>val</i> ÷ nur ganzzahliger Teil
mod <i>adr</i>	38	<b>modulo</b> : Rest bei Division des Akku durch den Wert an der Adresse <i>adr</i>
mod <i>val</i>	39	<b>modulo</b> : Rest bei Division des Akku durch den Wert <i>val</i>
cmp <i>adr</i>	40	<b>compare</b> : Vergleiche den Akkuinhalt mit dem Wert an der Adresse <i>adr</i>
cmp <i>val</i>	41	<b>compare</b> : Vergleiche den Akkuinhalt mit dem Wert <i>val</i>
jmp <i>tar</i>	50	<b>jump</b> : Springe zum Zielpunkt <i>tar</i> (Zeilennummer oder Marke)
jlt <i>tar</i>	52	<b>jump if less than</b> : Springe ..., wenn bei cmp der Akkuinhalt kleiner war
jeq <i>tar</i>	54	<b>jump if equal</b> : Springe ..., wenn bei cmp der Akkuinhalt gleich war
jgt <i>tar</i>	56	<b>jump if greater than</b> : Springe ..., wenn bei cmp der Akkuinhalt größer war
end	60	<b>end</b> : Beendet ein Programm

- ▶ Jede **Zeile** enthält genau einen Befehl; dieser muss ganz links beginnen.
- ▶ Zwischen Befehl und Operand muss mindestens ein **Leerzeichen** stehen; **Tabulatoren** können als Trennzeichen ebenfalls verwendet werden.
- ▶ **Kommentare** werden durch Semikolon gekennzeichnet. Alles, was rechts von einem Semikolon steht, wird nicht interpretiert.
- ▶ **Adressen** (im VNR-Hauptspeicher) werden durch ein vorangestelltes \$ (Dollarzeichen) gekennzeichnet. Der ansprechbare Adressraum für Daten ist auf die acht Adressen \$64 .. \$71 beschränkt. An Stelle von Adressen können auch die **Variablen** a .. h verwendet werden, die diesen Speicherstellen als fixe Aliase zugeordnet sind.
- ▶ **Sprungziele** können **Zeilennummern** oder selbst definierte **Marken** sein. Zeilennummern werden durch ein vorangestelltes # (Raute) gekennzeichnet, Marken bei ihrer Definition durch einen vorangestellten Doppelpunkt. Bei Verwendung einer Marke als Sprungziel wird nur die Marke (ohne Doppelpunkt) angegeben.
- ▶ Gültige **Namen für Marken** dürfen nur aus Buchstaben (ohne Umlaute und ß) und Ziffern bestehen; das erste Zeichen muss ein Buchstabe sein.
- ▶ Der gesamte Quelltext ist **case-insensitiv**, d.h. Groß- und Kleinschreibung werden nicht unterschieden. Empfohlen wird eine durchgängige Kleinschreibung.
- ▶ Das **Ende** eines jeden Programms wird durch den Befehl end festgelegt.

## Einfache Addition

Schreibe ein Programm, das die Zahlen 19 und 12 addiert und das Ergebnis ins Ausgaberegister schiebt.

Hinweis, wenn Du nicht weiterkommst:

- Lade zunächst die 19 in den Akku(mulator)
- Speichere den Akkuinhalt in der Zelle mit der Adresse \$64
- Lade nun die 12 in den Akku
- Addiere den Inhalt aus Zelle \$64 dazu
- Speichere den (neuen) Akkuinhalt in Zelle \$65
- Gib den Inhalt von \$65 aus

---

## Lösung Einfache Addition

```
ld 19
st $64
ld 12
add $64
st $65
out $65
end
```

Kommentar: Zum Nachvollziehen, wie ein Programm übersetzt wird und wie der MOPS arbeitet.

**EINE ERFOLGTE EINFÜHRUNG IN DIE VON-NEUMANN-ARCHITEKTUR UND IN DAS ARBEITEN MIT DEM MOPS WIRD HIER VORAUSGESETZT!**

### Addition zweier Zahlen

Schreibe ein Programm, das zwei Zahlen einliest, diese addiert und das Ergebnis ins Ausgaberegister schiebt.

Beispiel:    **A = 4, B = 6**

**Ausgabe: 10,**    da  $4 + 6 = 10$

---

### Lösung Addition zweier Zahlen

```
in $64
in $65
ld $64
add $65
st $66
out $66
end
```

### Aufgabe Naive Division

Schreibe ein Programm, das zwei Eingaben erwartet (Dividend und Divisor) und das Ergebnis der ganzzahligen Division im Ausgaberegister ausgibt.

