

Lektion 1

Computerarchitektur

Übersicht 2M, Maschinensprache, Assemblersprache, Little Man Computer

Lektion 1

Computerarchitektur

Ziele, CPU, Maschinensprache, Assemblersprache, LMC, Befehlssatz

Ziele

- Verstehen, was Grundfunktionsweise von Computer ist.
- Zusammenspiel von Software und Hardware verstehen.
- **Hauptziel: Verstehen, was beim Programmieren im Hintergrund passiert.**
- Code in Assemblersprache schreiben können.

Hardware von Computer

- Was sind die drei wichtige **Bauteile**, die in *jedem* Computer vorkommen?



1



Permanentspeicher (Festplatte, SSD)

- Speichern von Filmen, Doks, ...
- Installierte Programme
- Nach Herunterfahren noch vorhanden

2



Arbeitsspeicher (RAM)

- Temporäre Speicherung
- Daten, die Programme im Moment benötigen
- Gelöscht nach Herunterfahren

3

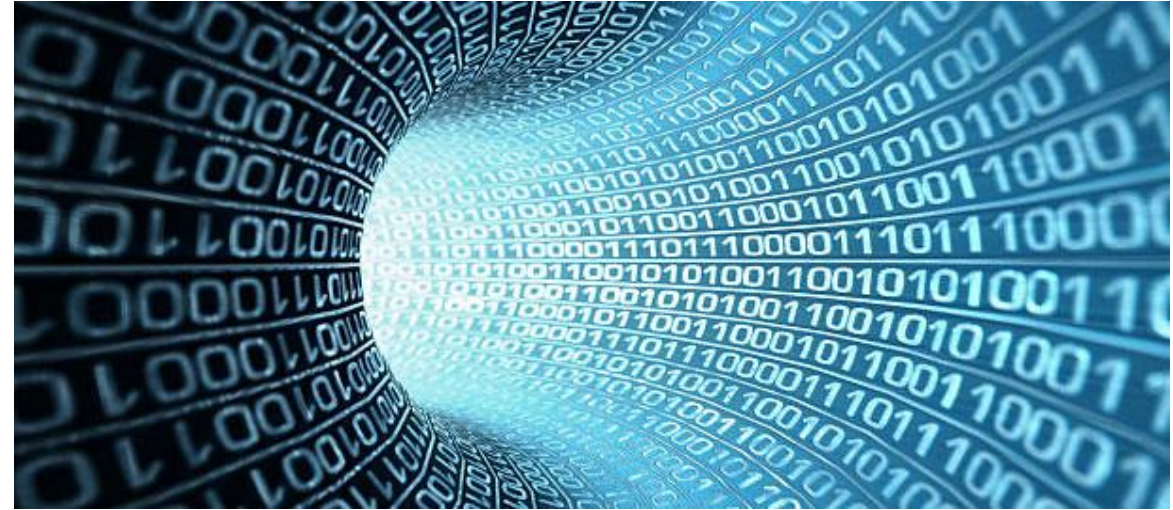


Prozessor (CPU)

- «Gehirn» des Computers
- Rechnet, führt Befehle aus

CPU

- Was CPU macht: **Befehle** ausführen
- CPU **kennt nur Binärzahlen** (0 & 1)



- Bedenke: CPU ist elektronisches Bauteil mit vielen Leitern, durch die (kein) Strom fließen kann
- Befehle, die CPU ausführen soll, müssen also in Sprache sein, die CPU versteht
- Beispiel: 10010100111011010010100101001101010010110101001...
- Diese Sprache heisst **Maschinensprache**
- **Verschiedene CPUs** haben verschiedene Maschinensprachen (aber alle rein binär)



Sprache der CPU

- *Wie können wir nun Programmieren?*
- **Variante 1:** Code *direkt in Maschinensprache*, also in 0 & 1, schreiben!



- **Variante 2:** *Programmiersprache* wie Python wählen



- **Variante 1b:** Programmieren in Assemblersprache (später mehr)



Programmiersprachen & CPU

- Problem: CPU versteht *kein* Python (oder Java, C#, ...)

```
age = int(input("Gib dein Alter ein"))  
  
if age >= 18:  
    print("Schnaps!")  
elif age >= 16:  
    print("Bier!")  
else:  
    print("Sirup!")
```

Python



- Wenn wir also mit Python programmieren, muss Python-Code *zuerst in Maschinensprache übersetzt* werden

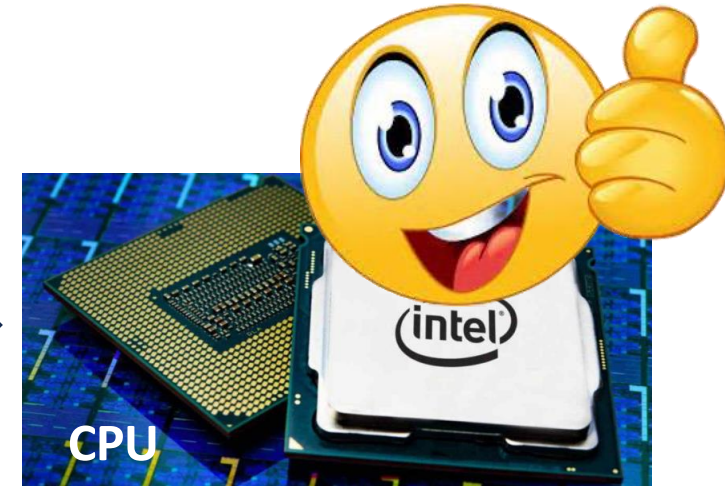
```
age = int(input("Gib dein Alter ein"))  
  
if age >= 18:  
    print("Schnaps!")  
elif age >= 16:  
    print("Bier!")  
else:  
    print("Sirup!")
```

Python



Maschinensprache

```
0110011110111101111011110  
1001101000010110110101000  
1001101110011011110001100  
0001001101100111011010111  
0010010111110101100000100  
0010110010011101101101111  
10111111101
```



CPU

What people think I do.

```
age = int(input("Gib dein Alter ein"))

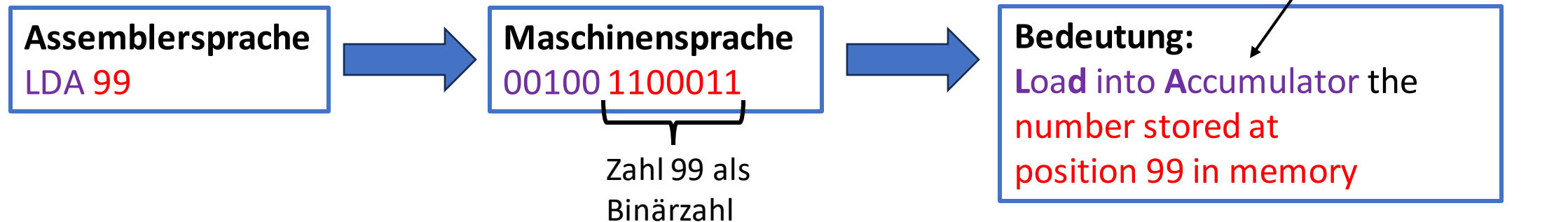
if age >= 18:
    print("Schnaps!")
elif age >= 16:
    print("Bier!")
else:
    print("Sirup!")
```

