

BSSEMBLER

EIN EXPERIMENT ZUR

COMPUTER EMULATION

Inhaltsverzeichnis

CPU-Emulator	Seite 3
B-ssembler und dessen Syntax	Seite 4
Speicheradressierung	Seite 6
Programmbeispiele	Seite 7
Übersicht der Speicheradressen	Seite 9
Ergebnis des Projekts	Seite 10

CPU Emulator

Es wird ein virtuelles Computer-System in C# programmiert; eine CPU und ein Arbeitsspeicher.

CPU:

Die CPU liest jeweils in Sequenzen und seriell 8 Byte aus. Diese 8 Byte setzen sich zusammen aus den ersten 2 Byte als Buchstaben, welche die Befehle darstellen wie **AD** für Addition, **SB** für Subtraktion oder **DC** für Deklaration. Die darauffolgenden 2* 3 Byte definieren Speicheradressen. Die CPU übernimmt den logischen und arithmetischen Anteil des Systems, sowie alles weitere verarbeitende.

RAM (Arbeitsspeicher):

Der Arbeitsspeicher besteht aus einer Ansammlung von Speicheradressen. Dadurch, dass die CPU 2 Byte für Operator, 3 Byte für die linke Speicheradresse und 3 Byte für die rechte Speicheradresse benötigt, wird der RAM aus dem Faktor 8 bestehen. Ein Programm kann aus maximal (RAM Größe / 8) Zeilen bestehen.

Da eine Zeile aus 8 Byte besteht, ist die Speicheradressenzuordnung Faktor 8, ergo $(8*8)*512 = 32.768$ Byte (32KB), somit 4096 Speicheradressen.

Die Gesamtanzahl der Zeilen eines Programmcodes bestehen also aus der Gesamtanzahl der Bytes im RAM geteilt durch 8, da eine Programmcode-Zeile immer aus 8 Byte besteht, somit $4096/8 = 512$ Zeilen Programmcode.

Diese besteht aus 8 Byte, da die Buchstaben der ersten drei Zeichen pro Zeile aus ASCII-Symbolen bestehen, welche 8 Bit (also 1 Byte) benötigen.