Beispiel:  $A = 9, B = 4$       Ausgabe: 2, da  $9 / 4 = 2$  (ganzzahlig, Rest 1)

Läuft das Programm immer fehlerfrei? Versuche eine Situation zu erzeugen, in der es zu einem schweren Fehler kommt!

---

### Lösung Naive Division

```
in $64
in $65
ld $64
div $65
st $66
out $66
end
```

Kommentar: Bei einer Division durch 0 kommt es zu einem Fehler. Über die Prüfung, ob der Divisor 0 ist, kann das Konzept der Entscheidungsanweisung eingeführt werden.

## Aufgabe Division mit Prüfung

Schreibe ein Programm, das zwei Eingaben erwartet (Dividend und Divisor) und das Ergebnis der ganzzahligen Division im Ausgaberegister ausgibt. Dabei sollen ungültige Eingaben nicht mehr möglich sein, es muss also auf Division durch 0 geprüft werden.

Beispiel: **A = 9, B = 4**

**Ausgabe: 2,** da  $9 / 4 = 2$  (ganzzahlig, Rest 1)

**A = 8, B = 0**

Programm wird beendet

## EXKURS zum Modulo:

Der Modulo ist der ganzzahlige Rest einer ganzzahligen Division, hier also immer der Rest:

$8 / 5 = 1$ , Rest **3**

$8 / 4 = 2$ , Rest **0**

$8 / 3 = 2$ , Rest **2**

also:

**$8 \bmod 5 = 3$**

**$8 \bmod 4 = 0$**

**$8 \bmod 3 = 2$**

usw.

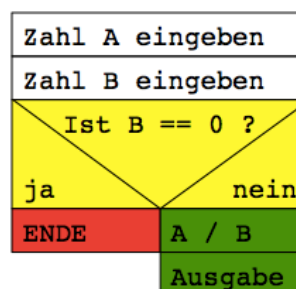
---

## Lösung Division mit Prüfung

```
in $64
in $65
ld $65
cmp 0
jeq M1
ld $64
div $65
st $66
out $66
end :M1
```

Kommentar: Gegenüberstellung Quelltext und Struktogramm:

```
in $64
in $65
ld $65
cmp 0
jeq M1
ld $64
div $65
st $66
out $66
end :M1
```



### Aufgabe Division mit Rest

Schreibe ein Programm, das zwei Eingaben erwartet (Dividend und Divisor) und zuerst das Ergebnis der ganzzahligen Division und danach den Rest der ganzzahligen Division im Ausgaberegister ausgibt. Zudem sollen diese beiden Zahlen (Ergebnis und Rest) in den Speicherzellen \$66 und \$67 gespeichert werden.

Beispiel:    **A = 9, B = 4**

**Ausgabe: 2 ... 1,    da  $9 / 4 = 2$ , Rest 1**

**A = 8, B = 5**

**Ausgabe: 1 ... 3,    da  $8 / 5 = 1$ , Rest 3**

---

### Lösung Division mit Rest

```
in $64
in $65
;Ergebnis
ld $64
div $65
st $66
;Rest
ld $64
mod $65
st $67
out $66
out $67
end
```

Kommentar: Einführungsmöglichkeit bzw. Wiederauffrischung für den Umgang mit dem Modulo.  
Auch mit Prüfung auf Division durch 0 kombinierbar.

### Aufgabe Gerade oder Ungerade

Schreibe ein Programm, das feststellt, ob eine Zahl gerade oder ungerade ist. Das Programm erwartet eine Eingabe X und gibt im Ausgaberegister 0 aus, wenn die Zahl gerade ist und 1, wenn sie ungerade ist.

Beispiel:    **Eingabe: 3**

**Ausgabe: 1**

**Eingabe: 8**

**Ausgabe: 0**

---

### Lösung Gerade oder Ungerade

```
in $64
ld $64
mod 2
st $65
out $65
end
```

Kommentar: Weitere Einführungsmöglichkeit bzw. Wiederauffrischung für den Umgang mit dem Modulo. Programm gibt 1 für ungerade und 0 für gerade aus.