What I actually do.

```
101001101000100010011100001110000000011100011111
011100111010100000111011000010100011111001010110
000100110001100110011000110001010010011011000001
100011111110001010001000011010111100110011101000
000011011001110101011010001001000010111101010100
111001011110011100001000111100000101001000111000
010101010101111101010000010000010001001100101100
001111001011000110000011010000010101111011101010
111010101111000001110101101010101000000100001111
100011001100000110001010111111110101010011001101
1111001000011101101111100101001110010111100101000
1100011111110000110001000000011010111101110101000
010011011000011001001101000100110010101101011101
000010000110011101000011110110000001001100010100
010011001110011110101011110010010000111001010110
100111100011010011011001101111111110100010101110
101000101111100100100101011101100011000110101001
010110100101011010000011000110011110011110011101
100100111100101010101100100001111010110011110011
```

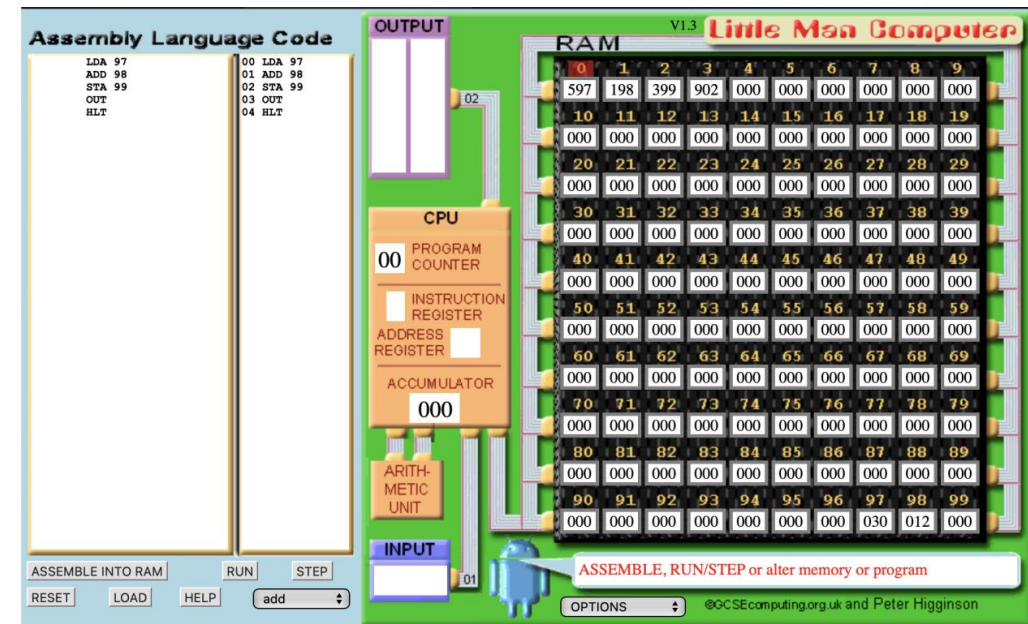
Assemblercode

- In Maschinensprache (0 & 1) programmieren: mühsam
- Ausweg: **Assemblersprache**
- ... ist Maschinensprache direkt übersetzt in etwas einigermaßen Lesbares
- Beispiel:



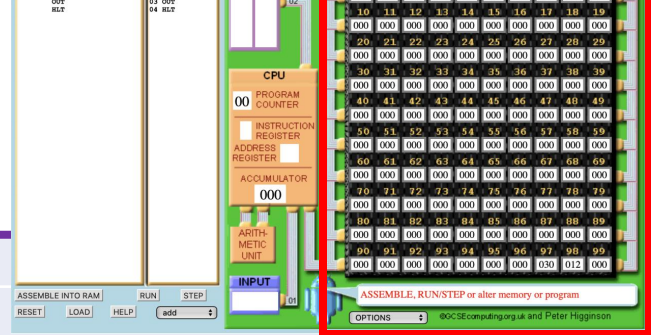
Little Man Computer (LMC)

- CPU-Typen haben eigene Maschinensprache ...
- ... und damit eigenen Assemblersprache
- Wir arbeiten mit **Little Man Computer**
 - Website, die einfache CPU simuliert
 - Schöne Visualisierung der Vorgänge
- Befehle (Instructions) die eine CPU versteht, werden in **Befehlssatz (instruction set)** angegeben



Befehlssatz für LMC

Code (dec)	Code (bin)	Name	Description
0	00000	HLT	Stop (Little Man has a rest).
1	00001	ADD	Add the contents of the memory address to the Accumulator
2	00010	SUB	Subtract the contents of the memory address from the Accumulator
3	00011	STA or STO	Store the value in the Accumulator in the memory address given.
4	00100		This code is unused and gives an error.
5	00101	LDA	Load the Accumulator with the contents of the memory address given
6	00110	BRA	Branch - use the address given as the address of the next instruction
7	00111	BRZ	Branch to the address given if the Accumulator is zero
8	01000	BRP	Branch to the address given if the Accumulator is zero or positive
9	01001	INP or OUT	Input or Output. Take from Input if address is 1, copy to Output if address is 2.
9	01001	OTC	Output accumulator as a character if address is 22. (Non-standard instruction)
		DAT	Used to indicate a location that contains data.



- Zeile Maschinensprache besteht also aus:

Instruktion (5 Bit) + Speicheradresse (7Bit)
}
 12 Bit

- Im LMC werden Befehle aber im Dezimalsystem angezeigt
-> einfacher!

Aufgaben

- Siehe Wiki
- Viele Aufgaben sind didaktisch aufgebaut:
 - Aufgabe mit mehreren Teilen
 - Nach jedem direkt Lösung, z.T. mit zusätzlicher Theorie
 - Wichtig:
 - Zuerst selbst lösen (nicht nur kurz darüber nachdenken)
 - Dann Lösung der LP genau studieren

Lektion 2

Computerarchitektur

Erste Computer, Von Neumann-Architektur

Warm-Up

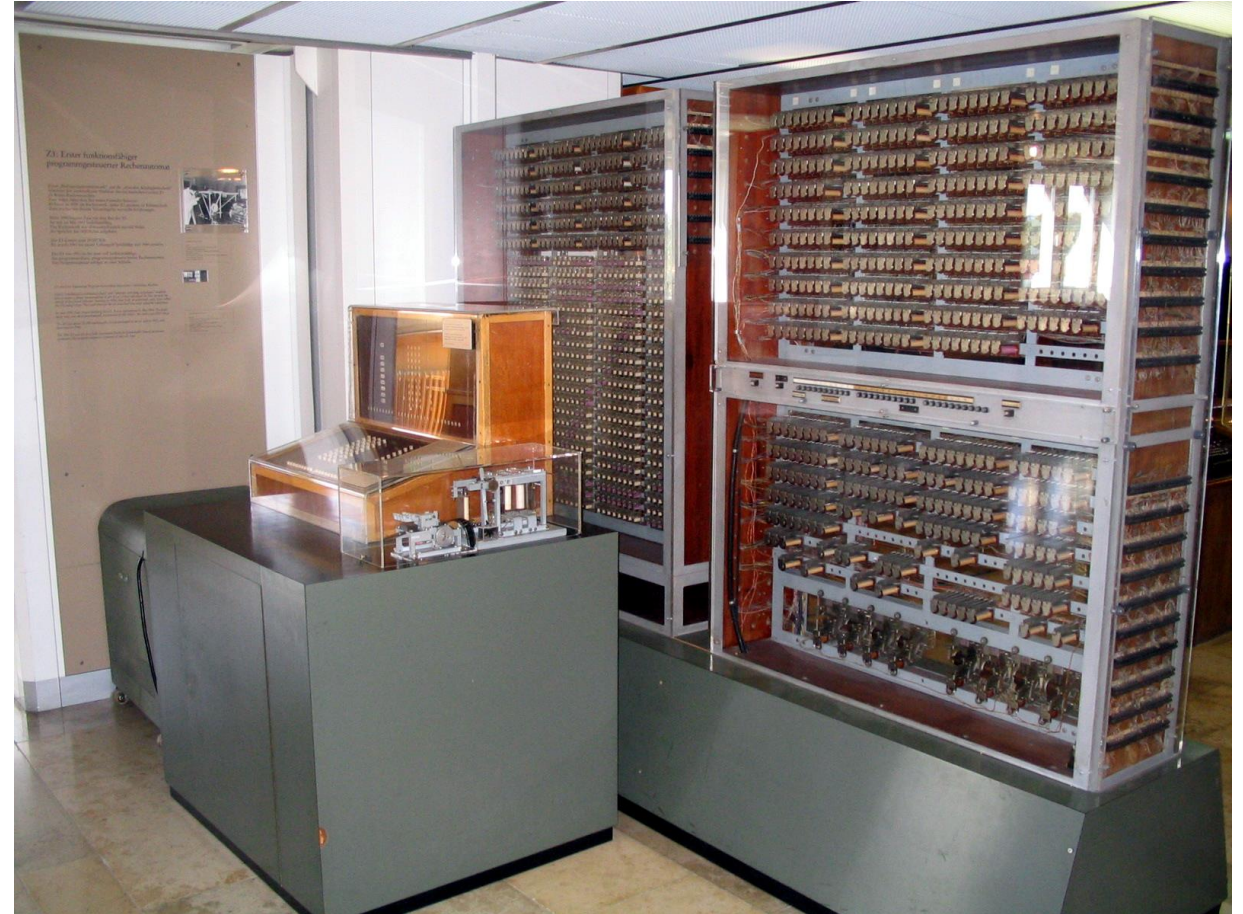
1. «Mit dem Befehl 'ADD 18' addiert man die Zahl 18 zum aktuellen Wert im Akkumulator.»
2. Um zwei Zahlen zu multiplizieren, verwendet man den Befehl 'MUL'.