0	0	0	32	100000	20	64	@	1000000	40	96	1100000	60	128	€	10000000	80	160	10100000	A0	192	À	11000000	C0	224	à	11100000	E0				
1	1	1	33	!	100001	21	65	A	1000001	41	97	a	1100001	61	129		10000001	81	161	ì	10100001	A1	193	À	11000001	C1	225	à	11100001	E1	
2	1	10	2	34	"	100010	22	66	B	1000010	42	98	b	1100010	62	130	,	10000010	82	162	¢	10100010	A2	194	À	11000010	C2	226	à	11100010	E2
3	L	11	3	35	#	100011	23	67	C	1000011	43	99	c	1100011	63	131	f	10000011	83	163	£	10100011	A3	195	À	11000011	C3	227	à	11100011	E3
4	J	100	4	36	\$	100100	24	68	D	1000100	44	100	d	1100100	64	132	"	10000100	84	164	¤	10100100	A4	196	À	11000100	C4	228	à	11100100	E4
5		101	5	37	%	100101	25	69	E	1000101	45	101	e	1100101	65	133	...	10000101	85	165	¥	10100101	A5	197	À	11000101	C5	229	à	11100101	E5
6	-	110	6	38	&	100110	26	70	F	1000110	46	102	f	1100110	66	134	†	10000110	86	166	!	10100110	A6	198	À	11000110	C6	230	¤	11100110	E6
7	*	111	7	39	'	100111	27	71	G	1000111	47	103	g	1100111	67	135	‡	10000111	87	167	§	10100111	A7	199	Ç	11000111	C7	231	ç	11100111	E7
8	■	1000	8	40	(101000	28	72	H	1001000	48	104	h	1100000	68	136)	100001000	88	168)	10101000	A8	200	È	11001000	C8	232	è	11101000	E8
9	001	9	41)	101001	29	73	I	1001001	49	105	i	11001001	69	137	%	100001001	89	169	©	10101001	A9	201	È	11001001	C9	233	é	11101001	E9	
10	1010	A	42	*	101010	2A	74	J	1001010	4A	106	j	11001010	6A	138	Š	100001010	8A	170	ª	10101010	AA	202	È	11001010	CA	234	è	11101010	EA	
11	z	1011	B	43	+	101011	2B	75	K	1001011	4B	107	k	1100111	6B	139	«	100001101	88	171	«	10101011	AB	203	È	11001011	CB	235	è	11101011	EB
12	□	1100	C	44	,	101000	2C	76	L	1001000	4C	108	l	1100100	6C	140	Œ	100001100	8C	172	¬	10101000	AC	204	Î	11001100	CC	236	î	11101100	EC
13	1101	D	45	-	101101	2D	77	M	1001101	4D	109	m	1101101	6D	141		10000101	8D	173	-	10101101	AD	205	Î	11001101	CD	237	î	11101101	ED	
14	¶	1110	E	46	.	101100	2E	78	N	1001110	4E	110	n	1101110	6E	142	Ž	100001110	8E	174	®	10101110	AE	206	Î	11001110	CE	238	î	11101110	EE
15	⌘	1111	F	47	/	101111	2F	79	O	1001111	4F	111	o	1101111	6F	143		100001111	8F	175	¬	10101111	AF	207	Î	11001111	CF	239	î	11101111	EF
16	+	10000	10	48	0	110000	30	80	P	1010000	50	112	p	1100000	70	144		100000000	90	176	°	101000000	B0	208	Ð	110000000	D0	240	ð	111000000	F0
17	◀	10001	11	49	1	100001	31	81	Q	1010001	51	113	q	1100001	71	145	‘	100000001	91	177	±	101000001	B1	209	Ñ	110000001	D1	241	ñ	111000001	F1
18	↑	10010	12	50	2	110000	32	82	R	1010010	52	114	r	1100000	72	146	’	100000010	92	178	×	101000010	B2	210	Î	110000010	D2	242	ò	111000010	F2
19	!!	10011	13	51	3	110001	33	83	S	1010001	53	115	s	1100001	73	147	”	100000011	93	179	³	101000011	B3	211	Ô	110000011	D3	243	ô	111000011	F3
20	¶	10100	14	52	4	110000	34	84	T	1010000	54	116	t	1100000	74	148	”	100000010	94	180	·	101000010	B4	212	Ô	110000010	D4	244	ô	111000010	F4
21	↓	10101	15	53	5	101001	35	85	U	101001	55	117	u	1100001	75	149	•	100000010	95	181	µ	101000010	B5	213	Ô	110000010	D5	245	ô	111000010	F5
22	τ	10110	16	54	6	110010	36	86	V	1010100	56	118	v	1100000	76	150	-	100000010	96	182	¶	101000010	B6	214	Ô	110000010	D6	246	ô	111000010	F6
23	↓	10111	17	55	7	110000	37	87	W	1010000	57	119	w	1100000	77	151	-	100000011	97	183	·	101000011	B7	215	×	110000011	D7	247	+	111000011	F7
24	↑	11000	18	56	8	110000	38	88	X	1010000	58	120	x	1100000	78	152	”	100000000	98	184	:	101000000	B8	216	Ø	110000000	D8	248	ø	111000000	F8
25	†	11001	19	57	9	110001	39	89	Y	1010001	59	121	y	1100001	79	153	™	100000001	99	185	·	101000001	B9	217	Ù	110000001	D9	249	ù	111000001	F9
26	→	11010	1A	58	:	110000	3A	90	Z	1010000	5A	122	z	1100000	7A	154	š	100000010	9A	186	º	101000010	BA	218	Ù	110000010	DA	250	ù	111000010	FA
27	←	11011	1B	59	:	110000	3B	91	[1010001	5B	123	{	1100001	7B	155	»	100000011	9B	187	»	101000011	BB	219	Ù	110000011	DB	251	ù	111000011	FB
28	11000	1C	60	<	110000	3C	92	\	1010000	5C	124		1100000	7C	156	œ	100000000	9C	188	¼	101000000	BC	220	Ù	110000000	DC	252	ù	111000000	FC	
29	11101	1D	61	=	110000	3D	93]	1010000	5D	125	}	1100001	7D	157		100000011	9D	189	½	101000011	BD	221	Ý	110000011	DD	253	ý	111000011	FD	
30	11110	1E	62	>	110000	3E	94	^	1010000	5E	126	~	1100000	7E	158	ž	100000010	9E	190	¾	101000010	BE	222	Þ	110000010	DE	254	þ	111000010	FE	
31	11111	1F	63	?	110000	3F	95	_	1010000	5F	127	111111	7F	159	Ý	100000011	9F	191	¸	101000011	BF	223	Ù	110000011	DF	255	ù	111000011	FF		