## Aufgabe Signum-Funktion

Ein Programm soll entscheiden, ob eine Zahl negativ, 0 oder positiv ist. Ist die Zahl negativ, soll -1 ausgegeben werden. Bei 0 entsprechend 0 und wenn die Zahl positiv ist, soll 1 ausgegeben werden.

Beispiel:	Eingabe: -65	Ausgabe: -1
	Eingabe: 0	Ausgabe: 0
	Eingabe: 45	Ausgabe: 1

## Lösung Signum-Funktion

in \$64	in \$64
ld \$64	ld \$64
cmp 0	cmp 0
jeq M1	jeq M1
jlt M2	jlt M2
jgt M3	jgt M3
out 0 :M1	out 0 :M1
jmp MEnd	jmp MEnd
out -1 :M2	out -1 :M2
jmp MEnd	jmp MEnd
out 1 :M3	out 1 :M3
end :MEnd	end :MEnd

Zahl x eingeben		
Ist X ...		
==0	<0	>0
0 aus- geben	-1 aus- geben	1 aus- geben
Ende		

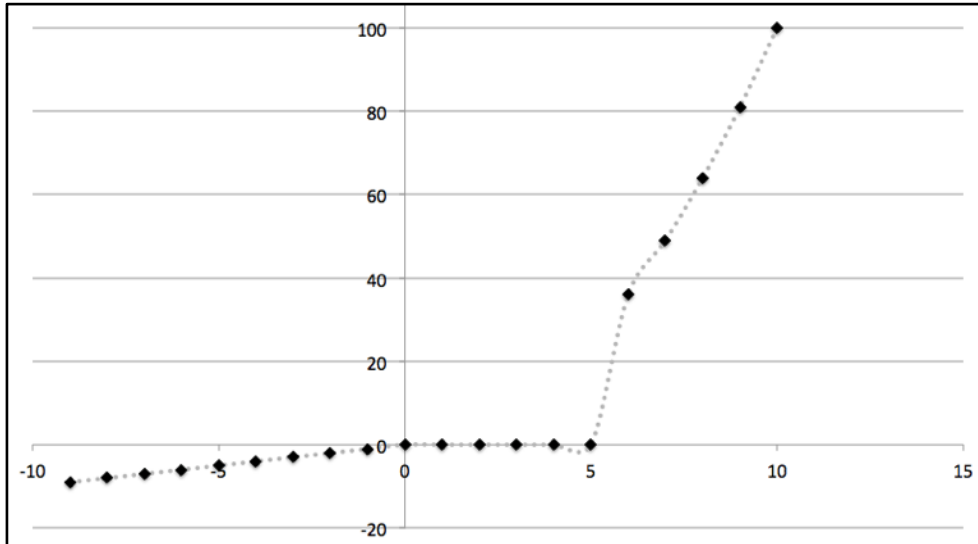
Kommentar: Hier sind auch Alternativen denkbar mit verschachtelten Entscheidungsanweisungen. In der hier verwendeten Version kann auch über eine Fallunterscheidung mit „sonst“-Teil diskutiert werden und über die Konsequenzen, wenn Fälle vergessen werden. Auch kann auf die switch-Anweisung vorgegriffen werden.

## Aufgabe Dreigeteilte Funktion

Schreibe ein Programm, das die folgende Funktion  $f(x)$  umsetzt und für einen eingegebenen Wert  $x$  den entsprechenden  $y$ -Wert ausgibt:

$$f(x) = \begin{cases} x, & \text{für } x < 0 \\ 0, & \text{für } (x \geq 0) \text{ und } (x \leq 5) \\ x^2, & \text{für } x > 5 \end{cases}$$

Der Funktionsgraph im Bereich  $[-10..10]$  sieht dementsprechend wie folgt aus:



## Lösung Dreigeteilte Funktion

```
in $64
ld $64
cmp 5
jgt M1
cmp 0
jlt M2

; sonst
ld 0
st $64
jmp M2

; > 5
mul $64 :M1
st $64
jmp M2

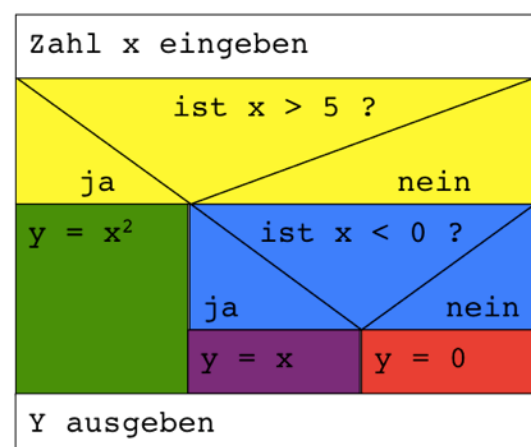
; Ausgabe
out $64 :M2
end
```

```
in $64
ld $64
cmp 5
jgt M1
cmp 0
jlt M2

; sonst
ld 0
st $64
jmp M2

; > 5
mul $64 :M1
st $64
jmp M2

; Ausgabe
out $64 :M2
end
```