Erste Computer

- In den 1940er Jahren
- Wollen zwei kurz anschauen:
 - **Zuse Z3**
 - **ENIAC**

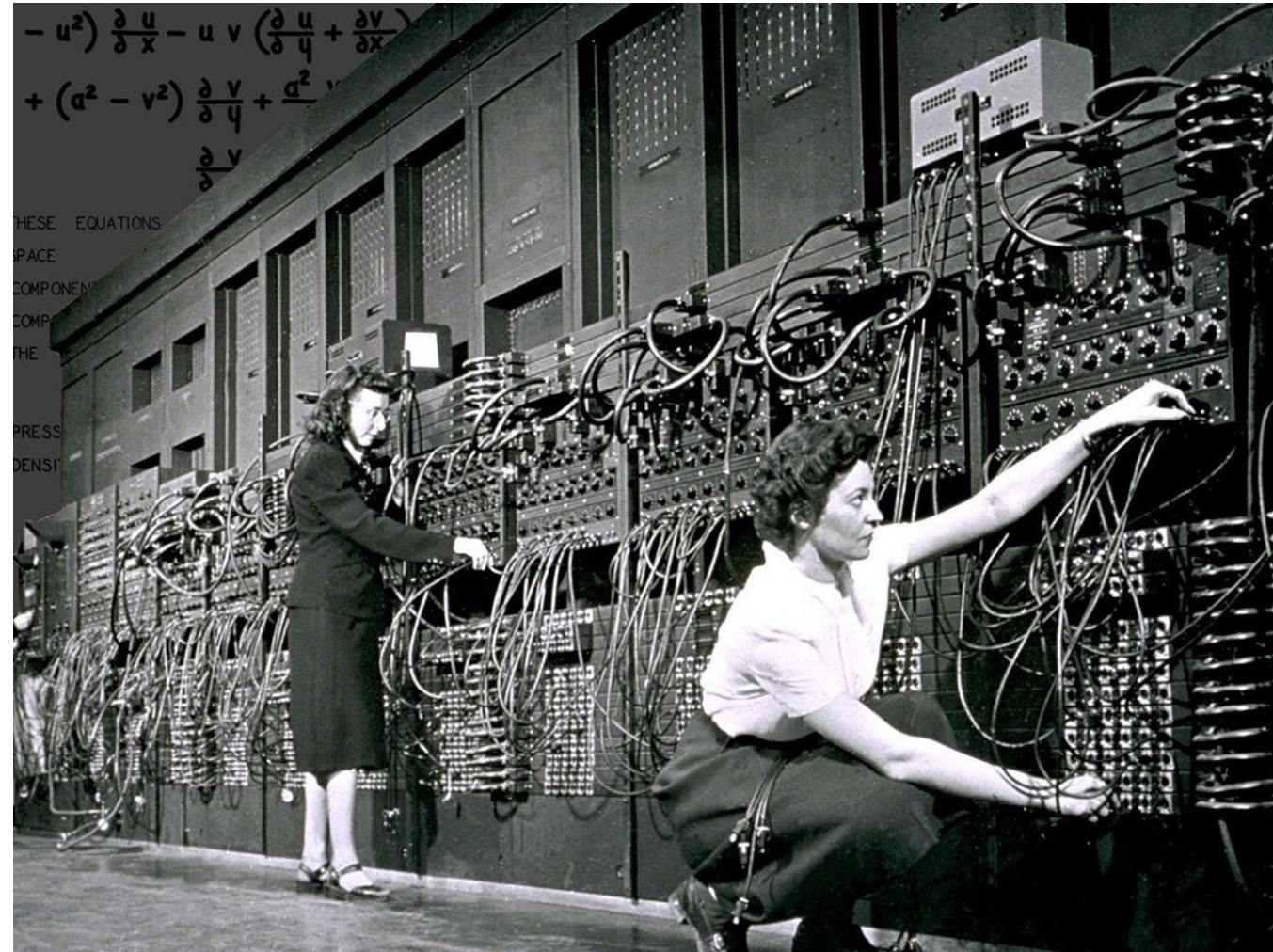
Zuse Z3

- Deutschland
- Vorstellung 1941
- 1943 bei Bombenangriff zerstört (Bild: Nachbau)
- Arbeitete im Binärsystem
- Computer wurde nicht als sehr wichtig betrachtet
- -> Kam nie zu Routinebetrieb