B-ssembler

Die Programmiersprache B-ssembler bildet die Schnittstelle zwischen Soft- und Hardware. Mittels vier substantieller Befehle werden die Bytes an den jeweiligen Speicheradressen bearbeitet. Diese sind:

DC: „Declare“	Hiermit deklarieren wir an einer bestimmten Speicheradresse einen Wert. <i>Syntax: DC [Speicheradresse] [Wert]</i> Bsp.: DC 200 001
AD: „Add“	Hiermit addieren wir den Wert auf Speicheradresse A mit dem auf Speicheradresse B. <i>Syntax: AD [Speicheradresse A] [Speicheradresse B]</i> Bsp.: AD 200 240
SB: „Subtract“	Hiermit subtrahieren wir den Wert auf B von dem auf A. <i>Syntax: SB [Speicheradresse A] [Speicheradresse B]</i> Bsp.: SB 210 200
MP: „Multiply“	Hiermit multiplizieren wir den Wert auf Adresse A mit dem Wert auf B. <i>Syntax: MP [Speicheradresse A] [Speicheradresse B]</i> Bsp.: MP 0F1 0D2
DV: „Divide“	Hiermit dividieren wir den Wert auf Adresse A durch den Wert auf B. <i>Syntax: DV [Speicheradresse A] [Speicheradresse B]</i> Bsp.: DV 3b8 0FF
CP: „Copy“	Hiermit kopieren wir den auf Adresse A gespeicherten Wert auf die Adresse B. <i>Syntax: CP [Speicheradresse A] [Speicheradresse B]</i> Bsp.: MV 210 218
MV: „Move“,	Hiermit bewegen wir den auf Adresse A gespeicherten Wert auf die Adresse B. <i>Syntax: MV [Speicheradresse A] [Speicheradresse B]</i> Bsp.: MV 02E 0b3
#: Kommentar	Zeilen mit einem „#“ am Anfang werden ignoriert und können bspw. Zum Kommentieren verwendet werden. Dafür MUSS das Zeichen am Anfang stehen, sonst funktioniert es nicht. <i>Syntax: #[Text]</i> Bsp.: #Funktionserklärung
GOTO: Sprungzeile	Hiermit springen wir zu der Zeile, die das auf die Anweisung folgende Argument enthält. Das Argument muss hierbei am gewünschten Punkt im Code, zu dem gesprungen werden soll, vorhanden sein, angeführt von einem „:“. <i>Syntax: GOTO [Argument]</i> Bsp.: GOTO JUMP :[Argument] :JUMP
IF:	Hiermit deklarieren wir eine Bedingung. Nur wenn diese Bedingung erfüllt ist, wird der darauf folgende Code ausgeführt. Operatoren: „=“ (gleich), „<“ (kleiner als), „>“ (größer als), „!“ (ist nicht). <i>Syntax: IF: [Bedingung] [Befehl] ENDIF</i> Bsp.: IF: 03C > 0FF GOTO ENDE ENDIF

ENDIF: Schließt einen vorangegangenen IF-Block, siehe oben.

INPUT: Startet eine Eingabeaufforderung, mit der Datensätze in den Speicher ab der darauf folgend angegebenen Adresse geschrieben werden.

Syntax: INPUT [Adresse] Bsp.: INPUT 2F8
[Eingabeaufforderung]

FR: „File Read“ Zunächst wird über INPUT der Dateipfad der gewünschten Datei angegeben und in den RAM geschrieben. Er wird automatisch mit einem ASCII-EOT* am Ende versehen – so weiß der Prozessor, dass dort der Inhalt aufhört. Mit **FR** und der zuvor bei INPUT angegebenen Adresse liest der Prozessor zunächst den Dateinamen bis zum EOT aus und beginnt dann, den Dateiinhalt zu lesen. Dieser Inhalt wird ab der auf den EOT folgenden Adresse in den RAM gespeichert und am Ende wird wieder ein EOT angefügt. Von nun an ist die Datei im RAM verfügbar.

Syntax: FR [Startadresse] | Bsp.: FR 218

FW: „File Write“ Speichert einen zuvor mit FR im RAM hinterlegten Dateiinhalt unter dem bei der Namensadresse hinterlegten Namen. Der Dateiinhalt wird ab der angegebenen Datenadresse bis zum nächsten EOT bezogen.
Die Namensadresse muss hierbei nicht die Adresse des ursprünglich bei FR hinterlegten Dateinamens sein.