## Aufgabe Zähler

Schreibe ein Programm, das von 1 bis 9999 zählt (mehr geht beim MOPS nicht). Die Ausgabe soll im Ausgaberegister erfolgen.

Beispiel: 1 ... 2 ... 3 ... 4 ... usw.

Hier bietet sich dann diese Einstellung an:

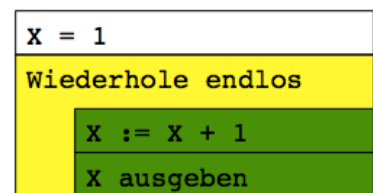
Ablauf  Animation

---

## Lösung Zähler

```
ld 1
add 1 :M1
st $64
out $64
jmp M1
end
```

```
ld 1
add 1 :M1
st $64
out $64
jmp M1
end
```



Kommentar: simple Einführung der Wiederholungsanweisung

## Aufgabe Reihe aufsagen

Schreibe ein Programm, das eine ganze Zahl erwartet. Es soll dann die Zahlenreihe dieser Zahl automatisch immer weiter fortsetzen.

Beispiel:    **Eingabe: 3**

**Ausgabe: 3, 6, 9, 12, 15, 18 , ...**

**Eingabe: 4**

**Ausgabe: 4, 8, 12, 16, 20, ...**

## Lösung Reihe aufsagen

in \$64	; Alternative	ld 0
out \$64	ld 0	st \$65
ld \$64	st \$65	in \$64
add \$64 :M1	in \$64	ld \$65 :M1
st \$65	ld \$65 :M1	add \$64
out \$65	add \$64	st \$65
jmp M1	st \$65	out \$65
end	out \$65	jmp M1
	jmp M1	end
	end	

Erg = 0
Zahl X eingeben
Wiederhole endlos
Erg = Erg + x
Erg ausgeben

## Aufgabe Summe aller Zahlen

Schreibe ein Programm, das die Summe aller Zahlen zwischen einer Startzahl und einer Endzahl berechnet. Dazu sollen zunächst zwei Zahlen ins Eingaberegister gelesen werden: A und B. Danach soll die Summe aller Zahlen von A bis B im Ausgaberegister ausgegeben werden.

Beispiel:    **A = 3, B = 8**

**Ausgabe: 33,**    da  $3 + 4 + 5 + 6 + 7 + 8 = 33$

**A = 12, B = 13**

**Ausgabe: 25,**    da  $12 + 13 = 25$

---

## Lösung Summe aller Zahlen

<pre>ld 0 st \$66 in \$64 ;A in \$65 ;B ld \$65 :M2 cmp \$64 jlt M1 ld \$64 add \$66 st \$66 ld \$64 add 1 st \$64 jmp M2 out \$66 :M1 end</pre>	<pre>ld 0 st \$66 in \$64 ;A in \$65 ;B ld \$65 :M2 cmp \$64 jlt M1 ld \$64 add \$66 st \$66 ld \$64 add 1 st \$64 jmp M2 out \$66 :M1 end</pre>	<table><tr><td>Erg = 0</td></tr><tr><td>Zahl A eingeben</td></tr><tr><td>Zahl B eingeben</td></tr><tr><td>Wiederhole bis B &lt; A</td></tr><tr><td>    Erg = Erg + A</td></tr><tr><td>    A = A + 1</td></tr><tr><td>Erg ausgeben</td></tr></table>	Erg = 0	Zahl A eingeben	Zahl B eingeben	Wiederhole bis B < A	Erg = Erg + A	A = A + 1	Erg ausgeben
Erg = 0									
Zahl A eingeben									
Zahl B eingeben									
Wiederhole bis B < A									
Erg = Erg + A									
A = A + 1									
Erg ausgeben									

Kommentar: Alternativ kann hier die Wiederholungsanweisung zunächst auch als Endlosschleife mit einem Break-Befehl eingeführt werden. Dann sollte jedoch der Begriff „bedingte Schleife“ eingeführt werden.