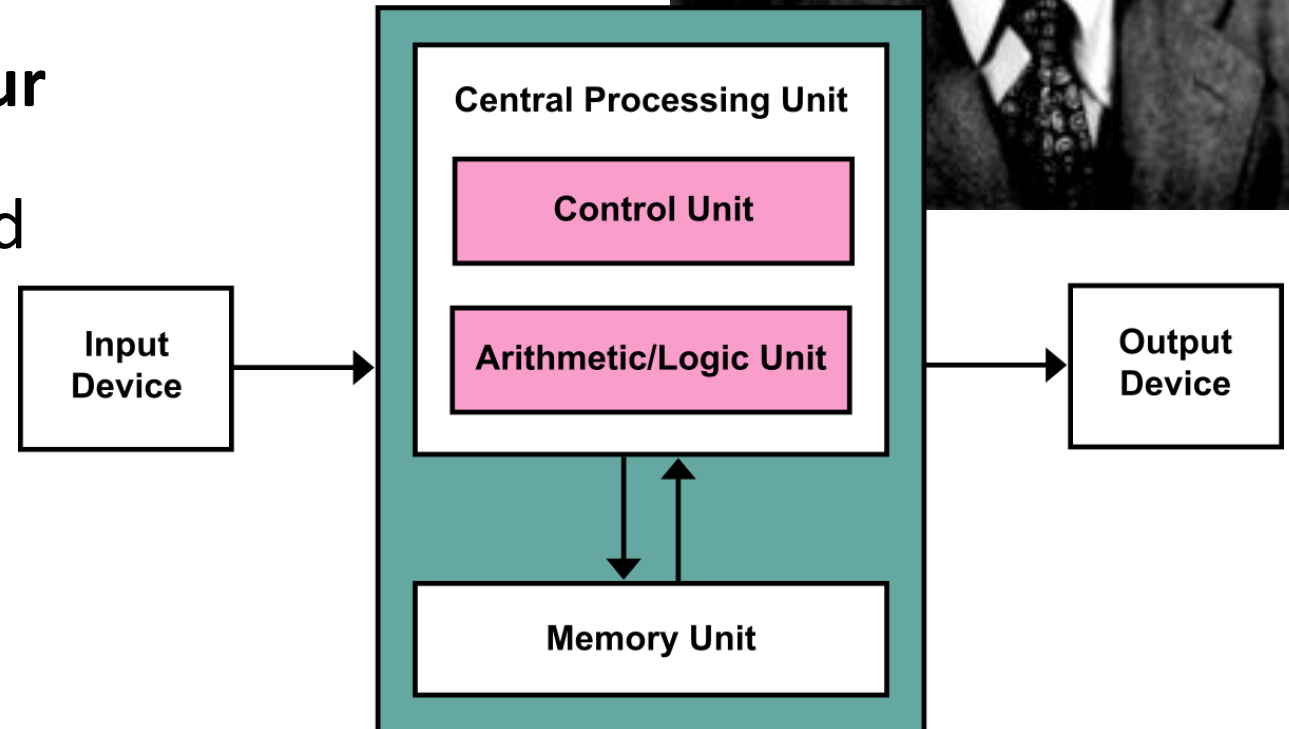
ENIAC

- Im Auftrag des US-Militärs gebaut
- Vorstellung 1946
- Maschinencode wurde direkt mit Kabelverbindungen geschrieben
- Wurde hauptsächlich von Frauen bedient («ENIAC Frauen»)
- Zweck: Berechnungen für Flugbahn von Artillerie



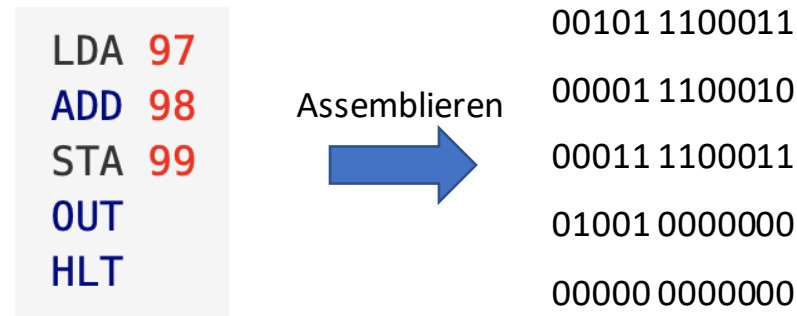
Von-Neumann-Architektur

- Grundidee zum Aufbau von Computer stammt von **John von Neumann** aus Jahr **1945**
- Seine **Von-Neumann-Architektur** ist ein einfaches **Modell** davon, wie Computer *aufgebaut* ist und *funktioniert*
- Moderne Computer basieren (mehrheitlich) auf Von-Neumann-Architektur



Von-Neumann Fun Fact

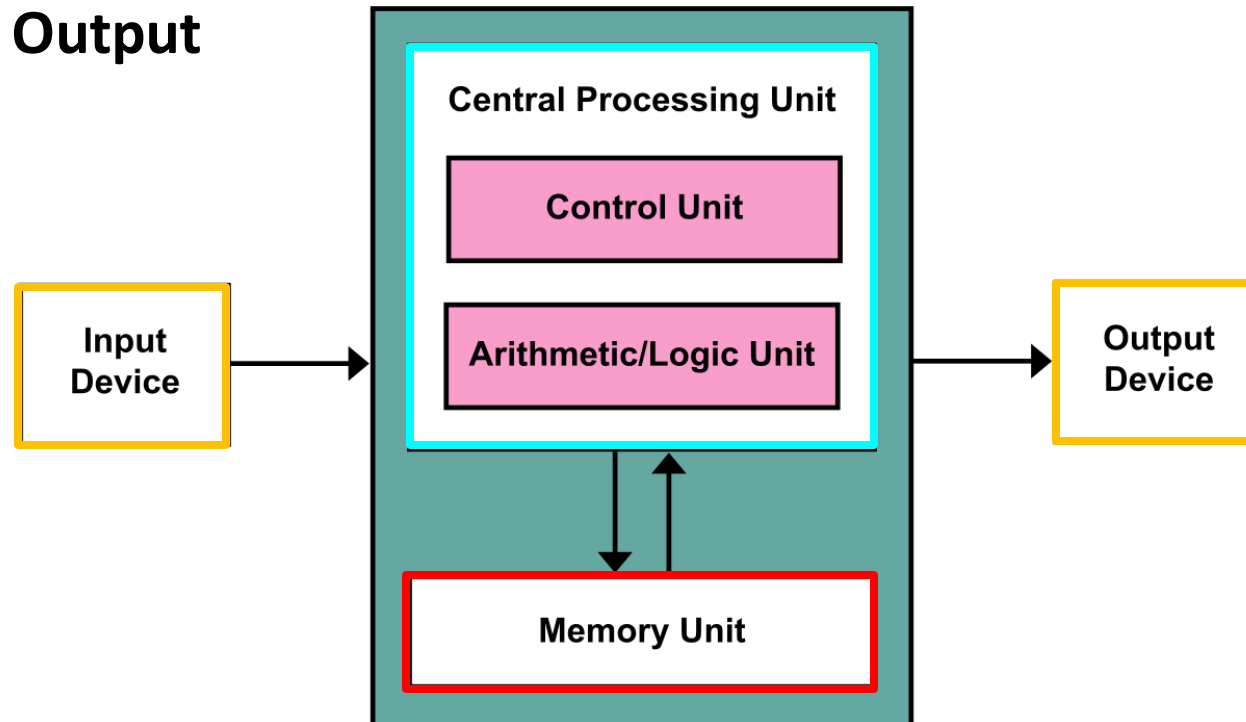
- Assemblieren: das Übersetzen von Assembly Code in Maschinencode.



- Auftrag von John von-Neumann an seine Student:innen:
 - Assemblercode von Hand in Maschinensprache umschreiben.
 - Um der langweiligen Arbeit zu entgehen, schrieben sie ein Programm, die das automatisierte.
 - Als von-Neumann das herausfand, war er verärgert, weil wertvolle Computerzeit für Büroarbeit verschwendet wurde, die geradesogut Menschen machen könnten...