Syntax: FW [Namensadresse] [Datenadresse] | Bsp.: FW 200 490

*EOT = End of Transmission

OP: „Output“ Hiermit schreibt man einen Buchstaben/ein Symbol auf den Konsolenbildschirm.

Syntax: OP [x-Byte] [y-Byte] [Farb-Byte] [ASCII]-Byte] | Bsp.: OP FC

OC: „Output Char“ Rechnet den HEX-Wert der Speicheradresse zu einem ASCII-Symbol um und gibt dieses aus.

Syntax: OC [Speicheradresse]

HALT: Beendet das Programm

Beendet das FR
Syntax: *HALT*

STRINGBEFEHLE:	OP \$[Text]	Gibt Text aus statt Adresse. Maximal 5 Zeichen nach „\$“
	\$S	Gibt ein Leerzeichen aus.
	\$N	Gibt einen Zeilenumbruch aus.
	\$T	Gibt einen Tabulator aus.
	OP NUM [Adresse]	zeigt die HEX-Zahl auf der gegebenen Adresse als normale Zahl an

COLOR: Setzt die Vorder- und Hintergrundfarbe des Konsolenfensters.

Syntax: COLOR [Background-BYTE][Foreground-BYTE]

Bsp.: COLOR AF

Steht ein X an der jeweiligen Stelle wird dieser Wert nicht neu gesetzt

Bsp.: COLOR XF → setzt nur den Foreground-Color-Wert

COLOR XF → setzt nur den Foreground-Color-Wert.
COLOR AX → setzt nur den Background-Color-Wert

CR: „Cursor“ Setzt den Cursor auf die Position des Wertes der angegebenen Adresse im Arbeitsspeicher.
Syntax: CR [X-Adresse] [Y-Adresse]

MANP: „Manipuliert“ Manipuliert einen String-Wert im RAM und ändert diesen zum angegebenen Character.
Syntax: **MANP [Adresse] [Char]** | Bsp.: **MANP 008 @**
→ setzt den Char an Adresse 008 zu einem @-Symbol.

§: Variablen Eine Variable ist ein Platzhalter für eine Adresse und den auf ihr hinterlegten Wert. Mit § gefolgt von einer zweistelligen Bezeichnung und der Bezugsadresse kann man in B-ssembler Variablen deklarieren.
Syntax: §[Name]=[Adresse] | Bsp.: §0A=210

Verwendung: IF: DC \$0A 002
 §0A = §0B AD \$0a \$0B
 [Befehl]
 ENDIF

Gültige Variablen sind §00 → §FF (256 Variablen).

DB: „Debug“ Listet den Arbeitsspeicher grafisch auf.

Speicheradressierung (Logisch):

Der Prozessor reserviert die ersten 1024 Bit für Variablen. Die Anzahl der Variablen ergibt sich aus der Anzahl der Byte geteilt durch 4, somit 256 Variablen. Errechnet werden die tatsächlichen Adressen der Variablen durch das Multiplizieren mit 4.

Beispiel:

Adresse §00 (Dezimal 0)	$\rightarrow 4 * 0$	= Adresse 0x 000 \rightarrow 0x 003
Adresse §01 (Dezimal 1)	$\rightarrow 4 * 1$	= Adresse 0x 004 \rightarrow 0x 007
Adresse §0A (Dezimal 10)	$\rightarrow 4 * 10$	= Adresse 0x 028 \rightarrow 0x 02B
Adresse §FE (Dezimal 254)	$\rightarrow 4 * 254$	= Adresse 0x 3F8 \rightarrow 0x 3FB

Debug Output:

Code Arbeitsspeicher

	0000	F	0001	F	0002	F	0003	0
1	§00=FFF		0004	F	0005	F	0006	F
2	§01=FFF		0028	F	0029	F	002a	F
3	§0A=FFF		03f8	F	03f9	F	03fa	F
4	§FE=FFF						03fb	0
5								
6	DB							

Einsteigerbeispiel

Hier ein Beispiel eines B-ssembler-Programms, das Fibonacci-Zahlen berechnet und ausgibt:

Code	Ergebnis
DC 210 000	
DC 218 001	HEX FIB
DC 2F8 030	001 1
:JUMP	032 2
AD 210 218	003 3
AD 210 2F8	035 5
OP 210	008 8
SB 210 2F8	03d 13
AD 218 210	015 21
AD 218 2F8	052 34
OP 218	037 55
SB 218 2F8	089 89
GOTO JUMP	
HALT	

Beispielprogramme:

„Timer“

```

DC 200 000 Deklariere auf 200 Wert 0
DC 208 001 Deklariere auf 208 Wert 1
:RTIM Sprungzeile für Reset-Timer
CP F28 200 Kopiere System-Sekunden zu Adresse 200
AD 200 208 Addiere 1 auf Adresse 200 (System-Sekunden)

:TIME Sprungzeile für Timer

IF: Wenn Adresse 200 (System-Sekunden + 1S) kleiner als aktuelle Sekunde
200<F28
  OP $Tic$N Gebe „Tic“ mit Zeilenumbruch aus
  GOTO RTIM Gehe zu Reset-Timer Sprungzeile
ENDIF

(Gandernfalls)
GOTO TIME Gehe zu Timer Sprungzeile zurück

```

ACHTUNG! Ist die System-Sekunde zur Laufzeit bei 59, läuft das Programm unendlich weiter, da es eine Sekunde in der Zukunft erst ausgelöst wird, und die System-Sekunde NIE 60s erreicht!

„ASCII-Tabelle“

```

#ASCII-Zähler, #Umbruchszähler
#Umbruchszaehler
DC 200 000
DC 208 000

#Zahl fuer Addition (Inkrementierung)
DC 210 001

#Maximaler Wert, bis wohin die ASCII-Liste geht
DC 218 OFF

#Maximaler Wert: Wann Umbruch
DC 228 008

#Sonderzeichen Umbruch, Doppelpunkt und Tabulator
DC 230 00A
DC 238 03A
DC 240 009

:LOOP
#Output der Werte:
OC 240
OP 200
OC 238
OC 240
OC 200

#Wenn Umbruch-Zähler grösser als 8, mache einen Umbruch und setze den Umbruch-Zähler auf 0
```

```

IF:
  208=228
  DC 208 000
  OC 230
ENDIF

```

```
#Wenn der ASCII-Zähler OFF (256) erreicht hat, beende Programm
```

```

IF:
  200=218
  HALT
ENDIF

```

```
#Inkrementiere die beiden Zähler
AD 200 210
AD 208 210
GOTO LOOP
```

„6-Stelliger Zähler“

```

#Der Maximalwert
DC 3F8 FFF

#Der rechte Zaehler
#Der linke Zaehler
DC 3F0 000
DC 3E8 000
DC 3E0 001

DC 280 00A

:LOOP
OP 3E8
OP 3F0

OC 280

IF:
  3F0=3F8
  AD 3E8 3E0
  DC 3F0 000
#Linker Zaehler +001
#Rechter Zaehler auf 0 setzen
ENDIF

AD 3F0 3E0

IF:
  3E8 = 3F8
  HALT
ENDIF
```

```
GOTO LOOP
```

System-Speicheradressen

Die System-Speicheradressen werden vom System selbst beschrieben und beinhalten System-informationen über Datum und Zeit

Adresse	Beschreibung	Beispielwert
F00 → F01	Tag	19
F08 → F09	Monat	11
F10 → F13	Jahr	2022
F18 → F19	Stunde	20
F20 → F21	Minute	15
F28 → F29	Sekunde	45

Sichere Speicheradressen:

(Gesamt RAM / 8)