## Aufgabe Dezimalzahl in Dualzahl

Schreibe ein Programm, das eine Dezimalzahl in eine 4-Bit Dualzahl umwandelt und ziffernweise über das Ausgaberegister ausgibt. Dabei wird das Problem soweit vereinfacht, dass die Dualzahl von rechts nach links ausgegeben werden darf.

Beispiel:    **Eingabe: 13** (1101)

**Ausgabe: 1 ... 0 ... 1 ... 1**

**Eingabe: 8** (1000)

**Ausgabe: 0 ... 0 ... 0 ... 1**

Das geht mit dem nachfolgenden Algorithmus relativ simpel:

Zahl X eingeben
Wiederhole bis X 0 ist
X modulo 2 ausgeben
X durch X/2 ersetzen
ENDE

**Hinweis:** Es reicht hier, dies für die Zahlen von 0 bis 15 zu machen.

---

## Lösung Dezimalzahl in Dualzahl

; Alternative

```
in $64
ld $64 :M1
mod 2
st $65
out $65
ld $64
div 2
st $64
cmp 0
jeq M2
jmp M1
end :M2
```

```
in $64

; KOPF
ld $64 :M1
mod 2
st $65
out $65
ld $64
div 2
st $64
cmp 0
jeq M2
jmp M1
out 0 :M2
; ENDE
```

end

**Kommentar:** Gegenüberstellung Quelltext und Struktogramm:

```
in $64
ld $64 :M1
mod 2
st $65
out $65
ld $64
div 2
st $64
cmp 0
jeq M2
jmp M1
end :M2
```

Zahl X eingeben
Wiederhole bis X 0 ist
X modulo 2 ausgeben
X durch X/2 ersetzen
ENDE

## Aufgabe Multiplikation durch Addition

Alle Grundrechenarten können auf die Addition zurückgeführt werden. So kann man die Subtraktion, die Multiplikation und die Division auch durch (mehrere) Additionen ausdrücken. Es reicht daher tatsächlich, wenn ein Computer addieren kann. Alles andere lässt sich darauf zurückführen.

Statt  $X \cdot Y$  zu rechnen, kann man auch  $X$ -mal  $Y$  aufaddieren:

Beispiel:  $3 \cdot 4 = 12$   
 $= 4 + 4 + 4$  (und zwar genau 3 mal)

$5 \cdot 2 = 10$   
 $= 2 + 2 + 2 + 2 + 2$  (und zwar genau 5 mal)

Schreibe ein Programm, das zwei eingegebene Zahlen multipliziert **ohne** den Befehl **mul** zu benutzen (also nur mit **add**). Das Ergebnis soll im Ausgaberegister ausgegeben werden.

## Lösung Multiplikation durch Addition

```
in $64          in $64
in $65          in $65
ld 0            ld 0
st $66 ;counter st $66 ;counter
ld 0            ld 0
st $67 ;ergebnis st $67 ;ergebnis
ld $66 :M1      ld $66 :M1
cmp $64        cmp $64
jeq M2         jeq M2
ld $67         ld $67
add $65        add $65
st $67         st $67
ld $66         ld $66
add 1          add 1
st $66         st $66
jmp M1         jmp M1
out $67 :M2     out $67 :M2
end            end
```

Zahl A eingeben
Zahl B eingeben
Counter = 0
Erg = 0
Wiederhole bis Counter = A
Erg = Erg + B
Counter = Counter + 1
Erg ausgeben

Kommentar: Einführung der Zählschleife mit Hilfe eines Counters. Als Zählschleife gehören daher alle gelb markierten Zeilen mit zur Wiederholungsanweisung. Evtl. Hinweis, dass eine Zählschleife ebenso nichts anderes ist als eine Endlosschleife mit Zähler und break.

```
ld 0
st $67 ;ergebnis
in $64
in $65
ld 0
st $66 ;counter
ld $66 :M1
cmp $64
jeq M2
ld $67
add $65
st $67
ld $66
add 1
st $66
jmp M1
out $67 :M2
end
```

Erg = 0
Zahl A eingeben
Zahl B eingeben
Wiederhole A mal
Erg = Erg + B
Erg ausgeben

## Aufgabe Potenzen

Schreibe ein Programm, das die Potenz von 2 hoch X berechnet und im Ausgaberegister anzeigt. X soll die Eingabe sein.