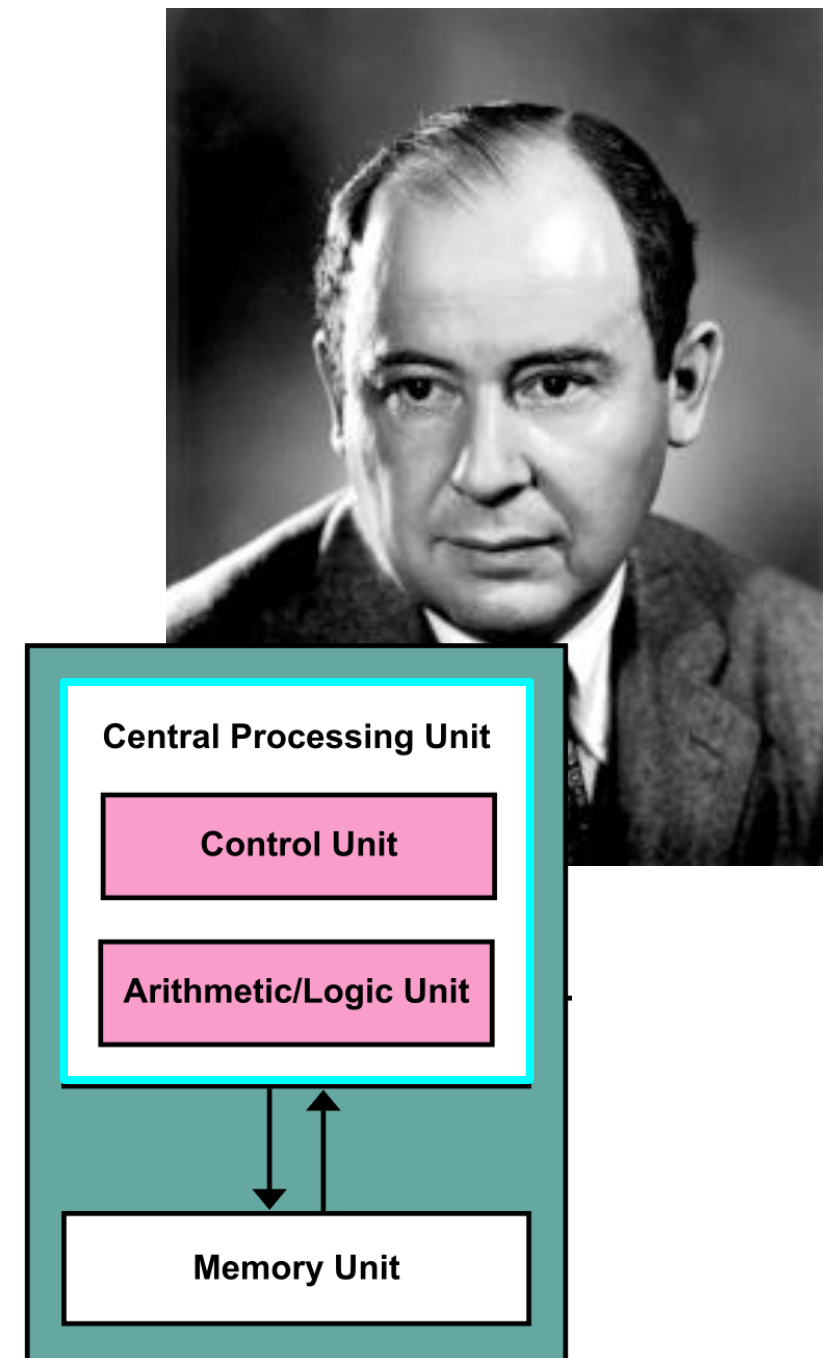
Von-Neumann-Architektur

- Computer besteht aus **drei Hauptkomponenten**:
 - **Prozessor: CPU**
 - **Speicher: Memory Unit (RAM)**
 - **Input / Output**



Von-Neumann-Architektur

- Zwei Hauptbestandteile der CPU:
 - 1. Rechenwerk** (Arithmetic/Logic Unit, ALU):
 - kann Rechnen
 - z.B. zwei Zahlen addieren
 - 2. Steuerwerk** (Control Unit):
 - «Dirigentin» der CPU
 - Stellt sicher, dass korrekte Schritte in richtiger Reihenfolge ausgeführt werden
 - Führt Von-Neumann-Instruktionszyklus aus (später mehr dazu)



Central Processing Unit (CPU)

Steuerwerk / Control Unit

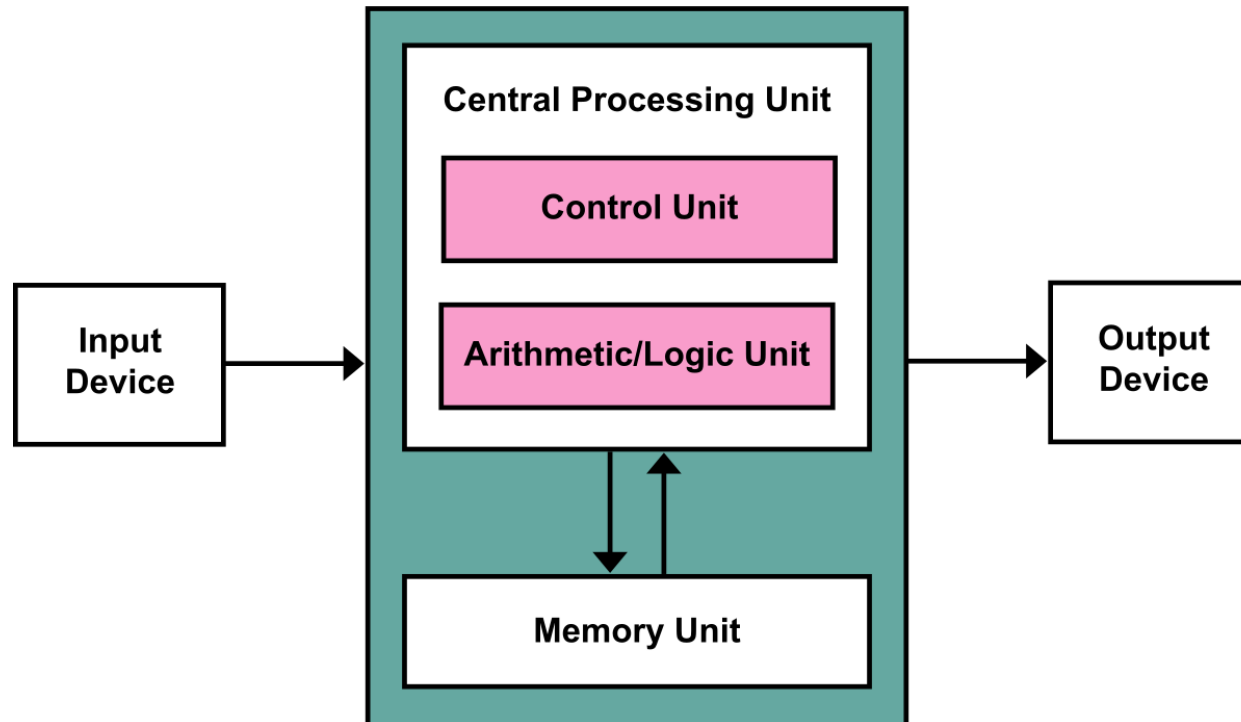


Rechenwerk / ALU



Von-Neumann-Architektur

- Little Man Computer basiert auch auf Von-Neumann-Architektur



CPU

Steuerwerk / Control Unit

Program Counter:

merkt sich Adresse im Speicher von nächster Anweisung



Instruction Register:

Merkt sich erste Stelle der aktuellen Anweisung

Address Register:

Merkt sich 2. und 3. Stelle

Arithmetic Unit:

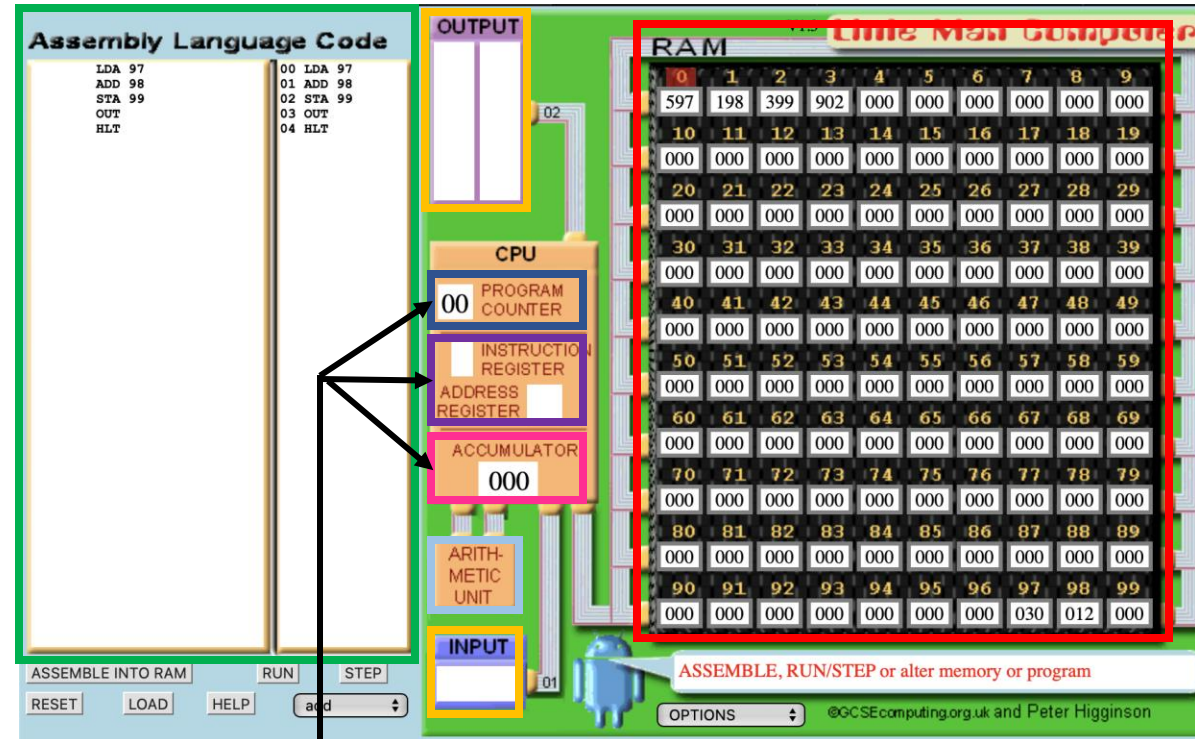
Rechenwerk, stellt Rechnungen an



Akkumulator:

Speichert Resultat der letzten Operation / Berechnung

Little Man Computer



Register: kleiner Speicher

Programmcode in Assemblersprache:

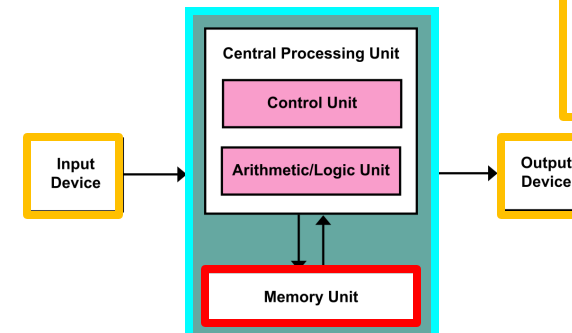
- Jede Zeile beinhaltet eine Anweisung
- Werden am Anfang in Speicher geschrieben

Speicher / Memory / RAM:

- Speichert Programmcode UND Daten
- Hier: 100 Speicheradressen (0-99)

Input / Output:

Werte eingeben und ausgeben lassen



Auftrag

Siehe Wiki

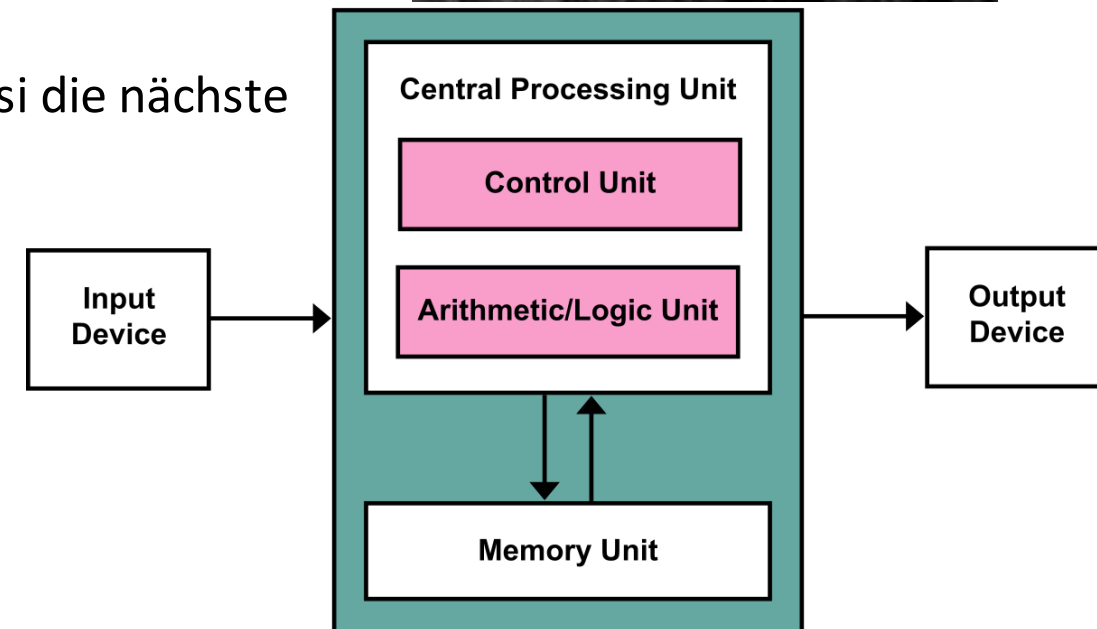
Lektion 3

Computerarchitektur

Von-Neumann-Instruktionszyklus

Von-Neumann-Architektur

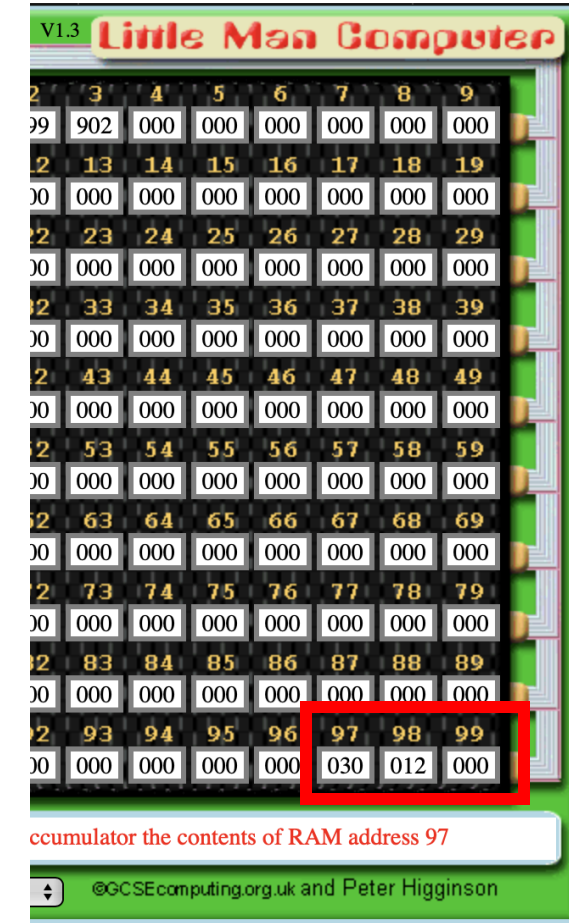
- Maschinensprache-Code wird im Memory abgelegt
- ... und von CPU *Zeile für Zeile* durchgegangen
- Für *jede Zeile* (also jede Instruktion) wird gesamter **Von-Neumann-Instruktionszyklus** durchgeführt:
 1. **Fetch** (holen gehen):
 - Hole im Speicher die nächste Instruktion – quasi die nächste Zeile Code
 2. **Decode** (decodieren):
 - Entschlüssele, was diese bedeutet.
 3. **Execute** (ausführen):
 - Führe sie aus.