000, 008, 010, 018, 020, 028, 030, 038, 040, 048, 050, 058,	060, 068, 070, 078, 080, 088, 090, 098,
0a0, 0a8, 0b0, 0b8, 0c0, 0c8, 0d0, 0d8, 0e0, 0e8, 0f0, 0f8,	100, 108, 110, 118, 120, 128, 130, 138,
140, 148, 150, 158, 160, 168, 170, 178, 180, 188, 190, 198,	1a0, 1a8, 1b0, 1b8, 1c0, 1c8, 1d0, 1d8,
1e0, 1e8, 1f0, 1f8, 200, 208, 210, 218, 220, 228, 230, 238,	240, 248, 250, 258, 260, 268, 270, 278,
280, 288, 290, 298, 2a0, 2a8, 2b0, 2b8, 2c0, 2c8, 2d0, 2d8,	2e0, 2e8, 2f0, 2f8, 300, 308, 310, 318,
320, 328, 330, 338, 340, 348, 350, 358, 360, 368, 370, 378,	380, 388, 390, 398, 3a0, 3a8, 3b0, 3b8,
3c0, 3c8, 3d0, 3d8, 3e0, 3e8, 3f0, 3f8, 400, 408, 410, 418,	420, 428, 430, 438, 440, 448, 450, 458,
460, 468, 470, 478, 480, 488, 490, 498, 4a0, 4a8, 4b0, 4b8,	4c0, 4c8, 4d0, 4d8, 4e0, 4e8, 4f0, 4f8,
500, 508, 510, 518, 520, 528, 530, 538, 540, 548, 550, 558,	560, 568, 570, 578, 580, 588, 590, 598,
5a0, 5a8, 5b0, 5b8, 5c0, 5c8, 5d0, 5d8, 5e0, 5e8, 5f0, 5f8,	600, 608, 610, 618, 620, 628, 630, 638,
640, 648, 650, 658, 660, 668, 670, 678, 680, 688, 690, 698,	6a0, 6a8, 6b0, 6b8, 6c0, 6c8, 6d0, 6d8,
6e0, 6e8, 6f0, 6f8, 700, 708, 710, 718, 720, 728, 730, 738,	740, 748, 750, 758, 760, 768, 770, 778,
780, 788, 790, 798, 7a0, 7a8, 7b0, 7b8, 7c0, 7c8, 7d0, 7d8,	7e0, 7e8, 7f0, 7f8, 800, 808, 810, 818,
820, 828, 830, 838, 840, 848, 850, 858, 860, 868, 870, 878,	880, 888, 890, 898, 8a0, 8a8, 8b0, 8b8,
8c0, 8c8, 8d0, 8d8, 8e0, 8e8, 8f0, 8f8, 900, 908, 910, 918,	920, 928, 930, 938, 940, 948, 950, 958,
960, 968, 970, 978, 980, 988, 990, 998, 9a0, 9a8, 9b0, 9b8,	9c0, 9c8, 9d0, 9d8, 9e0, 9e8, 9f0, 9f8,
a00, a08, a10, a18, a20, a28, a30, a38, a40, a48, a50, a58,	a60, a68, a70, a78, a80, a88, a90, a98,
aa0, aa8, ab0, ab8, ac0, ac8, ad0, ad8, ae0, ae8, af0, af8,	b00, b08, b10, b18, b20, b28, b30, b38,
b40, b48, b50, b58, b60, b68, b70, b78, b80, b88, b90, b98,	ba0, ba8, bb0, bb8, bc0, bc8, bd0, bd8,
be0, be8, bf0, bf8, c00, c08, c10, c18, c20, c28, c30, c38,	c40, c48, c50, c58, c60, c68, c70, c78,
c80, c88, c90, c98, ca0, ca8, cb0, cb8, cc0, cc8, cd0, cd8,	ce0, ce8, cf0, cf8, d00, d08, d10, d18,
d20, d28, d30, d38, d40, d48, d50, d58, d60, d68, d70, d78,	d80, d88, d90, d98, da0, da8, db0, db8,
dc0, dc8, dd0, dd8, de0, de8, df0, df8, e00, e08, e10, e18,	e20, e28, e30, e38, e40, e48, e50, e58,
e60, e68, e70, e78, e80, e88, e90, e98, ea0, ea8, eb0, eb8,	ec0, ec8, ed0, ed8, ee0, ee8, ef0, ef8,
f00, f08, f10, f18, f20, f28, f30, f38, f40, f48, f50, f58,	f60, f68, f70, f78, f80, f88, f90, f98,
fa0, fa8, fb0, fb8, fc0, fc8, fd0, fd8, fe0, fe8, ff0, ff8	

ACHTUNG!: Sicher im Sinne der Adressierung, nicht der Persistenz oder der Gewähr, andere Daten nicht zu überschreiben!

Ergebnis des Projektes

Es wurde erfolgreich ein Computersystem mit einfacher Arithmetik, Speicheradressenlogik und Dateiein- und -ausgabe erstellt. Die Ergebnisse auf Bezug der CPU sind außergewöhnlich positiv und stabil in der Ausführung, wohingegen die Arbeitsspeicherverwaltung starke Formfaktorenfehler besitzt, bei denen Adressen zum Faktor 8 vergeben werden und oftmals fünf Adressen pro Adressenfaktor leer bleiben. So zum Beispiel wird Adresse 200, 201 und 202 belegt, Adresse 203 bis 207 bleiben aber leer. Dieser Missstand kann behoben werden durch gezielte Speicheradressenanpeilung mit manueller Vergebung, ist allerdings nicht elegant (aber möglich). Die Geschwindigkeit des Programmcodes beträgt ungefähr 20.000 Rechenschrittadditionen pro Sekunde.



Viktoria Schmieder
&
Fabian Müller