Beispiel:     $X = 4$              $2^0$     Ausgabe 1  
                                  $2^1$     Ausgabe 2  
                                  $2^2$     Ausgabe 4  
                                  $2^3$     Ausgabe 8  
                                  $2^4$     **Ausgabe 16**

Hinweis: Die Zahl 2 verdoppelt sich dabei also von Schritt zu Schritt.

---

## Lösung Potenzen

```
in $64
ld 1
st $65
ld $64 :M2
cmp 0
jeq M1
sub 1
st $64
ld $65
mul 2
st $65
jmp M2
out $65 :M1
end
```

```
in $64
ld 1
st $65
ld $64 :M2
cmp 0
jeq M1
sub 1
st $64
ld $65
mul 2
st $65
jmp M2
out $65 :M1
end
```

Erg = 1
Zahl X eingeben
Wiederhole bis X == 0
X = X - 1
Erg = Erg * 2
Erg ausgeben

## Aufgabe Fakultät

Die Fakultät der Zahl X berechnet sich als Produkt aller Zahlen von 1 bis X. Die Fakultät wird mit dem Ausrufezeichen gekennzeichnet;

Beispiel:  $6!$  (sprich: sechs Fakultät) = 720, da  $1 * 2 * 3 * 4 * 5 * 6 = 720$

$4! = 24$ , da  $1 * 2 * 3 * 4 = 24$

Schreibe ein Programm, das die Fakultät einer ins Eingaberegister eingegebenen Zahl berechnet und im Ausgaberegister ausgibt!

---

## Lösung Fakultät

	;Alternative
ld 1	in \$65
st \$65	ld 1
in \$64	st \$64
ld \$64	st \$66
cmp 0 :M2	ld \$66 :Marke1
jeq M1	mul \$64
mul \$65	st \$66
st \$65	ld \$64
ld \$64	cmp \$65
sub 1	jeq Marke2
st \$64	add 1
jmp M2	st \$64
out \$65 :M1	jmp Marke1
end	out \$66 :Marke2
	end

## Aufgabe Suppentemperatur

Eine Suppe, frisch vom Herd gebracht, habe eine Temperatur von 92 Grad Celsius. Die Zimmertemperatur betrage 19 Grad Celsius. In jeder Minute verringert sich die Suppentemperatur um 10% der Differenz zur Zimmertemperatur.

Die Suppentemperatur ist daher über die Zeit gesehen wie folgt:

92, 85, 79, 73, 68, 64, 60, 56, 53, 50, 47, 45, 43, 41, 39, ... usw.

Schreibe ein Programm, das die Suppentemperatur in Minutenabständen angibt. Gestalte das Programm von Anfang an so, dass zwei Werte eingegeben werden können (Suppentemperatur und Raumtemperatur) und dass alle minütlichen Zwischenergebnisse im Ausgaberegister ausgegeben werden.

---

## Lösung Suppentemperatur

```
in $64 ;Temp_Suppe
in $65 ;Temp_Raum
ld $64 :M1
sub $65
div 10
st $66
ld $64
sub $66
st $64
out $64
jmp M1
end
```

## Aufgabe Collatz-Problem

Schreibe ein Programm für ein echtes Problem der Mathematik:

- Beginne mit irgendeiner natürlichen Zahl  $n > 0$ , die der Benutzer eingibt.
- Ist  $n$  gerade, so nimm als nächstes  $n/2$ ,
- Ist  $n$  ungerade, so nimm als nächstes  $3n + 1$ .
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl.

So erhält man zum Beispiel für die Startzahl  $n = 19$  die Folge

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Anscheinend mündet jede Folge mit  $n > 0$  früher oder später in den Zyklus 4, 2, 1, unabhängig davon, welche Startzahl  $n$  man probiert hat. Die Collatz-Vermutung lautet daher „Jede so konstruierte Zahlenfolge mündet in den Zyklus 4, 2, 1, egal, mit welcher natürlichen Zahl  $n > 0$  man beginnt“. Trotz zahlreicher Anstrengungen gehört diese Vermutung noch immer zu den ungelösten Problemen der Mathematik.

Finde die längste Zahlenfolge bei geschickter Startwertwahl!