Von-Neumann-Instruktionszyklus

- Wollen genau anschauen für folgendes Programm:
 - An Speicheradressen 97 & 98 sind zwei Zahlen gespeichert
 - Addiere diese beiden Zahlen.
 - Schreibe Resultat an Speicheradresse 99 und gebe es aus.
- Auftrag: Do it (am Computer)! 2min!
- Lösung:

Assembly Language Code	
LDA 97	00 LDA 97
ADD 98	01 ADD 98
STA 99	02 STA 99
OUT	03 OUT
HLT	04 HLT



Von-Neumann-Instruktionszyklus

- Zuerst muss **Programm in RAM gespeichert** werden:
 - Erste Zeile ('LDA 97') an Speicherposition 0
 - Zweite Zeile ('ADD 98') an Speicherposition 1
 - ...
- In welchem Format werden Befehle gespeichert? Im Speicher können nur Zahlen gespeichert werden.
- -> verwende Instruction Set



The image shows a screenshot of the 'Little Man' computer simulation interface, specifically the RAM (Random Access Memory) section. The title bar at the top indicates 'V1.4a Little Man'. Below the title, the word 'RAM' is displayed. The memory is organized into a grid of 27 columns (addresses 0 to 26) and 4 rows. Each cell in the grid contains a 3-digit hexadecimal value, currently all set to '000'. The column headers are labeled with addresses: 0, 1, 2, 3, 4, 5, 6 in the first row; 10, 11, 12, 13, 14, 15, 16 in the second row; and 20, 21, 22, 23, 24, 25, 26 in the third row. The fourth row does not have a header label.

0	1	2	3	4	5	6
000	000	000	000	000	000	000
10	11	12	13	14	15	16
000	000	000	000	000	000	000
20	21	22	23	24	25	26
000	000	000	000	000	000	000

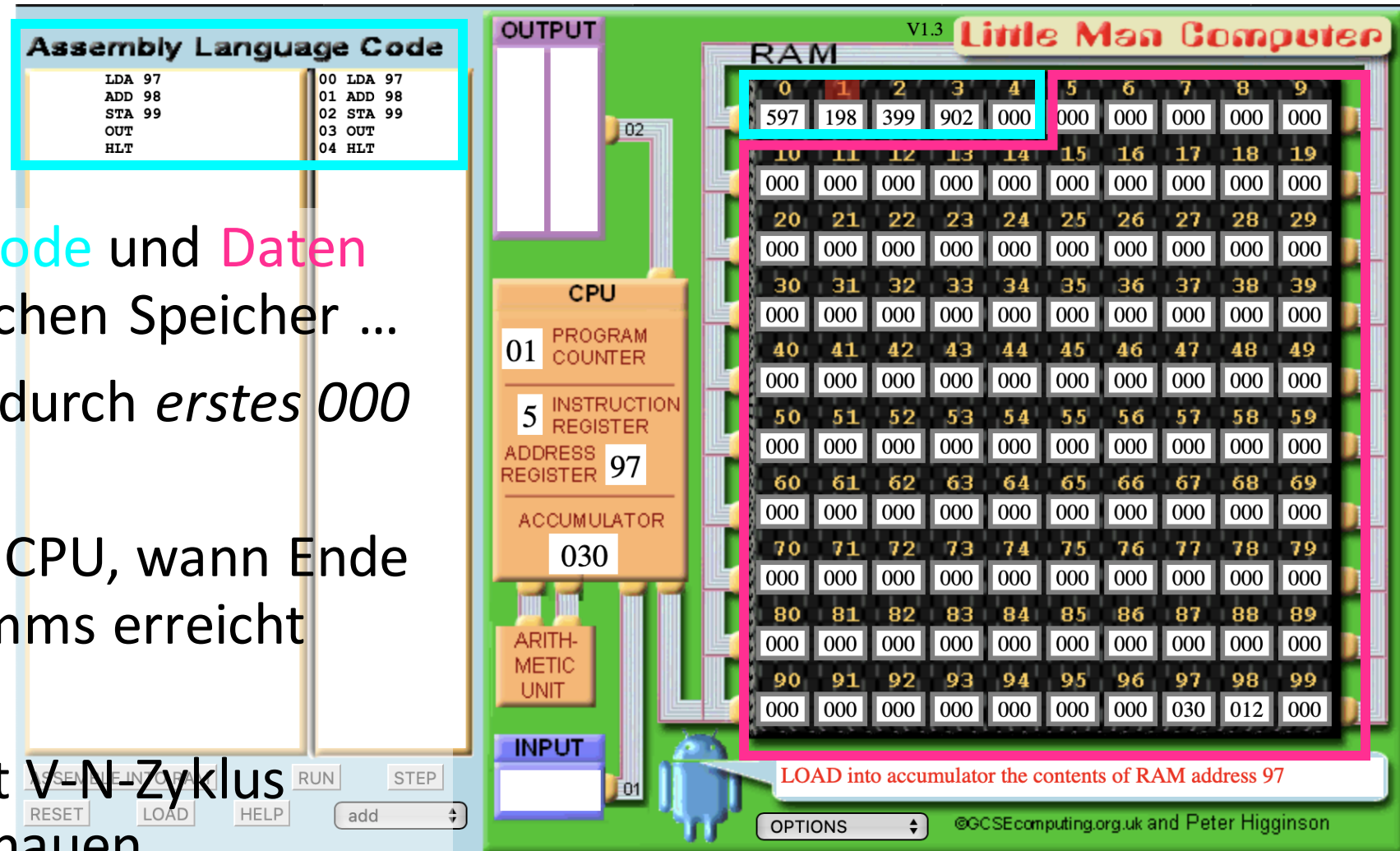
Von-Neumann-Instruktionszyklus

Code	Name	Description
0	HLT	Stop (Little Man has a rest).
1	ADD	Add the contents of the memory address to the Accumulator
2	SUB	Subtract the contents of the memory address from the Accumulator
3	STA or STO	Store the value in the Accumulator in the memory address given.
4		This code is unused and gives an error.
5	LDA	Load the Accumulator with the contents of the memory address given
6	BRA	Branch - use the address given as the address of the next instruction
7	BRZ	Branch to the address given if the Accumulator is zero
8	BRP	Branch to the address given if the Accumulator is zero or positive
9	INP or OUT	Input or Output. Take from Input if address is 1, copy to Output if address is 2.
9	OTC	Output accumulator as a character if address is 22. (Non-standard instruction)
	DAT	Used to indicate a location that contains data.