---

## Lösung Collatz-Problem

```
ld 0
st $65 ; anzahl
in $64
ld $64 :M1
cmp 1
jeq M2
mod 2
cmp 0
jeq JA
ld $64 ; NEIN
mul 3 ; NEIN
add 1 ; NEIN
jmp M3
ld $64 :JA
div 2 ; JA
st $64 :M3
out $64
ld $65
add 1
st $65
jmp M1
end :M2
```

## Aufgabe Fibonacci-Folge

Die Fibonacci-Folge ist die unendliche Folge von natürlichen Zahlen, die mit **0, 1** beginnt und in der die Summe zweier aufeinanderfolgender Zahlen die unmittelbar danach folgende Zahl ergibt:

**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... usw**

Die darin enthaltenen Zahlen heißen Fibonacci-Zahlen. Benannt ist die Folge nach Leonardo Fibonacci, der damit im Jahr 1202 das Wachstum einer Kaninchenpopulation beschrieb.

Man kann also mit 0 und 1 starten und die darauf folgende Zahl durch Addition berechnen. Das geht dann immer so weiter:

**0 + 1 = 1**  
**1 + 1 = 2**  
**1 + 2 = 3**  
**2 + 3 = 5**  
**3 + 5 = 8**

**...usw...**

Schreibe ein Programm, das mit den Startwerten 0 und 1 anfängt und im Ausgaberegister immer automatisch das nächste Folgenmitglied ausgibt.

---

## Lösung Fibonacci-Folge

```
ld 0
st $64
ld 1
st $65
;Endlosschleife
ld $64 :M1
add $65
st $66
ld $65
st $64
ld $66
st $65
out $65
jmp M1
end
```

**Kommentar:** Es bieten sich hier z.B. Referate über den goldenen Schnitt oder Fibonacci-artige Vorkommen in der Natur etc. an.

## Aufgabe Zufallszahlengenerator

Zufallszahlen können im Computer nicht zufällig entstehen. Es muss dafür einen Algorithmus geben, der Zahlen produziert, die scheinbar zufällig aussehen. Ein solcher Algorithmus ist der des linearen Kongruenzgenerators, der wirklich in einigen Programmiersprachen zur Erzeugung zufälliger Zahlen zum Einsatz kommt:

Man wählt möglichst geschickte ganzzahlige Werte für die folgenden Variablen (hier ist Ausprobieren oder Nachdenken angesagt):

- $m$  (z.B. 9)
- $a$  (z.B. 4)
- $c$  (z.B. 1)

Dann startet man, indem man für die Zahl  $X$  zum Zeitpunkt  $n$  (also für  $X_n$ ) einen Startwert festlegt. (z.B. 0)

Mit der Formel:

$$X_{n+1} = (aX_n + c) \bmod m$$

kann nun der nächste darauffolgende Wert  $X_{n+1}$  berechnet werden.

Beispiel:  $m = 9, a = 4, c = 1, X_0 = 0$

$(4 \cdot 0 + 1) \bmod 9 = 1$   
 $(4 \cdot 1 + 1) \bmod 9 = 5$   
 $(4 \cdot 5 + 1) \bmod 9 = 3$   
 $(4 \cdot 3 + 1) \bmod 9 = 4$   
 $(4 \cdot 4 + 1) \bmod 9 = 8$   
 $(4 \cdot 8 + 1) \bmod 9 = 6$   
 $(4 \cdot 6 + 1) \bmod 9 = 7$   
 $(4 \cdot 7 + 1) \bmod 9 = 2$   
 $(4 \cdot 2 + 1) \bmod 9 = 0$

$(4 \cdot 0 + 1) \bmod 9 = 1$

jetzt geht es wieder von vorne los...

### EXKURS zum Modulo:

Der Modulo ist der ganzzahlige Rest einer ganzzahligen Division, hier also immer der Rest:

$8 / 5 = 1, \text{ Rest } 3$   
 $8 / 4 = 2, \text{ Rest } 0$   
 $8 / 3 = 2, \text{ Rest } 2$

also:

$8 \bmod 5 = 3$   
 $8 \bmod 4 = 0$   
 $8 \bmod 3 = 2$

usw.