LDA 97	00	LDA 97	→	597
ADD 98	01	ADD 98	→	198
STA 99	02	STA 99	→	399
OUT	03	OUT	→	902
HLT	04	HLT	→	000

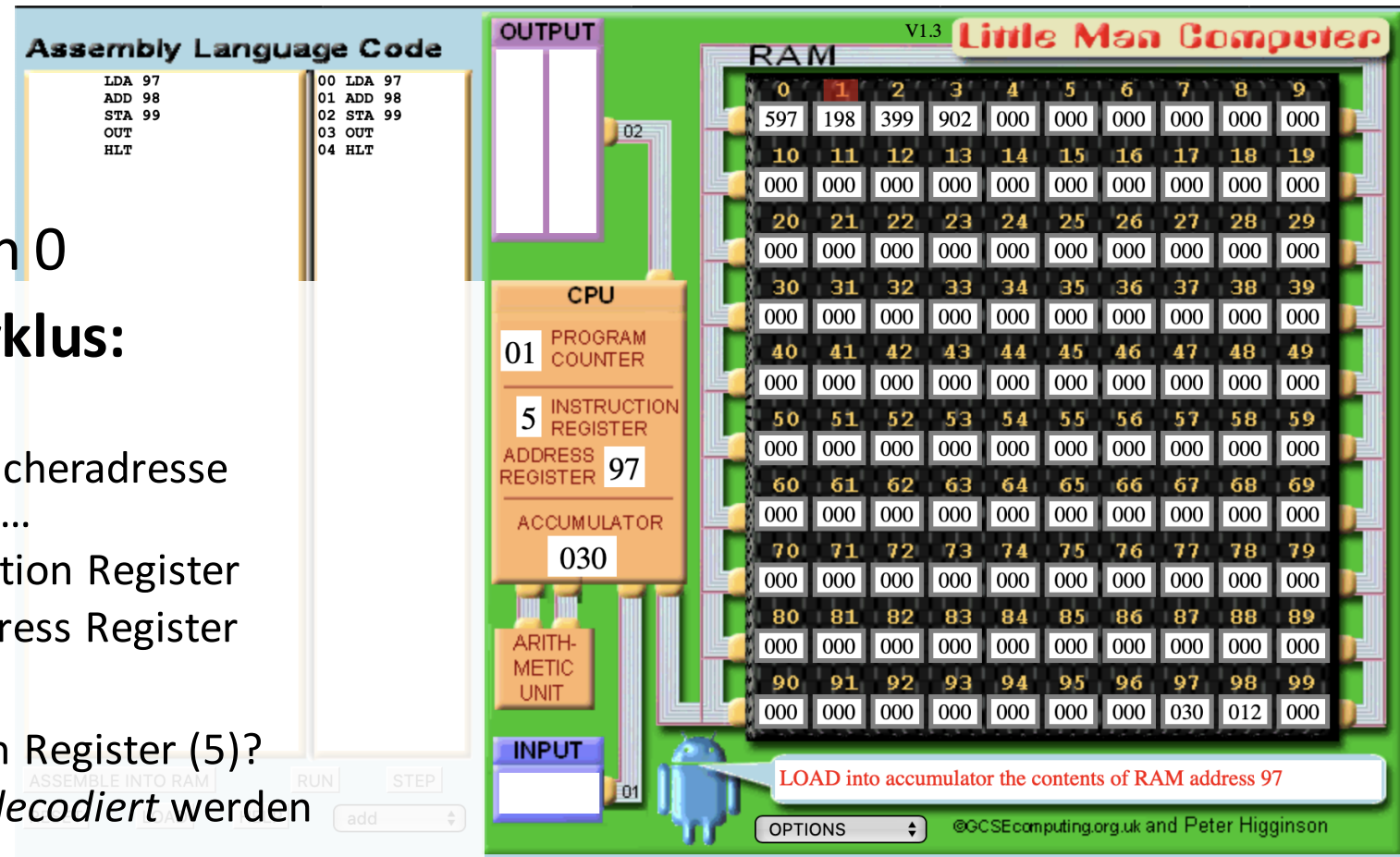
Von-Neumann-Instruktionszyklus

- Programmcode und Daten sind im gleichen Speicher ...
- ... getrennt durch *erstes 000* (HLT)
- Dieses sagt CPU, wann Ende des Programms erreicht wurde
- Wollen jetzt V-N-Zyklus genau anschauen



Zeile 1: LDA 97

- Program Counter ist anfänglich 0
- **Von-Neumann-Instruktionszyklus:**
 1. **Fetch:**
 - Hole (*fetche*) Instruktion an Speicheradresse (Program Counter), also 597 un ...
 - schreibe erste Zahl (5) in Instruction Register
 - schreibe zweite Zahl (97) in Address Register
 2. **Decode:**
 - Was bedeutet Zahl in Instruction Register (5)?
 - 5 ist ein Code und muss zuerst *decodiert* werden
 - CPU ermittelt, dass 5 bedeutet:
«**Lade in Akkumulator (5)** den Wert an der im Address Register angegebenen Speicherposition (97)»
 3. **Execute:** führe (*execute*) diesen Befehl aus
- In Zwischenzeit wurde Programcounter um 1 erhöht ...
- Und V-N-Zyklus beginnt von vorne für nächste Instruktion



Auftrag

Siehe Wiki

Lektion 4

Computerarchitektur

Loops & Jumps mit LMC

Schleifen und Co.

- Essenziell fürs Programmieren:
 - Schleifen (while, for)
 - Verzweigungen (if-elif-else)
 - Funktionen
- Diese Elemente gibt es in praktisch allen Programmiersprachen
- Doch wie realisiert man diese Elemente in Assemblersprache / Maschinensprache?
- **Sprünge!**
- Springe zu einer Zeile Code

Schleife

- Führe gleichen Codeblock mehrfach aus, indem du danach wieder an gleiche Zeile zurückspringst.

```
x = input("Enter a positive number")
while x <= 0:
    x = input("Enter a positive number")

print("Congratulations, you entered a positive number!")
```

Verzweigung

- Zeilen Code überspringen.

```
age = int(input("Enter your age"))

if age >= 18:
    print("Schnaps!")
elif age >= 16:
    print("Beer!")
else:
    print("Sirup!")
```

Funktion

- Springe zu Zeile irgendwo im Code, führe Codeblock aus und springe nachher wieder zurück.

```
def linear_search(li,el):  
    i = 0  
    while i < len(li):  
        if el == li[i]:  
            return i  
        i = i + 1  
    return None  
  
print(linear_search(['Adam', 'Berta', 'Christine', 'Dagobert'], 'Berta'))
```

Branch

- Alles das kann mithilfe der **Branch**-Befehle im LMC umgesetzt werden
- Branch: Ast, to branch: abzweigen
- LMC: BRA, BRZ, BRP
- BRA 4: Code springt zu Programmzeile 4
- BRZ 4 oder BRP 4: Springt ebenfalls zu Programmzeile 4 aber nur, *falls* eine Bedingung erfüllt ist (siehe Befehlssatz).



Auftrag

Siehe Wiki

Lektion 6

Computerarchitektur

Zeichentabellen & Symbole im LMC

Symbole

- Computer speichert und rechnet nur mit (Binär)Zahlen.
- Wie kann man mit Computer nun Symbole (z.B. Buchstaben) darstellen?
- Ideen?
- Trick: Weise jedem Symbol einen Wert zu.
- -> **Zeichentabelle**
- Beispiel: ASCII ----->

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

ASCII & Unicode

- ASCII: 7-Bit -> 128 Symbole
- Zum Beispiel: Grossbuchstaben: 65 bis und mit 90
- Unicode:
 - Erweiterung von ASCII
 - Identisch in ersten 128 Zeichen
 - Ca. 150'000 Symbole (2023)
- Werden ASCII & Unicode in Zusammenhang mit anderem Thema genauer anschauen

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Symbole im LMC

- Kann mit LMC Unicode-Symbole ausgeben ...
- ... zumindest die ersten 1000 Symbole
- Grund: Max. Zahl ist 999
- Befehl OTC gibt Unicode-Symbol von aktuellem Wert im Akkumulator aus

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Auftrag

- Gebe "KSR" aus mit LMC.
- Danach weiter wie angegeben auf Wiki