- Schreibe nun ein Programm, das mit den o.g. Werten Zufallszahlen nacheinander ins Ausgaberegister schreibt und ausgibt.
  - Probiere danach andere Werte für  $m$ ,  $a$  und  $c$  aus. Findest Du „bessere“ oder vielleicht auch „schlechtere“ Werte, die bessere oder schlechtere „Zufallszahlen“ erzeugen?
  - Diskutiere danach, inwiefern sich Zufallszahlen überhaupt mit dem Computer erzeugen lassen.
  - Entwickle eine Idee, wie man echte Zufallszahlen auf den Computer bekommen könnte.
  - Schaue erst danach auf [www.random.org](http://www.random.org) vorbei.
-

## Lösung Zufallszahlengenerator

```
ld 9 ;m
st $64
ld 4 ;a
st $65
ld 1 ;c
st $66
ld 0
st $67
;start prng
ld $67 :M1
mul $65
add $66
mod $64
st $67
out $67
jmp M1
end
```

Kommentar: Zahlreiche Anknüpfungspunkte zum Themenkomplex Informatik, Mensch und Gesellschaft, da Computer per se keine Zufallszahlen erzeugen können.

## Aufgabe Restfrei durch 1 bis 9

Schreibe ein Programm, um die kleinste Zahl zu ermitteln, die ohne Rest (!) gleichzeitig durch 2, 3, 4, 5, 6, 7, 8 und 9 teilbar ist. Die gefundene Zahl soll dann im Ausgaberegister ausgegeben werden.

**Hinweis:** Das fertige Programm kann durchaus eine halbe Stunde laufen bis die Zahl gefunden wurde.

---

## Lösung Restfrei durch 1 bis 9

ld 1	; Alternative
st \$64 ; zahl	
	ld 1
jmp FResCount :Sch1	st \$64
ld \$64 :Sch2	ld \$64 :M1
mod \$65	
cmp 0	mod 8
jeq FIncCount	cmp 0
jlt FIncNum	jgt M2
jgt FIncNum	
	ld \$64
;counter auf 1	mod 9
ld 1 :FResCount	cmp 0
st \$65	jgt M2
jmp Sch2	
	ld \$64
;counter erhöhen	mod 6
ld \$65 :FIncCount	cmp 0
add 1	jgt M2
cmp 10	
jeq FOut	ld \$64
st \$65	mod 7
jmp Sch2	cmp 0
	jgt M2
;zahl erhöhen	
ld \$64 :FIncNum	ld \$64
add 1	mod 5
st \$64	cmp 0
jmp Sch1	jgt M2
	jmp M3
;ergebnis ausgeben	
out \$64 :FOut	ld \$64 :M2
end	add 1
	st \$64
	jmp M1
	ld \$64 :M3
	out \$64
	end

**Kommentar:** Die kleinste Zahl, die restfrei durch 1 bis 9 teilbar ist, ist **2520**.

Links eine Lösung mit ineinander geschachtelten Schleifen, rechts eine Lösung mit einfachem Testen (es werden nur notwendige Divisoren überprüft, daher kommt man mit dem Speicher aus).

## Aufgabe Quadrierte und Summiere

Wenn man die Ziffern einer ganzen mehrstelligen Zahl einzeln aufteilt, dann einzeln quadriert und dann zusammenaddiert, erhält man eine neue Zahl. Wiederholt man das mehrfach, bekommt man eine Reihe, die irgendwann bei 1 oder bei 89 in eine Endlosschleife mündet.

Beispiel:    44      ( $4^2 + 4^2 = 32$ )  
             32      ( $3^2 + 2^2 = 13$ )  
             13      ( $1^2 + 3^2 = 10$ )  
             10      ( $1^2 + 0^2 = 1$ )  
             1       ( $1^2 = 1$ ) usw. usw.

85, 89, 145, 42, 20, 4, 16, 37, 58, 89

Schreibe ein Programm, das eine beliebige Zahl als Eingabe erwartet und in Adresse \$64 immer automatisch das nächste Folgenglied ausgibt.

Fertige eine Tabelle an für die Zahlen von 1 bis 20 und bestimme, ob diese in eine 89-Schleife oder in eine 1-Schleife münden.

**Tipp:** Du wirst fast den ganzen Speicher brauchen. Denke auch daran, dass die Zwischenergebnisse bis zu vier Stellen haben können.

---

## Lösung Quadrierte und summiere

```
in $64
ld $64 :M1
```

```
div 1000
st $68
mul $68
st $68
```

```
ld $64
mod 1000
div 100
st $69
mul $69
st $69
```

```
ld $64
mod 1000
mod 100
div 10
st $70
mul $70
st $70
```

```
ld $64
mod 1000
mod 100
mod 10
st $71
mul $71
st $71
```

```
add $70
add $69
add $68
st $64
jmp M1
```

```
end
```